THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm biên soạn và Giảng viên có đóng góp ý kiến: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Lê Ngọc Thành – Phạm Trọng Nghĩa - Nguyễn Công Nhựt – Trần Ngọc Việt - Hoàng Thị Kiều Anh – Huỳnh Thái Học

TP.HCM – Năm 2019

MỤC LỤC

BÀI 1: CƠ SỞ LOGIC VÀ TẬP HỢP

Mục tiêu:

- *Nắm vững được Python để viết các đoạn lệnh xử lý về: mệnh đề, logic, đúng/sai.*
- *Sử dụng tốt công cụ xử lý trên tập hợp, bao gồm: định nghĩa và các phép toán.*

Nội dung chính:

1. Các phép toán luận lý trong Python

1.1. Luận lý trong Python

Python có kiểu dữ liệu luận lý là True và False cho các phép xử lý là == (so sánh bằng), != (so sánh khác), <, >, is (là), is not và các phép toán liên quan là: and, or, not, ^ (XOR).

Ví dụ:

```
>>> a = True

>>> b = False

>>> a and b

False

>>> a or b

True

>>> a ^ b

True
```

1.2. Biểu thức điều kiện if

Trong Python, biểu thức điều kiện if else là một trong cấu trúc rẽ nhánh cơ bản. Cấu trúc của lệnh if như sau:

```
>>> if (điều kiện 1):
        # khối lệnh xử lý điều kiện 1
elif (điều kiện 2):
        # khối lệnh xử lý điều kiện 2
```

elif (*điều kiện 3…*):

    # khối lệnh xử lý *điều kiện 3…*

else: # trường hợp còn lại

    # khối lệnh xử lý *các trường hợp còn lại*

Việc sử dụng hiệu quả cấu trúc sẽ dẫn đến chương trình tinh gọn. Sinh viên thực hành lệnh dưới đây:

>>> *def* **kiemtra_nuocsoi**(nhiet_do):

    if nhiet_do < 100:

        return "Nuoc chua soi!"

    else:

        return "Nuoc da soi!"

>>> kiemtra_nuocsoi(100)

……..……..…… ← sinh viên ghi kết quả.

>>> kiemtra_nuocsoi(99)

……..……..…… ← sinh viên ghi kết quả.

Sau đó, sinh viên thử một cách viết hàm tinh gọn như sau:

```
>>> def kiemtra_nuocsoi1(nhiet_do):
        return 'Nuoc ' + ('da soi!'  if nhiet_do == 100 else 'chua soi.')
```

Và thực hiện các kiểm tra:

>>> kiemtra_nuocsoi1(100)

……..……..…… ← sinh viên ghi kết quả.

>>> kiemtra_nuocsoi1(99)

……..……..…… ← sinh viên ghi kết quả.

Hơn thế nữa, lệnh rẽ nhánh if được sử dụng trong "thiết kế" danh sách các giá trị. Sinh viên thực hiện các lệnh sau:

- Xây dựng danh sách gồm các phần tử thỏa điều kiện >5:

```
>>> for diem_so in range(10):
        if diem_so > 5:
            ds_dau.append(str(diem_so))


>>> ds_dau
……..……..……  ← sinh viên ghi kết quả.
```

Và đoạn code ngắn gọn:

```
>>> ds_dau = [str(diem_so) for value in range(10) if diem_so > 5]
>>> ds_dau
……..……..……  ← sinh viên ghi kết quả.
```

1.3. Thứ tự tính toán trong Python

Bên cạnh đó, là một ngôn ngữ lập trình, các trình biên dịch Python thường xử lý phép toán từ trái sang phải (left-to-right order). Theo đó, các biểu thức logic sẽ bị ảnh hưởng bởi thứ tự tính toán. Ví dụ: trong phép toán AND, nếu yếu tố đầu tiên không thỏa thì ngay lập tức biểu thức sẽ mang giá trị False (sai), nhằm giảm thiểu các tính toán không cần thiết; hoặc với phép toán OR, nếu biểu thức đầu tiên thỏa thì các biểu thức phía sau không cần tính toán.

Sinh viên thực hành các lệnh sau để hiểu được thứ tự tính toán trong Python:

```
>>> a = True
>>> b = False
>>> 1/0

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    1/0
ZeroDivisionError: integer division or modulo by zero
>>> if (a == b) and (1/0 > 0):
        print ("a=b")
else:
        print ("a khac b")
```

Kết quả câu lệnh if là: ……..

Sinh viên thực hành đoạn lệnh khác:

```
>>> a = True
>>> b = False
>>> if (1/0 > 0) and (a == b):
        print ("a=b")
else:
        print ("a khac b")
```

Sinh viên hãy ghi nhận kết quả đoạn lệnh trên và giải thích.

……..……..……..……..

2. Dữ liệu dạng tập hợp trong Python: Set

Python cung cấp những cú pháp đơn giản để thể hiện các khái niệm về dữ liệu rời rạc cơ bản như tập hợp, … cùng với các kỹ thuật tính toán đếm và tổ hợp trên đó. Cụ thể:

- **Phép toán trên tập hợp (set operators):**

Tập hợp và các phép toán trên tập là khái niệm cơ bản của toán rời rạc. Theo đó, Python hỗ trợ kiểu dữ liệu list và set đối tượng và các tập phép toán trên hai kiểu dữ liệu. Với list, đó là tập các đối tượng có thể sửa đổi.

Ví dụ 1: Tập các 20 số nguyên chẵn đầu tiên từ 0 theo toán học được định nghĩa là:

$$S = \{x | x = 2n; 0 \leq n \leq 19\}$$

Python sẽ mô tả tập (kiểu list) theo định nghĩa trên là: S = [2*x for x in range(20)]. Lệnh:

>>> S = [2*x for x in range(20)]

Sau đó, sinh viên hãy điền kết quả với lệnh sau:

>>> S

……..

Ví du 2: Bên cạnh **list**, Python hỗ trợ kiểu dữ liệu **set** với các phép toán so sánh tập. Sinh viên thực tập: So sánh tập nghiệm của phương trình $A = \{x | x^2 + x - 6 = 0\}$ và tập $B = \{-3,2\}$

>>> A = set([x for x in range(-50, 50) if x**2 + x - 6 == 0])

>>> B = set([2, -3])

>>> A == B

…… ← sinh viên ghi kết quả tại đây (gợi ý: True hoặc False).

Câu hỏi cho sinh viên: khoảng (-50, 50) có ý nghĩa gì trong lệnh trên? Thay thế khoảng khác được không? Lí do?

Sinh viên trả lời: ……..……

Ví dụ 3: Xây dựng tập các số nguyên tố nhỏ hơn 40.

Theo định nghĩa, số nguyên tố là số chỉ chia hết cho 1 và chính nó. Do vậy, trong Python, chúng ta có thể xây dựng một hàm kiểm tra, trả về **False** nếu số không thỏa điều kiện định nghĩa số nguyên tố, ngược lại trả về **True**.

```
>>> def isPrime(N):
        for i in [ x+1 for x in range(N) ]:
            if N % i == 0 and (i!=1 and i!=N):
                return False
        return True
```

Và sau đó, chúng ta có thể xây dựng tập hợp các số nguyên tố:

>>> S_prime = [isPrime(k) ***for*** *k* ***in*** **range**(1,40)]

Kết quả là:

```
>>> S_prime
[True, True, True, False, True, False, True, False, False, False, True, False, True, False, False, False,
True, False, True, False, False, False, True, False, False, False, False, False, True, False, True, False
, False, False, False, False, True, False, False]
>>> S_prime[37]
False
>>> S_prime[36]
True
```

Sinh viên thực hành:

- Số 36 không phải là số nguyên tố nên S_prime[36] = True, ngược lại số S_prime[37] = False.
- Sinh viên hãy thay thế lệnh S_prime trên bằng cách viết theo set như sau:

>>> S_prime = **set**([x for x in range(40) if (isPrime(x) == True and x>0)])

>>> S_prime

…… ← Sinh viên điền kết quả.

Ví dụ 4: Tạo tập tích từ hai tập (set product). Giả sử, chúng ta cần lấy 2 lần giá trị nghiệm, 20 lần giá trị nghiệm và 200 lần giá trị nghiệm của phương trình $x^2 + x - 6 = 0$. Chúng ta có thể viết lệnh như sau:

>>> Apro = set([2*x *y for x in range(-50,50) if x**2+x-6 == 0 for y in [1, 10, 100]])

>>> Apro

……← sinh viên điền kết quả.

Ví dụ 5: Ý tưởng tương tự ví dụ 4 nhưng tập tích được tạo giữ hai nhóm giá trị từ 2 tập nguồn:

>>> AB = set([(x, 2*y) for x in range(-50,50) if x**2+x-6 == 0 for y in [1, 10, 100]])

>>> AB

……← sinh viên điền kết quả.

3. Dữ liệu dạng tập hợp trong Sympy: FiniteSet

Trong kí hiệu toán học, chúng ta thể hiện các phần tử của tập hợp nằm trong dấu {} (curly brackets). Ví dụ : {3,5,7} là tập hợp thể hiện các phần của nó là: 3, 5, 7.

3.1. Xây dựng và các thao tác cơ bản trên tập hợp

3.1.1. Xây dựng tập hợp

Để tạo tập hợp trong gói sympy của Python, chúng ta có thể sử dụng lớp **FiniteSet** từ gói sympy như sau:

>>> from sympy import FiniteSet

>>> s = FiniteSet(3, 5, 7)

>>> s

{3, 5, 7}

Trong đoạn lệnh trên, đầu tiên chúng ta import lớp **FiniteSet** từ gói **SymPy** và sau đó tạo đối tượng từ lớp này bằng việc chuyển các phần tử của tập hợp. Chúng ta đặt tập hợp tên là s.

Để lưu trữ các loại số khác nhau, bao gồm số nguyên, số thực, phân số trong cùng một tập hợp như sau:

```
>>> from sympy import FiniteSet

>>> from fractions import Fraction

>>> s = FiniteSet(1, 1.5, Fraction(1, 5))

>>> s

{1/5, 1, 1.5}
```

Số lượng (cardinality) của một tập hợp là số lượng phần tử trong tập hợp đó. Hàm len() sẽ được sử dụng để đếm số lượng phần tử:

```
>>> s = FiniteSet(1, 1.5, Fraction(8, 2))

>>> s

{1, 1.5, 4}

>>> len(s)

3
```

3.1.2. Kiểm tra một số trong một tập hợp

Để kiểm tra sự tồn tại của một số trong tập hợp, chúng ta sử dụng toán tử `in`. Toán tử này sẽ trả về giá trị chân trị True (nếu tồn tại) hoặc False (nếu không tồn tại). Ví dụ: chúng ta có thể kiểm tra giá trị 8 và 1 có nằm trong tập s bên trên bằng lệnh sau:

```
>>> 8 in s

False

>>> 1 in s

True
```

3.1.3. Tạo tập hợp rỗng

Để tạo một tập rỗng (empty set), nghĩa là tập không có phần tử, chúng ta tạo đối tượng FiniteObject mà không cần đưa thông số vào. Lệnh như sau:

```
>>> from sympy import FiniteSet

>>> s = FiniteSet()

>>> s

EmptySet()
```

3.1.4. Tạo tập hợp từ List hoặc Tuple

Trong Python, một tập hợp có thể được tạo từ một List hoặc một Tuple như sau:

```
>>> phantu = [2, 4, 6, 8, 10]

>>> tap = FiniteSet(*phantu)

>>> tap

{2, 4, 6, 8, 10}
```

Ở đây, ta thấy: thay vì chuyển các danh sách phần tử trực tiếp vào tham số **FiniteSet** thì chúng ta có thể lưu trữ chúng trong một danh sách phantu (kiểu list). Sau đó, chúng ta có thể chuyển danh sách này vào trong FiniteSet bằng cú pháp Python đặc biệt như trên. Bằng phương pháp này, chúng ta sẽ sử dụng uyển chuyển khi viết các chương trình khi các phần tử của tập hợp được tính toán trong chương trình.

3.1.5. Loại bỏ các phần tử trùng và sắp thứ tự tập hợp

Kiểu tập hợp trong Python (như tập hợp trong toán học) sẽ loại bỏ những phần tử trùng và không quan tâm đến thứ tự của phần tử trong tập hợp (nghĩa là tập hợp có thể sắp xếp lại, không giữ thứ tự ban đầu). Ví dụ: khi chúng ta tạo 1 tập từ list có nhiều phần tử giống nhau, khi đó, các số được thêm vào trong tập hợp chỉ đúng 1 lần. Xét ví dụ sau:

```
>>> from sympy import FiniteSet

>>> phantu = [6, 7, 8, 9, 6, 7]

>>> taphop = FiniteSet(*phantu)

>>> taphop

{6, 7, 8, 9}
```

Rõ ràng danh sách ban đầu có 2 phần tử 6 và 7 có trùng nhau. Tuy nhiên, khi chúng ta đưa vào tập hợp thì chúng sẽ được loại bỏ một cách tự động.

Lưu ý: Trong Python, hai kiểu danh sách list và tuple sẽ lưu trữ mỗi phần tử theo thứ tự của mỗi phần tử. Tuy nhiên, với kiểu tập hợp thì điều đó không còn đúng nữa. Xét chương trình in dữ liệu từ tập trên:

```
>>> from sympy import FiniteSet

>>> phantu = [6, 7, 8, 9, 6, 7]

>>> taphop = FiniteSet(*phantu)

>>> taphop

{6, 7, 8, 9}

>>> for thanhphan in taphop:

        print(thanhphan)
```

…….. ← sinh viên điền vào kết quả

Thực hiện đoạn lệnh lặp in nhiều lần chúng ta sẽ thấy thứ tự các phần tử xuất ra không là vấn đề.

Khi đó, chúng ta sẽ có khái niệm là so sánh hai tập hợp. Hai tập hợp bằng nhau theo toán học khi chúng cùng phần tử. Trong Python, chúng ta có thể sử dụng phép toán == để so sánh hai tập hợp bằng nhau:

```
>>> from sympy import FiniteSet

>>> ds1 = [2, 4, 6]

>>> ds2 = [6, 2, 4]

>>> ds1 == ds2
```

…….. ← sinh viên điền kết quả

```
>>> s = FiniteSet(*ds1)

>>> t = FiniteSet(*ds2)

>>> s == t
```

…….. ← sinh viên điền kết quả

Kết quả cho thấy thứ tự của các phần tử không ảnh hưởng đến tập hợp.

3.2. Tập con (subset), tập cha (superset) và tập các tập con (power set)

Một tập s là tập con của tập t nếu mọi phần tử của s đều là phần tử của t. Ví dụ: tập {1} là tập con của tập {1, 2}. Việc kiểm tra tập con được thực hiện bằng lệnh **is_subset**().

```
>>> s = FiniteSet(1)

>>> t = FiniteSet(1,2)

>>> s.is_subset(t)

…….. ← sinh viên điền kết quả

>>> t.is_subset(s)

…….. ← sinh viên điền kết quả

>>> t.is_subset(t)

…….. ← sinh viên điền kết quả
```

Tương tự, ta gọi tập t là superset (tập cha) của tập s nếu t chứa tất cả các phần tử được chứa trong s. Lệnh kiểm tra là **is_superset**():

```
>>> s.is_superset(t)

…….. ← sinh viên điền kết quả

>>> t.is_superset(s)

…….. ← sinh viên điền kết quả
```

Tập các tập con (powerset) của một tập s là tập tất cả các tập con của s. Từ đó, với 1 tập, chúng ta sẽ có $2^{|s|}$phần tử (mỗi phần tử là 1 tập hợp) với $|s|$ là số phần tử của tập hợp. Ví dụ: tập {1,2,3} có 3 phần tử, do vậy, số lượng tập con của nó là $2^3 = 8$, bao gồm: {} (tập rỗng), {1}, {2}, {3}, {1,2}, {1,3}, {2,3} và {1,2,3}.

Kiểu dữ liệu tập FiniteSet cung cấp hàm để chúng ta xử lý, đó là hàm **powerset()**. Ví dụ:

```
>>> s = FiniteSet(1,2,3)

>>> ps = s.powerset()
```

>>> len(ps)

…….. ← sinh viên điền kết quả

>>> ps

…….. ← sinh viên điền kết quả

Bên cạnh đó, theo định nghĩa toán học, một tập vừa là tập con và vừa là tập cha của chính nó. Bên cạnh đó, phép so sánh tập cha "ngặt" hoặc tập con "ngặt" để đảm bảo số phần từ tập cha luôn lớn hơn số phần tử tập con. Ví dụ: tập s = {1,2,3} sẽ là tập cha "ngặt" của các tập như {1}, {2,3}, {1,3} và ngược lại, các tập như {1}, {2,3}, {1,3} có số phần tử ít hơn tập cha {1,2,3}. Quan hệ đó được thể hiện trong kiểu dữ liệu FiniteSet qua 2 lệnh kiểm tra về tập con "ngặt" và tập cha "ngặt" lần lượt là: is_proper_subset() và is_proper_superset(). Ví dụ:

>>> from sympy import FiniteSet

>>> s = FiniteSet(1, 2, 3)

>>> t = FiniteSet(3, 2, 1)

>>> s.**is_proper_subset**(t)

…….. ← sinh viên điền kết quả

>>> t.**is_proper_powerset**(s)

…….. ← sinh viên điền kết quả

>>> t = FiniteSet(3, 2, 1, 4)

>>> s.**is_proper_subset**(t)

…….. ← sinh viên điền kết quả

>>> t.**is_proper_ powerset** (s)

…….. ← sinh viên điền kết quả

3.3. Các phép toán trên tập hợp

Các phép toán tập hợp như hợp, giao và tích Cartesian cho phép tổ hợp các tập hợp trong một số phương cách nhất định. Những phép tính tập hợp này đều hữu dụng trên thế giới thực để giải các

bài toán khi chúng ta cần xử lý nhiều tập hợp với nhau. Ngoài ra, những phép tính này chính là tiền đề để tính toán dữ liệu và xác suất các sự kiện ngẫu nhiên.

3.3.1. Union và Intersection

Union của 2 tập hợp là phép nối/hợp của 2 tập chứa các phần tử khác nhau trong 2 tập. Trong lý thuyết tập hợp được kí hiệu bằng ∪. Ví dụ: $\{1,2\} \cup \{2,3\} = \{1,2,3\}$. Trong Sympy, phép hợp được hiện thực bằng phương thức union():

```
>>> from sympy import FiniteSet

>>> s = FiniteSet(2, 4, 6)

>>> t = FiniteSet(3, 5, 7)

>>> kq = s.union(t)

>>> kq

{2, 3, 4, 5, 6, 7}
```

Như vậy, bên trên, chúng ta tìm hợp của 2 tập s và t bằng phương thức union. Kết quả trả về là tập thứ 3 có tên là kq. Ở đây, mỗi phần tử của tập kq là một phần tử của một hoặc hai tập đầu tiên. Ngoài ra, chúng ta có thể xử dụng phép toán + để hợp nối 2 tập hợp.

```
>>> from sympy import FiniteSet

>>> u = FiniteSet(10, 20, 50)

>>> v = FiniteSet(1, 2, 5)

>>> u + v

{1, 2, 5, 10, 20, 50}
```

Phép toán giao (intersection) của hai tập là việc tạo lập 1 tập mới từ các phần tử chung của 2 tập. Ví dụ: giao giữa 2 tập {1, 2, 10} và {1, 3, 10} sẽ được kết quả là tâp mới gồm những phần tử chung, nghĩa là tập mới sẽ là {1, 10}. Về mặt toán học, chúng ta viết như sau

$$\{1,2,10\} \cap \{1,3,10\} = \{1,10\}$$

Trong SymPy, lệnh intersect() để tìm tập giao:

```
>>> from sympy import FiniteSet

>>> s = FiniteSet(1, 2, 10)
```

```
>>> t = FiniteSet(1, 3, 10)

>>> giao = s.intersect(t)

>>> giao

{1, 10}
```

Lưu ý: với các phép toán union() và intersect() đều trả về các tập hợp (các FiniteSet) nên chúng ta có thể sử dụng các phép lồng nhau khi chúng ta có những tính toán liên tiếp. Ví dụ:

```
>>> from sympy import FiniteSet

>>> s = FiniteSet(1, 2, 10)

>>> t = FiniteSet(1, 3, 10)

>>> u = FiniteSet(10, 20, 50)

>>> hop = s.union(t).union(u)

>>> hop

{1, 2, 3, 10, 20, 50}

>>> giao = s.intersect(t).intersect(u)

>>> giao

{10}
```

Nếu tập trả về là tập rỗng thì SymPy sẽ báo đó là tập EmptySet(). Ví dụ:

```
>>> from sympy import FiniteSet

>>> u = FiniteSet(10, 20, 50)

>>> v = FiniteSet(1, 2, 5)

>>> rong = u.intersect(v)

>>> rong

EmptySet()

>>> rong = v.intersect(u)

>>> rong
```

EmptySet()

3.3.2. Tích Descart – Cartesian Product

Tích Descart của tập hữu dụng để tìm tất cả các tổ hợp của các phần tử trong tập hợp. Bằng cách sử dụng tích của tập với chính nó (có thể sử dụng phép toán nhân * hoặc lũy thừa **), chúng ta sẽ tìm được các tổ hợp trong tập hợp, cụ thể là các chỉnh hợp.

Ví dụ:

```
>>> from sympy import FiniteSet

>>> v = FiniteSet(1, 2, 5)

>>> p = v ** 2

>>> p

{1, 2, 5} x {1, 2, 5}

>>> for phantu in p:

        print(phantu)

……

……

……

….

…… ← sinh viên liệt kê ra
```

3.3.3. Áp dụng công thức cho tập nhiều biến

Để xử lý từng phần tử trong một tập FiniteSet, chúng ta có thể sử dụng lệnh lặp **for** như sau:

```
>>> from sympy import FiniteSet

>>> K = FiniteSet(35, 18, 27, 29, 24)

>>> for k in K:

        T = k/10.0

        print (T, k)
```

…… ← sinh viên liệt kê ra

Thứ tự của các số được sắp xếp tăng dần.

3.3.4. Ứng dụng: Tính toán xác suất sự kiện A và sự kiện B cùng xảy ra

Khi chúng ta có 2 sự kiện và chúng ta muốn tính toán xác suất xảy ra 2 sự kiện cùng lúc. Ví dụ: cơ hội cho chúng ta lựa chọn vừa là số nguyên tố vừa là số lẻ (như khi chọn biển số xe).

Để tính toán được, chúng ta cần xác định xác suất của sự kiện **giao** giữa 2 sự kiện:

$$E = A \cap B = \{2, 3, 5, 7\} \cap \{1, 3, 5, 7, 9\} = \{3, 5, 7\}$$

Chúng ta tính toán xác suất của sự kiện cho cả A và B xảy ra cùng lúc bằng phương thức ***intersect***(). Cụ thể:

```
>>> from sympy import FiniteSet

>>> s = FiniteSet(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) # cac so co the trong bien so

>>> a = FiniteSet(2, 3, 5, 7) # cac so nguyen to

>>> b = FiniteSet(1, 3, 5, 7, 9) # cac so le

>>> e = a.intersect(b)

>>> len(e)/len(s)

0.333333333333
```

Như vậy, xác suất để 1 số là số nguyên tố lẻ được chọn từ 10 số từ 0 đến 9 là 0.33, nghĩa là 33%.

Thông tin thêm: Ứng dụng trong y tế cộng đồng: Bài toán trên tuy đơn giản nhưng thường được áp dụng khi tìm kiếm các ca nghi bệnh với tập hợp là những người xuất hiện trong các khu thương mại nơi có bệnh nhân. Để có được tập hợp, người ta sử dụng công nghệ dò sóng điện thoại như GMS/3G/4G/…, để định vị trí người mang theo điện thoai. Và sau đó xử lý để thông báo về y tế của địa phương người cư ngụ.

BÀI TẬP CHƯƠNG 1

Bài 1: Tìm hiểu sự khác nhau và giống nhau giữa 2 loại kiểu dữ liệu tập hợp: set (của ngôn ngữ Python) và FiniteSet (của gói sympy).

Hướng dẫn: tìm kiếm trên Google.

Bài 2: Ở một trường, hiệu trưởng quy định xét học bổng, tất cả các sinh viên ***có điểm trung bình dưới 7 hoặc điểm rèn luyện dưới 7 đều không được xét học bổng***, trường hợp ngược lại sẽ Được xem xét học bổng. Hãy cho biết đoạn script dưới đây đã thể hiện đúng đắn theo quy định xét học bổng chưa, nếu muốn mô tả theo đúng ý của quy định thì cần điều chỉnh lại như thế nào?

```
if not (diemtrungbinh > 7 and diemrenluyen > 7):
    print ("Khong duoc xet hoc bong")
else:
    print ("Duoc xem xet hoc bong")
```

Bài 3: Bác sĩ yêu cầu một kỹ sư viết một phần mềm với bảng dữ liệu về bệnh huyết áp thấp/cao được thu thập như sau:

Phân loại (huyết áp)	Tâm thu (Systolic)	Biểu thức	Tâm trương (díatolic)
Thấp (hypotension)	<90	And/or	<60
Tốt nhất (optimal)	<120	And	<80
Bình thường (normal)	120 – 129	And/or	80 – 84
Bình thường cao	130 – 139	And/or	85 – 89
THA độ 1 (nhẹ)	140 – 159	And/or	90 – 99
THA độ 2 (trung bình)	160 – 179	And/or	100 – 109
THA độ 3 (nặng)	≥180	And/or	≥110
THA tâm thu đơn độc	≥140	And	<90

Bảng định nghĩa và phân loại các cấp độ huyết áp (theo đơn vị mmHg)

Từ bảng dữ liệu về bệnh huyết áp cao (HBP - high blood pressures) như trên, sinh viên hãy viết các dòng lệnh rẽ nhánh **if** trong Python.

Với: cột biểu thức mang giá trị And/or nghĩa là khi hai giá trị ở 2 mức khác nhau thì chọn mức "có bệnh" cao hơn. Mức có bệnh là mức thấp hoặc cao; mức không bệnh là mức Tốt nhất (optimal). Ví dụ: Tâm thu = 85, tâm trương = 55 thì sẽ là mức độ huyết áp thấp. Hoặc khi tâm thu = 125 nhưng tâm trương là 105 thì kết quả là THA độ 2.

Bài 4: Thực hiện bằng kiểu set các tập hợp sau:

a. Cho tập hợp A với x là biến số nguyên như sau:

$$A = \{x | x^2 + 4x + 4 = 0\}$$

Viết đoạn script bằng Python in ra tất cả các giá trị x thỏa điều kiện của tập A.

b. Cho tập hợp A với n là biến số nguyên như sau:

$$A = \{với\ mọi\ n \mid n^2 + 4 < 100\}$$

Viết đoạn script bằng Python in ra tất cả các giá trị n thỏa điều kiện của tập A.

Bài 5: Sử dụng biểu đồ Venn để thể hiện 2 tập hợp

Biểu đồ Venn (Venn diagram) là phương pháp đơn giản để thấy được quan hệ giữa tập hợp. Nó cho chúng ta thấy có bao nhiêu phần tử chung giữa hai tập hợp, bao nhiêu phần tử trong một tập hợp và bao nhiêu phần tử không có trong tập hợp. Ví dụ: Xét tập A gồm các số dương lẻ (posotive odd) nhỏ hơn 20, nghĩa là A = {1, 3, 5, 7 ,9 ,11, 13, 15, 17, 19} và xét tập B là những số nguyên tố (prime) nhỏ hơn 20, nghĩa là B = {2, 3, 5, 7, 11, 13, 17, 19}. Chúng ta có thể vẽ biểu đồ Venn bằng Python với các lệnh như sau:

```
>>> from matplotlib_venn import venn2
>>> import matplotlib.pyplot as plt
>>> from sympy import FiniteSet
>>> def draw_venn(sets):
        venn2(subsets=sets)
        plt.show()


>>> s1 = FiniteSet(1, 3, 5, 7, 9, 11, 13, 15, 17, 19)
>>> s2 = FiniteSet(2, 3, 5, 7, 11, 13, 17, 19)
>>> draw_venn([s1, s2])
```

Sau khi chúng ta import các module và hàm cần thiết (như là venn2(), matplotlib.pyplot và FiniteSet), tất cả việc còn lại là tạo 2 tập và gọi hàm venn2 cho tập hợp ghép:

>>> s3 = [s1, s2]

>>> s3

[{1, 3, 5, 7, 9, 11, 13, 15, 17, 19}, {2, 3, 5, 7, 11, 13, 17, 19}]

Với tập ghép, chúng ta sẽ vẽ được biểu đồ Venn và biểu đồ Venn cho ta kết quả:

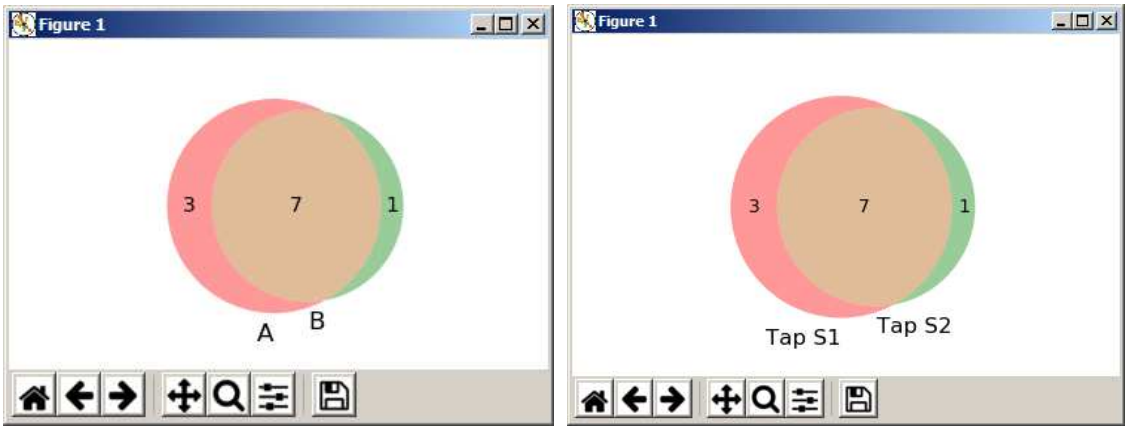
- Tập A có 3 phần tử riêng.
- Tập B có 1 phần tử riêng.
- Tập A và B có 7 phần tử chung.

Để rõ hơn, chúng ta có thể đặt tên các tập hợp bằng cách điều chỉnh hàm vẽ biểu đồ Venn như sau:

```
>>> def draw_venn(sets):
        venn2(subsets=sets, set_labels=('Tap S1', 'Tap S2') )
        plt.show()
```

Và sau đó thực hiện lại lệnh vẽ:

```
>>> draw_venn([s1, s2])
```



Biểu đồ Venn thể hiện quan hệ giữa 2 tập hợp A và B

Điều kiện: Sinh viên cài đặt hoàn tất gói matplotlib_venn

Sinh viên đọc thêm bài dưới đây để làm các bài tập: Bài tập 6, Bài tập 7 và Bài tập 8:

- **Ứng dụng: Nền tảng của công nghệ RAID khôi phục dữ liệu**

Sinh viên đọc và tìm hiểu về phương pháp sử dụng XOR trong khôi phục dữ liệu khi đĩa cứng chứa dữ liệu bị hư trên hệ thống máy chủ nhiều đĩa cứng.

Bài toán thực tiễn

Hư hỏng đĩa cứng dẫn đến mất mát dữ liệu là vấn đề xảy ra thường xuyên đối với người sử dụng máy tính. Để khắc phục tình trạng trên, các hệ thống server hỗ trợ công nghệ có tính năng đặc

biệt là: mỗi khi có một đĩa cứng trên hệ thống bị hư thì hệ thống vẫn tiếp tục chạy và dữ liệu không bị mất để thay thế ổ cứng khác.

Giải pháp đơn giản nhất là công nghệ sao chép dữ liệu trùng nhau. Điều này nghĩa là sẽ có hai đĩa cứng có nội dung giống y chang nhau. Từ đó, mỗi khi có một đĩa cứng bị hư và khi thay thế bằng ổ đĩa mới, dữ liệu sẽ được tự động sao chép từ ổ đĩa còn lại sang ổ đĩa mới. Giải pháp này dễ cài đặt, dễ hiểu và đơn giản nhưng hoàn toàn tốn kém về đĩa do dung lượng lưu trữ bị tăng gấp đôi, cụ thể là hiệu suất lưu trữ chỉ đạt 50% lưu trữ.

Từ đó, người ta cần tìm một giải pháp để lưu trữ dữ liệu, có khả năng lưu trữ hiệu suất cao, sử dụng nhiều đĩa cứng cùng lúc và việc phục hồi dữ liệu cho bất kỳ đĩa cứng hỏng nào. Nghĩa là mô hình lưu trữ đạt được bao gồm 3 yếu tố:

- ***Yếu tố 1***: Hiệu suất lưu trữ cao.
- ***Yếu tố 2***: Khi một đĩa cứng bị hỏng, khả năng phục hồi dữ liệu là 100%.
- ***Yếu tố 3***: Dữ liệu có khả năng đọc/truy xuất được song song cùng lúc.

Phép XOR toán học và thể hiện trên Python

Phép XOR toán học là phép "cộng nhị phân không nhớ" với bảng chân trị như sau:

TT	a	b	a XOR b	Lưu ý
1	0	0	0	
2	1	0	1	
3	0	1	1	
4	1	1	0	Không "nhớ" sang giá trị hàng "chục" như phép cộng

Ở Python, chúng ta sử dụng dấu ^ để mô tả về phép XOR. Ngoài ra, lớp operator có module xor để thực hiện phép XOR. Ví dụ về phép XOR:

>>> **from *operator* import *xor***

>>> a = 2 # số nhị phân của số 2 là 10

>>> b = 3 # số nhị phân của số 3 là 11

>>> axorb = **xor**(a, b) # dự kiến kết quả: 10 XOR 11 = 01

>>> axorb

…….…… ← sinh viên ghi kết quả vào và giải thích.

Ví dụ 2: Sinh viên thực hành các câu lệnh sau và hãy cho biết giá trị của phép toán XOR:

>>> a = 15 # mã nhị phân tương ứng 01111

>>> b = 14 # mã nhị phân tương ứng 01110

>>> c = 16 # mã nhị phân tương ứng 10000

>>> d = a ^ b ^ c

>>> d

…… ← sinh viên ghi kết quả vào và giải thích.

Tư tưởng ứng dụng phép XOR dữ liệu

Từ ví dụ 2, hãy thực hiện ví dụ sau:

Ví dụ 3: Sinh viên hãy cho biết giá trị của phép toán XOR sau:

>>> a = 15 # mã nhị phân tương ứng 01111

>>> e = 17 # mã nhị phân tương ứng 10001

>>> c = 16 # mã nhị phân tương ứng 10000

>>> f = a ^ e ^ c

>>> f

…… ← sinh viên ghi kết quả vào và giải thích.

Từ kết quả trên, chúng ta thấy được điều kì diệu của phép toán XOR, đó là khả năng phục hồi dữ liệu. Trong ví dụ trên, giả định dữ liệu được lưu trữ là a, b, c và d chính là dữ liệu kiểm (parity). Khi lưu đủ 4 khối trên, giả định mất một khối dữ liệu b thì việc khôi phục hoàn toàn là có thể.

Tổng quát: Gọi A là kết quả của phép toán XOR của n số từ a_1 đến a_n

$$A = a_1 \hat{} a_2 \ldots \hat{} a_i \hat{} \ldots \hat{} a_n$$

Khi đó, a_i sẽ được khôi phục theo công thức:

$$a_i = a_1 \hat{} a_2 \ldots \hat{} A \hat{} \ldots \hat{} a_n$$

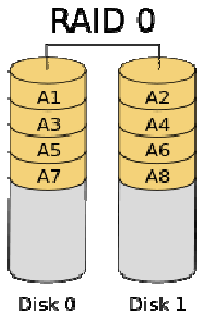
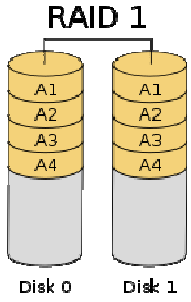
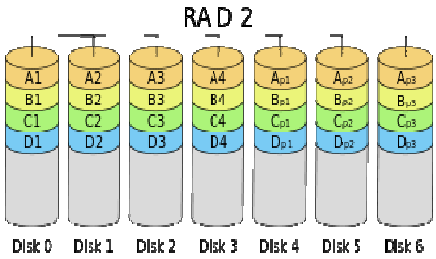
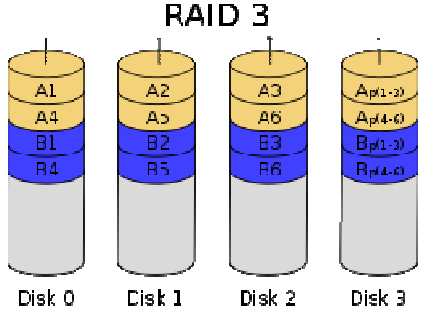
(Sinh viên có thể tự chứng minh)

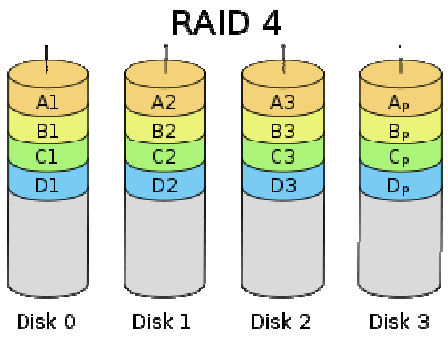
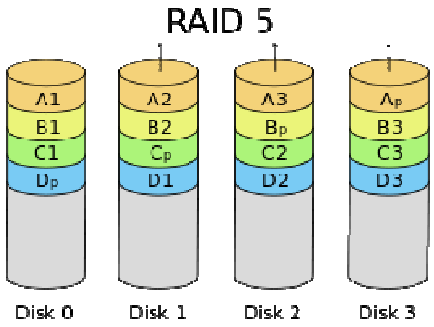
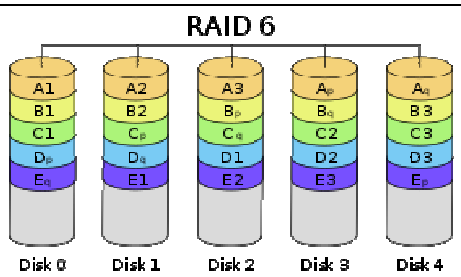
Mở rộng ra, với bài toán được nêu ở mục 1 cùng với các tiêu chuẩn, chúng ta có 2 tiêu chuẩn được đáp ứng:

- Đáp ứng ***yếu tố 1***: vì chỉ cần 1 ổ đĩa để lưu trữ dữ liệu parity (dữ liệu XOR).
- Đáp ứng ***yếu tố 2***: vì khả năng khôi phục dữ liệu với phép xor dữ liệu

Các giải pháp RAID và giải pháp RAID5

Trên thực tế, công nghệ dãy các đĩa độc lập (RAID – Redundant array of independent disks) được phát triển để lưu trữ dữ liệu một cách an toàn và hiệu quả. Với các giải pháp RAID 0, RAID 1, RAID 2, RAID 3, RAID 4, RAID 5 và sau đó nhiều công nghệ RAID sau 5 hiện tại được triển khai ở các hệ thống toàn cầu. Cụ thể, chúng ta có bảng so sánh và tìm hiểu về cách lưu trữ dữ liệu như sau:

TT	RAID	Mô hình	Đặc tính kỹ thuật cơ bản	Ghi chú
1	RAID 0		Số lượng đĩa tối thiểu: 2. Lưu trữ vòng (stripes), không có thông tin kiểm lỗi (parity), nghĩa là không dung lỗi (fault tolerance). Thích hợp: tính toán khoa học nhanh hoặc các hệ thống game. RAID này chủ yếu tăng tốc đọc dữ liệu.	Chỉ tăng tốc, đĩa cứng hư vẫn mất dữ liệu.
2	RAID 1		Số lượng đĩa tối thiểu: 2. Lưu mirror dữ liệu, mỗi dữ liệu lưu 2 nơi. Hiệu suất lưu không cao, tốc độ lưu trữ chậm. Có khả năng phục hồi dữ liệu khi 1 đĩa cứng bị hư.	
3	RAID 2		Số lượng đĩa tối thiểu: 3. Sử dụng hamming code hoặc error corection. Hiện tại ít hệ thống sử dụng do hiệu qua không cao	Sinh viên từ tìm hiểu hamming code hoặc error correction là gì.
4	RAID 3		Số lượng đĩa tối thiểu: 3. Lấy đĩa cuối cùng là parity. Cho các ứng dụng đọc dữ liệu nhỏ, chỉnh sửa video không nén. Nhanh chóng được thay thế bởi RAID 5	Đây là hình thức sơ khai XOR dữ liệu.

5	RAID 4		Số lượng đĩa tối thiểu: 3. Cải tiến RAID 3. Gia tăng khối lưu	
6	RAID 5		Số lượng đĩa tối thiểu: 3. Cải tiến RAID 4, các khối parity lưu xoay vòng	Đáp ứng 3 tiêu chí đề ra trong bài toán ban đầu.
7	RAID 6		Số lượng đĩa tối thiểu: 4. Tăng cường parity	Sử dụng 2 parity nhằm khôi phục bất kì 2 đĩa nào bị hư. Ví dụ: Đĩa parity1 = XOR(đĩa 1, 2, 3) Đĩa parity2 = XOR(đĩa 1, 2, 4)
8	Các dạng RAID lồng nhau		Ví dụ: RAID 0+1 (gọi là RAID 01); RAID 03; RAID 10; RAID 50; RAID 60; RAID 100.	Kết hợp các giải pháp RAID. Tùy ứng dụng.

Đọc thêm tại: https://en.m.wikipedia.org/wiki/Standard_RAID_levels

Lời kết về ứng dụng XOR trong RAID

Từ đây, chúng ta có thể nói mục tiêu thứ 3 sẽ hoàn thành khi chúng ta điều chỉnh/quản lý được nơi lưu trữ dữ liệu parity, cụ thể là:

- Đáp ứng ***yếu tố 3***: vì nếu như dữ liệu được "băm" thành các khối và parity được lưu xoay vòng các ổ đĩa thì dữ liệu được trãi đều ở các ổ đĩa (xem hình mô hình RAID5)

Ngoài ra, với RAID 6, chúng ta có mức dung lỗi (fault-tolerance) cao hơn khi cùng lúc 2 đĩa có thể bị hỏng mà dữ liệu vẫn khôi phục được.

Trong bài này, giải pháp RAID, đặc biệt là RAID 5 được giới thiệu cùng với phép toán XOR. Phép toán XOR là một phép toán đơn giản trong luận lý nhưng ứng dụng được trong việc xây dựng hệ thống lưu trữ cao cấp hiện nay.

---------- Phần bài tập:

Bài 6: Giả sử ta có tập dữ liệu a1, a2, và a3 và A là dữ liệu dùng để phục hồi trong trường hợp một trong các dữ liệu a1, a2 hay a3 bị lỗi.

$$A = a1\hat{}a2\hat{}a3$$

a) Giả sử a1 = 10, a2 = 16, A = 17. Hãy xây dựng công thức đồng thời phục hồi lại dữ liệu a3. Viết đoạn script minh họa.
b) Giả sử a1 = 10, a3 = 11, A = 17. Hãy xây dựng công thức đồng thời phục hồi lại dữ liệu a2. Viết đoạn script minh họa.
c) Giả sử a2 = 16, a3 = 11, A = 17. Hãy xây dựng công thức đồng thời phục hồi lại dữ liệu a1. Viết đoạn script minh họa.

Bài 7*: Viết chương trình mô phỏng phương pháp XOR để bảo toàn dữ liệu: Chia cắt 1 tập tin text thành các dữ liệu nhỏ để chuyển đi và có thể dựng lại thành file text nhỏ hơn. Khi cần sẽ tập hợp lại (minh họa các RAID 3/4/5).

Bài 8*: Tìm hiểu và viết chương trình mô phỏng cách thực hiện của RAID 6.

*(Các bài tập * có thể nộp vào cuối khóa để được nhiều điểm cộng vào điểm thi thực hành)*

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm biên soạn và Giảng viên có đóng góp ý kiến: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Lê Ngọc Thành – Phạm Trọng Nghĩa - Nguyễn Công Nhựt – Trần Ngọc Việt - Hoàng Thị Kiều Anh – Đỗ Đình Thủ - Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 2: ÁNH XẠ VÀ QUY NẠP TOÁN HỌC

Mục tiêu:

- *Hiểu được các loại ánh xạ. Có khả năng thể hiện trong ngôn ngữ Python.*
- *Hiểu được quy nạp toán học và phương pháp viết các hàm đệ quy trong Python.*

Nội dung chính:

1. Ôn luyện kiến thức cơ bản Python

Nhắc lại một số vấn đề về xây dựng hàm (module) trong Python:

1.1. Viết hàm trong Python

Trong Python, một hàm số đơn giản được khai báo thông qua từ khóa def với tên hàm và kết quả trả về bằng từ khóa return. Ví dụ: Hàm tính lũy thừa số nguyên x^n như sau:

```
>>> def luythua(x, n):
        ketqua= 1
        for i in range(n):
            ketqua = ketqua *x
        return ketqua
>>> luythua(2,1)
…… ← sinh viên điền kết quả
>>> luythua(2,0)
…… ← sinh viên điền kết quả
```

Lưu ý: Một số đặc điểm nổi bật trong Python về hàm:

- *Hàm "main" của một tập tin .py:*

Là đoạn chương trình nằm trong một khối lệnh, thường ở vị trí cuối của tập tin .py. Khi khối lệnh này tồn tại, chúng ta có thể "thực thi" tập tin Python đó từ dòng lệnh của hệ điều hành, như:

C:\> python abc.py

Khối lệnh bắt đầu từ việc so sánh 2 từ khóa riêng __name__ và __main__ của Python như sau:

if __name__ == "__main__":

#... thực hiện gì đó….

Lưu ý:

- *Hàm (module) lồng trong hàm (module):* Một module có thể có các module con trong nó.
- *Hàm của lớp đối tượng (object)*: là các module được định nghĩa trong một đối tượng class. Các module riêng tư phía trước phải có 2 __.
- *Sử dụng hàm trong file Python cùng thư mục:*

Câu lệnh import là để triệu gọi một thư viện. Ví dụ file A.py muốn sử dụng module x trong file B.py. Do đó, với một file Python có sẵn các module, chúng ta có thể sử dụng các module của nó khi chúng ta thêm được vị trí thư mục của thư viện vào lúc thực thi (runtime). Hai bước lệnh dưới đây hỗ trợ việc khai báo thư mục cho tập tin thư viện (tập tin B.py) cùng thư mục với tập tin thực thi (tập tin A.py):

Bước 1: Đưa tập tin thư viện vào cùng thư mục với tập tin sử dụng các hàm của nó. Nghĩa là sao chép file B vào cùng thư mục file A.

Bước 2: Khai báo thư mục của tập tin thư viện để import với 3 lệnh (được viết trong file A.py):

from os **import** sys, path

sys.path.append(path.dirname(path.dirname(path.abspath(__file__))))

from B import x

lưu ý: khi sử dụng sẽ là:

B.x(…)

1.2. Kiểu dữ liệu tập hợp (set) trong Python

Python hỗ trợ kiểu tập hợp là kiểu set, kiểu set được đặt trong 2 dấu **{}**.

Ví dụ 1: Tạo tập hợp 3 phần tử:

>>> {2, 4, 6}

Ví dụ 2: Lệnh tạo ra tập hợp các số lẻ nhỏ hơn 100:

>>> { n for n in range(100) if n % 2 == 1 }

Ví dụ 3: Tạo tập rỗng:

>>> tap_rong = **set**()

>>> print (tap_rong)

…….. ← sinh viên điền kết quả

Với đối tượng tập hợp (set) trong Python, các lệnh xử lý sẽ được hỗ trợ tương ứng với các lệnh toán học như sau:

Set operators (Toán tử tập hợp)	Mô tả toán học	Mô tả trong Python
Thuộc về	∈	in
Không thuộc về	∉	not in
Tập con	⊆	<=
Tập con (ngặt)	⊂	<
Hợp (union)	∪	|
Hội (intersection)	∩	&
Hiệu (difference)	–	-

Sinh viên thực tập các lệnh sau:

>>> setA = { n for n in range(100) if n % 2 == 1 }

>>> setB = { n for n in range(100) if n % 2 == 1 }

>>> setA | setB

……

>>> setA & setB

……

>>> setA - setB

……

>>> setA < setB

……

>>> setA <= setB

……

>>> 10 in setA

……

>>> 11 in setA

……

>>> setA.add(0)

……

>>> setA > setB

……

1.3. Hàm lambda trong Python

Hàm lambda là một dạng lập trình hàm chỉ tồn tại trên ngôn ngữ bậc cao. Dưới đây, bài thực hành chỉ nêu một số vấn đề cơ bản của dạng hàm lambda. Những cài đặt phức tạp hơn sẽ có trong các cấp độ lập trình cao cấp hơn.

Có từ phiên bản 2.2, hàm **lambda** trong Python gọi là hàm "vô danh" (anonymous function) *trong lúc chạy*. Theo đó, có 3 phương thức làm cho hàm này trở nên đặc biệt: **filter()**, **map()** và **reduce()**. Một đặc điểm là hàm **lambda** không có lệnh ***return*** trả về như các module của Python nhưng nó luôn thực hiện các lệnh trong đó để trả về giá trị.

Ví dụ 1: Sử dụng hàm lambda để tính x^2:

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

g = lambda x: x**3

Ví dụ 2: Tính thuế theo địa phương

Bài toán: Giả sử thuế một mặt hàng nhập khẩu vào TP.HCM là 0.012 (nghĩa là 1.2%); ở Hà Nội là 0.010 (nghĩa là 1%). Hãy viết hàm tính (lưu ý: hệ thống tính thuế sẽ áp dụng cho nhiều địa

phương). Như vậy, ở địa phương nào cần thay đổi thuế thì chỉ cần thay đổi tại phương đó mà không ảnh hưởng đến ***công thức chung*** cho cả nước.

>>> **def *thue***(phan_tram): return lambda x: x * phan_tram

>>> hcm = ***thue***(0.012) # ← khai báo mức thuế ở Hà Nội.

>>> hn = ***thue***(0.01) # ← khai báo mức thuế ở TP.HCM.

>>> hcm(1000000) # ← minh họa gọi hàm tính thuế cho 1 triệu đồng tại TP.HCM.

12000.0

Ví dụ 3: Liệt kê các số chia hết cho 3 từ 2 đến 49.

Danh sách số từ 2 → 49 là: day_so = range(2, 50)

Lọc bằng hàm lambda (sử dụng filter)

print **filter**(lambda x: **x % 3 == 0**, *day_so*)

Ví dụ 4*:* Về hàm **map**()

Cho 1 dãy số, muốn cộng cho dãy số đó 1 giá trị

>>> print map(lambda x: x * x , day_so)

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Ví dụ 5*:* Giới thiệu hàm reduce() cho hàm lambda.

Khi muốn tính tổng của một dãy số.

>>> day_so = range(1, 11)

>>> print reduce(lambda x, y: x+y, day_so)

55

Ví dụ 6*:* Tìm số lớn nhất trong một dãy số

Cho dãy số ds = [20, 25, 50, 103, 13, 19]

Viết 1 hàm lambda để chọn số:

>>>f = lambda a,b: a if (a>b) else b

Sau đó viết hàm **reduce** áp hàm f lên dãy số để được kết quả.

```
>>> print reduce(f, ds)
```

……

[Hàm lambda đối với ma trận]

Ví dụ 7: in ra ma trận đường chéo I_n với n là số nguyên (n>1):

```
>>>imatrix = lambda n:[[1 if j==i else 0 for j in range(n)] for i in range(n)]
>>> imatrix(3)
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

Hoặc đoạn mã với thư viện numpy cho ra ma trận đơn vị với số thực:

```
>>> import numpy
>>> list_eye = lambda n: numpy.eye(n).tolist()
>>> list_eye(4)
[[1.0, 0.0, 0.0, 0.0], [0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0, 1.0]]
```

Ví dụ 8*:* In ra ma trận hệ số: với n==3 thì là ma trận 3x3 có đường chéo là căn 2.

```
>>> g = lambda n:[[1 if (j+1==i or j==i+1) else math.sqrt(2) for j in range(n)] for i in range(n)]
>>> g(3)
[[1.4142135623730951, 1, 1.4142135623730951], [1, 1.4142135623730951, 1],
[1.4142135623730951, 1, 1.4142135623730951]]
```

Ví dụ 9: Như ví dụ 8 nhưng ô ở giữa bằng 0

```
>>> g = lambda n:[[1 if (j+1==i or j==i+1) else 0 if(i==n/2) else math.sqrt(2) for j in range(n)]
for i in range(n)]
>>> g(3)
[[1.4142135623730951, 1, 1.4142135623730951], [1, 0, 1], [1.4142135623730951, 1,
1.4142135623730951]]
```

2. Ánh xạ và hàm hợp

2.1. Phân loại ánh xạ và một số tính chất

Các ví dụ sau về đơn ánh (tiếng Anh gọi là injection, là hàm có nhiều nhất một nghiệm), toàn ánh (surjection, là hàm luôn có nghiệm, có thể có nhiều hơn 1 nghiệm) và song ánh (bijection, là hàm một nghiệm duy nhất).

- Đơn ánh: $f: \mathbb{R} \longrightarrow \mathbb{R}$ $với\ f(x) = 2^x + 15$ hoặc $f: \mathbb{Z} \longrightarrow \mathbb{Z}$ $với\ f(x) = 2x - 1$
- Toàn ánh: $f: \mathbb{R} \longrightarrow \mathbb{R}$ $với\ f(x) = x^3 + 3x^2$ hoặc

$$f: \mathbb{Z} \longrightarrow \{0,1\}\ với\ f(x) = \begin{cases} 0\ nếu\ x\ chẵn \\ 1\ nếu\ x\ lẻ \end{cases}$$

- Song ánh: $f: \mathbb{R} \longrightarrow \mathbb{R}$ $với\ f(x) = x + 10$ hoặc $f: \mathbb{Z} \longrightarrow \mathbb{Z}$ $với\ f(x) = x + 100$
- Không phải toàn ánh cũng không phải đơn ánh: $f: \mathbb{R} \longrightarrow \mathbb{R}$ $với\ f(x) = x^2 + 1$

Một số tính chất đặc biệt của hàm hợp các ánh xạ:

- Nếu hàm f và hàm g toàn ánh thì $f \circ g$ sẽ là toàn ánh.
- Nếu hàm f và hàm g đơn ánh thì $f \circ g$ sẽ là đơn ánh.
- Nếu hàm f và hàm g song ánh thì $f \circ g$ sẽ là song ánh.
- Nếu hàm f song ánh thì ánh xạ ngược của f là f^{-1} sẽ là một song ánh

Mệnh đề đảo của 3 khẳng định trên là không đúng. Tuy nhiên, chúng ta có các mệnh đề sau:

- Nếu $f \circ g$ sẽ là toàn ánh thì f là toàn ánh.
- Nếu $f \circ g$ sẽ là đơn ánh thì g là toàn ánh.
- Nếu $f \circ g$ sẽ là song ánh thì f là toàn ánh và g là đơn ánh.

Lưu ý: $(f \circ g)(x) = f\big(g(x)\big)$

2.2. Hàm hợp trong Python

Python cho phép người sử dụng định nghĩa hàm hợp thông qua việc sử dụng hàm **lambda**. Sinh viên thực tập các lệnh dưới đây:

```
>>> def alpha(a):   # hàm y = x+10
        return a+10
>>> def beta(b):   # hàm y = 3x
        return b*3
>>> def ham_hop(g, f):                # hàm compose y = 3*(x+10)
        return (lambda x: g(f(x)))
>>> h = ham_hop(alpha, beta)
```

```
>>> h(1)
……                                          ← sinh viên ghi kết quả vào và giải thích.
>>> g = ham_hop(beta, alpha)    # Hàm compose y = (3*x) + 10
>>> g(1)
……                                          ← sinh viên ghi kết quả vào và giải thích.
```

3. Quy nạp toán học và xây dựng hàm đệ quy trong Python

Quy nạp (inductive) toán học thường là một kỹ thuật định nghĩa hoặc chứng minh theo nguyên tắc quy nạp (induction principle). Do đó, thông thường quy nạp toán học sẽ thực hiện 2 bước: bước 1 là chứng minh sự đúng đắn của mệnh đề P(1) được phát biểu trong trường hợp đơn giản và sau đó bước 2 là mở rộng điều đó vẫn đúng khi P(k+1), với k là số nguyên. Trong một số trường hợp, kết quả chứng minh quy nạp giúp chúng ta có cái nhìn tổng thể nhằm ***giảm được xử lý tính toán cho máy tính***. Ví dụ: biểu thức dưới đây: bên trái cần tính toán toán n phép cộng, trong khi đó bên phải chỉ cần 3 phép toán gồm (1 cộng, 1 nhân và 1 chia) cho bất kì giá trị n:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Sinh viên thực tập các câu lệnh sau:

- *Cho biết các hàm tính thời gian và phương pháp sử dụng như sau:*

```
>>> from datetime import datetime    # Tham chiếu đến thư viện lấy ngày giờ hiện hành
>>> from datetime import timedelta  # tham chiếu đến thư viện tính toán khoảng thời gian
>>> def millis(start_time):                  # start_time là tham số lưu trữ thời gian trước đó
        dt = datetime.now() - start_time  # lấy hiệu 2 thời gian: thời gian hiện hành và trước đó
        ms = (dt.days * 24 * 60 * 60 + dt.seconds) * 1000 + dt.microseconds / 1000.0
        return ms                         # trả về là giá trị mili giây
>>> def milisec_passed(start_time):
        thoigian_troi_qua = millis(start_time)    # thời gian trôi qua
```

```
        s =  "Thoi gian da troi qua:" + str(int(thoigian_troi_qua)) +  " mili giay."
        return s
```

- *Xây dựng các hàm tính toán theo vế trái và vế phải:*

```
>>> def tong(n):
        bat_dau = datetime.now()  # Ghi nhận thời điểm bắt đầu
        ketqua = 0
        for i in range(n+1):
            ketqua = ketqua + i
        thoi_gian = str(milisec_passed(bat_dau))  # Tính toán thời gian trôi qua
        print(thoi_gian)
        return ketqua
>>> tong(100000000)     # tính toán tổng từ 1 đến 1 trăm triệu
……  ← sinh viên ghi kết quả vào.
……
>>> def tong1(n):
        bat_dau = datetime.now()  # Ghi nhận thời điểm bắt đầu
        ket_qua = n*(n+1)/2
        thoi_gian = str(milisec_passed(bat_dau))  # Tính toán thời gian trôi qua
        print(thoi_gian)
        return ket_qua
>>> tong1(100000000)    # tính toán tổng từ 1 đến 1 trăm triệu
……  ← sinh viên ghi kết quả vào.
……
```

Trong khi đó, đệ quy (recursive) là một kỹ thuật trong lập trình để triệu gọi lại hàm đang thực thi ở một điều kiện nào đó. Như vậy, một định nghĩa quy nạp chính là mô tả việc đệ quy. Tuy nhiên, thực hiện đệ quy chưa chắc là một mô tả về quy nạp.

Ví dụ: Định nghĩa về: lũy thừa: $x^n = x.x^{n-1}$ và $x^0 = 1$; và định nghĩa giai thừa: $k! = (k-1)!\,k$ và $1! = 1$.

Đệ quy thường được sử dụng trong các định nghĩa hàm/thủ tục/module trong các ngôn ngữ lập trình bằng việc nó thực hiện/gọi lại công việc của nó. Cấu trúc đệ quy đôi khi có thể mô tả bằng cấu trúc lặp và rẽ nhánh. Tuy nhiên, một số vấn đề sẽ phù hợp hoặc không phù hợp với dạng đệ quy hoặc lặp. Về nguyên tắc, để xây dựng một tiến trình đệ quy, 3 quy luật được khuyến cáo sử dụng là:

- Phải có trường hợp cơ sở. Nghĩa là trường hợp hàm không gọi chính nó nữa. Đôi khi người ta gọi là điều kiện dừng. Ví dụ: khi tính giai thừa thì trường hợp dừng là khi tính 0!
- Phải có dòng lệnh để hàm/module gọi lại chính nó.
- Phải đi theo định hướng xử lý từ trường hợp đệ quy sang trường hợp cơ sở. Nghĩa là việc triệu gọi hàm chắc chắn sẽ có hướng dừng thuật toán.

Ba quy luật trên để đảm bảo mọi hàm/module đệ quy đều thực hiện ít nhất 1 lần và định hướng được tính dừng của thuật toán. Nếu chúng ta không kiểm soát được các vấn đề trên, việc đệ quy thường gây vượt bộ nhớ hoặc vòng lặp không thoát (vĩnh cửu).

Ví dụ: Hãy viết chương trình đệ quy cho dãy số Fibonacci F. Dãy Fibonacci được định nghĩa như sau F(n): F(0) = 0, F(1) = 1 và F(n) = F(n-1) + F(n-2), n>= 2, nghĩa là chuỗi Fibonacci sẽ là: 0,1, 1, 2, 3, 5, 8, 13, 21,… Lưu ý rằng: các chỉ số có thể thay đổi do có một số định nghĩa quy định số đầu tiên của chuỗi là 0 hoặc 1, cụ thể là F(0) = 0 hay F(0) = 1. Do đó, một số chương trình tính toán F(3) = 3, F(4) = 5, F(5)=8 trong khi đó một số chương trình khác lại cho kết quả F(3) = 2, F(4) = 3, F(5) = 5. Sinh viên thực hành viết lệnh đệ quy như sau:

```
>>> def fibo(n):
        if (n == 0) or (n == 1):
            return 1
        else:
            return fibo(n-1) + fibo(n-2)
```

Sau đó, sinh viên thử nghiệm:

```
>>> fibo(2)
……
>>> fibo(3)
……
>>> fibo(4)
……
>>> fibo(5)
……
```

- **Kỹ thuật Memoization áp dụng trong đệ quy với Python**

Với cách cài đặt đệ quy như định nghĩa bên trên, việc tính toán sẽ gần như bị "nhân đôi". Ví dụ: giá trị F(n-2) bị tính toán 2 lần từ đệ quy, cụ thể là khi tính F(n) và khi tính F(n-1). Và do đó, (n-1) giá trị từ F(0),…F(n-2) sẽ "bị" tính toán 2 lần. Việc tính toán lặp dẫn đến mất nhiều thời gian. Để khắc phục nhược điểm trên, một kỹ thuật hỗ trợ người lập trình là cơ chế **memoization**, nghĩa là các giá trị đã được tính toán sẽ được lưu trữ ở một dạng dữ liệu gọi là từ điển (**dict**) và khi đã có chỉ cần triệu gọi ra. Dưới đây là cài đặt tính toán đệ quy với kỹ thuật memoization bằng Python:

```
>>> def fibo_mem(n, so_da_tinh = {0:0, 1: 1}):
        if  n  not in so_da_tinh:
                so_da_tinh[n] = fibo_mem(n-1, so_da_tinh) + fibo_mem(n-2, so_da_tinh)
        return so_da_tinh[n]
```

Sinh viên điền các kết quả tính toán như sau:

```
>>> fibo_mem(3)
……
>>> fibo_mem(4)
……
>>> fibo_mem(5)
……
>>> fibo_mem(200)
……
>>> fibo_mem(500)
…….
…….
…….
```

BÀI TẬP CHƯƠNG 2

Câu 1: Hãy viết các chương trình (module) Python tính toán theo 2 cách vế trái và vế phải của các biểu thức sau:

$$1^2 + 2^2 + \cdots + (n-1)^2 + n^2 = \left(\frac{1}{6}\right) n(n+1)(2n+1)$$

Và cho biết tính toán thời gian trôi qua để máy tính thực hiện 2 hàm trên (theo mili giây) cho tổng từ 1 đến $(10000000)^2$

Câu 2: Dãy số Fibonacci được định nghĩa như sau F(n):

F(0) = 0, F(1) = 1

F(n) = F(n-1) + F(n-2), n>= 2

Hãy viết chương trình tính toán F với n bất kì (gợi ý: n = 10, 100, 1000, 10000) và tính toán thời gian xử lý tương ứng. Yêu cầu:

a. Chương trình (module) viết theo kỹ thuật lặp (không đệ quy).
b. Chương trình (module) viết bằng cách tính toán trực tiếp giá trị số Fibonacci theo công thức ma trận (sẽ học trong thực hành đại số tuyến tính).
c. So sánh thời gian thực thi của các chương trình vừa viết và trong bài học.

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Phạm Trọng Nghĩa – Nguyễn Công Nhựt – Trần Ngọc Việt - Hoàng Thị Kiều Anh – Lê Ngọc Thành – Đỗ Đình Thủ – Nguyễn Hữu Trí Nhật – Lê Công Hiếu – Nguyễn Thị Thanh Bình – Nguyễn Thái Hải – Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 3: PHÉP ĐẾM: VỀ CÁC NGUYÊN LÝ

Mục tiêu:

- *Về các nguyên lý cộng, nhân, bù trừ và nguyên lý chuồng bồ câu.*

Nội dung chính:

1. Các nguyên lý: cộng, nhân, bù trừ

*Lưu ý: Trong Python 3.x, dữ liệu của lệnh **print** bắt buộc trong (), như: **>>> print(<dữ liệu>)**.*

1.1. Các nguyên lý

Trong Python cũng như các ngôn ngữ lập trình cao cấp khác, các bài toán tổ hợp thường được cấu trúc bằng vòng lặp for. Với *nguyên lý cộng*, chúng ta có thể sử dụng/thực thi các vòng lặp theo tuần tự, nghĩa là từng **vòng lặp độc lập với nhau**. Ví dụ:

Cầu thủ bóng đá Việt Nam gồm: Văn Lâm, Tiến Dũng, Anh Đức, Công Phượng,…

Cầu thủ bóng đá thế giới gồm: Messi, Ronaldo, Thonglao, Mbappé, …

+ Liệt kê cầu thủ bóng đá Việt Nam:

```
>>> bongda_VN = [‘Văn Lâm’, ‘Tiến Dũng’, ‘Anh Đức’, ‘Công Phượng’]
>>> for cau_thu in bongda_VN:
        print ("Ten cau thu VietNam: ", cau_thu)
```

…… ← Sinh viên cho biết kết quả.

+ Liệt kê số cầu thủ bóng đá thế giới:

```
>>> bongda_TG = [‘Messi’, ‘Ronaldo’, ‘Thonglao’, ‘Mbappé’]
>>> for cau_thu in bongda_TG:
        print ("Ten cau thu The gioi: ", cau_thu)
```

…… ← Sinh viên cho biết kết quả.

Từ đó, chúng ta có thể sử dụng vòng lặp lồng ghép vào nhau (nested loop) để tính toán cho *nguyên lý nhân*. Ví dụ: Tìm sự tranh chấp Quả bóng vàng giữa cầu thủ Việt Nam và Thế giới là sự chọn 1 từ mỗi tập, số trường hợp được liệt kê là tích của số lượng phần tử của 2 tập:

```
>>> for bong_VN in bongda_VN:
        for bong_TG in bongda_TG:
```

```
        print ("Khả năng tranh chấp quả bóng vàng giữa: ", bong_VN, " với ", bong_TG)
```

…… ← Sinh viên cho biết kết quả.

Dưới đây là 4 ví dụ cụ thể hơn để in ra tập số bằng việc sử dụng vòng lặp lồng nhau trong miền số nguyên.

+ Ví dụ 1: *In ra các phương án chọn 3 số nguyên nhỏ hơn N không âm* ***không có thứ tự*** *và* ***có lặp****:*

```
>>> N = 4    # ← giả định các số nguyên không vượt quá N là 4
>>> for i1 in range(0, N):
        for i2 in range(0, N):
            for i3 in range(0, N):
                print (i1, i2, i3)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

+ Ví dụ 2: *In ra các phương án chọn 3 số nguyên nhỏ hơn N không âm* ***có thứ tự*** *và* ***không lặp****:*

```
>>> for i1 in range(0, N):
        for i2 in range(0, N):
            for i3 in range(0, N):
                if i1 != i2 and i2 != i3 and i1 != i3:
                    print (i1, i2, i3)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

+ Ví dụ 3: *In ra các phương án chọn 3 số nguyên nhỏ hơn N không âm có* ***tăng dần*** *và* ***không lặp****:*

```
>>> for i1 in range(0, N):
        for i2 in range(i1+1, N):
            for i3 in range(i2+1, N):
                print (i1, i2, i3)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

+ Ví dụ 4: *In ra các phương án chọn 3 số nguyên nhỏ hơn N không âm có **không giảm** và **có lặp**:*

```
>>> for i1 in range(0, N):
        for i2 in range(i1, N):
            for i3 in range(i2, N):
                print (i1, i2, i3)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

Từ các kết quả trên, chúng ta có thể sử dụng giải pháp lồng ghép các vòng lặp để tìm các tổ hợp phù hợp. Hơn nữa, việc thực hiện trên các số nguyên chính là đề cập đến chỉ số của một danh sách (list) cụ thể. Ví dụ: chúng ta có thể thay thế lệnh **print** (i1, i2, i3) bên trên bằng lệnh cụ thể hơn với tập dữ liệu cần xử lý, như:

```
print bong_VN [i1], bong_VN [i2], bong_VN [i3]
```

Hoặc chúng ta có thể lưu lại các "phương án" lựa chọn trong danh sách, ví dụ:

```
>>> N = 3
>>> ketqua = []
>>> for i1 in range(0, N):
        for i2 in range(i1, N):
            for i3 in range(i2, N):
                ketqua = ketqua + [(i1, i2, i3)]
>>> ketqua
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

```
>>> ketqua[3]
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

1.2. Thể hiện biểu thức luận lý bằng Python

Sự kết nối giữa luận lý, các chứng minh và lập trình là lĩnh vực rộng vô vàn trong toán rời rạc. Kiểu dữ liệu Bool sử dụng khi có một biểu thức cần biết đúng hoặc sai. Sinh viên hãy giải thích hàm **"ngụ ý" (implies)** bên dưới và tìm hiểu giá trị các lệnh dưới đây:

```
>>> def implies(a, b):
        if a:
            return b
        else:
            return True
>>> implies(10, 11)
>>> implies(1, 11)
>>> implies(0, 11)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

Phân tích, với hàm **implies** trên, chúng ta dễ dàng "thể hiện" các biểu thức logic bằng ngôn ngữ Python như sau:

$$x \geq 0 \; and \; y \geq 0 \Longrightarrow x * y \geq 0$$

Khi đó, chúng ta có thể đưa biểu thức trên vào hàm implies để biểu diễn:

```
>>> x, y = 2, 3
>>> implies(x>=0 and y >=0, x*y >=0)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

```
>>> x, y = -2, 3
>>> implies(x>=0 and y >=0, x*y >=0)
```

[Sinh viên hãy tìm hiểu và điền kết quả] ……

Giải thích: Biểu thức 2 số x và y dương thì tích của chúng luôn dương là điều luôn đúng. Trong lúc thực thi chương trình, mặc dù giá trị x và y khác nhưng biểu thức trên là điều luôn đúng. Tuy nhiên, lưu ý rằng Python bắt buộc phải hiểu các biến. Do đó, các biến x và y cần được khởi tạo trước.

2. Lý thuyết tập hợp với kiểu dữ liệu list trong Python

Bên cạnh cấu trúc dữ liệu ***tập hợp*** **set** (được giới thiệu trong chương 1). Dưới đây, chúng ta sẽ xem xét một số đoạn mã sử dụng cấu trúc ***danh sách*** **list** để mô tả và xử lý dữ liệu tập hợp trong môi trường Python. Khác với **set**, một giá trị đã tồn tại trong **list** hoàn toàn có thể thêm mới vào.

Với dữ liệu **list**, Python chỉ hỗ trợ toán tử **in** để kiểm tra phần tử trong ***danh sách*** **list**. Ví dụ:

```
>>> N = 3
>>> ketqua = []
>>> for i1 in range(0, N):
        for i2 in range(i1, N):
            for i3 in range(i2, N):
                ketqua = ketqua + [(i1, i2, i3)]


>>> (0, 2, 2) in ketqua
…… # sinh viên điền và giải thích kết quả
>>> (0, 2, 1) in ketqua
…… # sinh viên điền và giải thích kết quả
```

- Dưới đây là một số đoạn code mô phỏng **xử lý tập hợp với kiểu list** trong Python:

+ Phép giao (**intersection**) giữa 2 tập A và B:

```
>>> A = [0, 1, 2, 3, 4, 5, 6, 7]
>>> B = [2, 4, 6, 8, 10]
>>> def intersection(A, B):
        ketqua = []
        for x in A:
            if x in B:
                ketqua = ketqua + [x]
        return ketqua
```

```
>>> intersection(A, B)
```

…… # sinh viên điền và giải thích kết quả

+ Phép tìm tập hiệu (**difference**) giữa 2 tập A và B:

```
>>> def difference(A, B):
        ketqua = []
        for x in A:
            if x not in B:
                ketqua = ketqua + [x]
        return ketqua


>>> A = [0, 1, 2, 3, 4, 5, 6, 7]
>>> B = [2, 4, 6, 8, 10]
>>> difference(A, B)
```

…… # sinh viên điền và giải thích kết quả

+ Phép kiểm tra tập con (**subset**) giữa 2 tập A và B:

```
>>> def isSubSet(A, B):
        ketqua = True  # DUNG
        for x in A:
            if x not in B:
                ketqua = False # SAI
        return ketqua
>>> A = [0, 1, 2, 3, 4, 5, 6, 7]
>>> B = [2, 4, 6, 8, 10]
>>> isSubSet(A, B)
```

…… # sinh viên điền và giải thích kết quả

```
>>> B = [1, 3, 5, 7]
>>> isSubSet(A, B)
…… # sinh viên điền và giải thích kết quả
>>> isSubSet(B, A)
…… # sinh viên điền và giải thích kết quả
```

Từ đó, chúng ta có thể xây dựng một module để kiểm tra sự bằng nhau giữa hai tập hợp:

```
>>> def equalSets(A, B):
        return isSubSet(A, B) and isSubSet(B, A)
>>> equalSets([1, 3, 5, 7], [7, 3, 1, 5])
…… # sinh viên điền và giải thích kết quả
```

3. Bài toán ứng dụng 1: Xây dựng danh sách tour địa điểm du lịch tại TP.HCM

Bạn Thành là sinh viên ngành Khoa học Dữ liệu. Bạn đang thực tập tại công ty du lịch. Công ty muốn phát triển nhiều tour du lịch tại Thành phố Hồ Chí Minh, các tour du lịch phải phong phú về địa điểm để các du khách lựa chọn. Bên cạnh đó, công ty có thể để chọn lựa giúp du khách thay thế những địa điểm đang trùng tu/bảo dưỡng bằng các địa điểm tương tự. Bạn Thành nghiên cứu xử lý công việc như sau: *"Một đoàn du khách cần các phương án du lịch ngắn ngày tại TP.HCM. Họ đề nghị phía công ty du lịch tư vấn các gói tham quan, nghĩa là lựa chọn tập các điểm đến. Được biết, mỗi địa điểm đến tham quan sẽ tốn một lượng thời gian nhất định. Cho trước danh sách địa điểm tham quan du lịch tại TP.HCM và thời gian tham quan dự kiến tại mỗi địa điểm cùng với số ngày đoàn khách đến thăm. Hãy lập tập hợp những phương án để gửi tư vấn cho đoàn du khách lựa chọn"*.

Ví dụ: Dữ liệu minh họa được cho trong bảng sau, giả định đoàn du khách chỉ ghé thăm TP.HCM không quá 1.5 ngày.

Tiêu chí	**Danh sách địa điểm**								
Địa điểm	Cần Giờ	Củ Chi	Chợ Bến Thành	Sở thú	BV Thẩm mỹ răng	Chí Hòa	Dinh Độc Lập	Bảo Tàng	Trường Văn Lang
Thời gian (ngày)	1.0	1.0	0.6	0.4	1.5	0.3	0.3	0.3	0.6

Sinh viên hãy suy nghĩ về bài toán và thực hiện các bước dưới đây để giúp bạn Thành:

3.1. Xây dựng dữ liệu đầu vào

Như vậy, dữ liệu đầu vào bao gồm: tập hợp các địa điểm có thể đi du lịch, thời gian dự kiến đi du lịch tại các địa điểm trên và số ngày du lịch không vượt quá của đoàn. Chúng ta có thể sử dụng danh sách list để mô tả dữ liệu đầu vào như sau:

```
>>> sodiem_DL = 9

>>> diadiem = ['Cần Giờ', 'Củ Chi', 'Chợ Bến Thành', 'Sở Thú', 'BV thẩm mỹ răng', 'Chí Hòa', 'Dinh Độc lập', 'Bào tàng', 'Trường Văn Lang']

>>> thoigian = [1.0, 1.0, 0.6, 0.4, 1.5, 0.3, 0.3, 0.3, 0.6]

>>> songay_dulich = 1.5   # Nghĩa là đoàn du lịch chỉ đi tham quan 1.5 ngày
```

3.2. Phương pháp "vét cạn" tìm tất cả các giải pháp

Biểu diễn hệ nhị phân: Như chúng ta đã biết, một số tự nhiên có thể biểu diễn ở các hệ số khác nhau như: hệ thập phân, hệ nhị phân. Với hệ nhị phân, một số được biểu diễn là một chuỗi các số 0 và 1. Với các số tự nhiên 0, 1, 2, 3, 4, 5 ta có tương ứng là các số 0, 1, 10, 11, 100, 101 trong hệ nhị phân.

Áp dụng nguyên lý nhân: với 2 số 0 và 1, một chuỗi nhị phân có độ dài n sẽ biểu diễn được tối đa là 2^n giá trị. Lưu ý: trong chương 1, Sinh viên được học qua khái niệm superset của thư viện sympy với kiểu dữ liệu finiteset. Theo đó, superset là tập hợp chứa tất cả các tập con của 1 tập. Mục này diễn giải phương pháp thiết lập superset bằng phương pháp nhị phân

Như vậy, với số nhị phân có 3 chữ số, giá trị biểu diễn tối đa của nó là: $2^3 - 1 = 7$, do tính số 0 là số đầu tiên. Cụ thể như sau:

Hệ thập phân	Hệ nhị phân	Hệ nhị phân có 3 chữ số
0	0	000
1	1	001
2	10	010
3	11	011
4	100	100
5	101	101
6	110	110
7	111	111
8	1000	1000 ← vượt 3 chữ số, 3 chữ số cuối là '000'

Nhận xét: Giả sử, với 1 chuỗi nhị phân s có độ dài n. Ban đầu chuỗi s có n chữ số 0 (nghĩa là: $s = '0 \ldots . 00'$). Sau đó, khi chúng ta lần lượt tăng 1 đơn vị cho s đến khi s đạt giá trị $2^n - 1$ thì s sẽ đầy đủ các trường hợp tổ hợp $'0'$ và $'1'$ tại mọi vị trí.

Từ nhận xét trên, chúng ta tiến hành xây dựng hàm cộng chuỗi nhị phân như sau:

```
>>> def lietke_chuoinhiphan(n):
    danh_sach = []
    #bat_dau = 1
    for i in range(0, 2**n):  # chay den 2^n-1
        gia_tri = i
        chuoi_np = bin(gia_tri).replace('0b', '')
        while (len(chuoi_np) < n):
            chuoi_np = '0' + chuoi_np
        danh_sach.append(chuoi_np)  # them chuoi vao kq
        #print (chuoi_np)
    return danh_sach
```

Với đoạn mã trên, chúng ta sẽ chuyển đổi số ở hệ thập phân sang hệ nhị phân bằng lệnh **bin()**.

Minh họa lệnh bin(): **>>> bin(10)**

…….. ← Sinh viên điền kết quả

Tuy nhiên, do chuỗi nhị phân trả về từ lệnh bin có 2 kí tự nhận dạng nhị phân là '0b' nên chúng ta sử dụng lệnh thay thế replace('0b', '') để xây dựng chuỗi nhị phân thuần. Thử nghiệm:

>>> bin(10).replace('0b', '')

…….. ← Sinh viên điền kết quả

Với chuỗi nhị phân có sẵn, chúng ta sẽ thêm vào vị trí đầu tiên của chuỗi số lượng số 0 để chuỗi có độ dài n với n là số cho trước. Các lệnh dưới đây minh họa **n = 9** (tương ứng với 9 địa điểm):

>>> n = 9

>>> chuoi_np = bin(10).replace('0b','')

>>> while (len(chuoi_np) < n):

chuoi_np = '0' + chuoi_np

>>> chuoi_np

…….. ← Sinh viên điền kết quả

Sau khi nắm vững được đoạn mã trên, từ đó, chúng ta có thể thực thi đọa mã

Thử nghiệm đoạn mã trên:

>>> lietke_chuoinhiphan(3)

…….. ← Sinh viên điền kết quả

>>> lietke_chuoinhiphan(4)

…….. ← Sinh viên điền kết quả

Lưu ý: không nên liệt kê đến 9 vì số lượng phần tử sẽ tương đối lớn ($2^9 = 512$*)*

Với từng chuỗi nhị phân được tạo trong danh sách, chúng ta có thể kiểm tra từng phương án. Ở đây, vị trí thứ i có giá trị bằng '0' trong chuỗi nhị phân nghĩa là chúng ta không ghé thăm địa điểm thứ i. Ngược lại, vị trí thứ i của chuỗi nhị phân có giá trị bằng '1' thì chúng ta sẽ ghé thăm địa điểm đó. Tất nhiên, ràng buộc của bài toán là tổng thời gian du lịch sẽ nhỏ hơn giá trị tổng thời gian (gọi là giá trị giới hạn) cho phép xác định (là **songay_dulich = 1.5**). Với

```
>>> def kiemtra_chuoi(chuoi_dang_chon, danhsach, gioihan):
    tong_thoigian = 0
    for j in range(0, len(chuoi_dang_chon)):
        if chuoi_dang_chon[j] == '1':
            tong_thoigian = tong_thoigian +thoigian[j]
    ketqua = True  # nghia la OK
    if tong_thoigian > gioihan:
        ketqua = False # khong duoc
    return ketqua, tong_thoigian
```

Điểm nhấn trong đoạn mã trên là tìm những vị trí của chuỗi nhị phân mang giá trị $'1'$. Tại các vị trí đó, thời gian sẽ được cộng dồn vào.

Cuối cùng, chúng ta sẽ thực hiện việc tìm phương án bằng việc duyệt tất cả các giá trị trong danh sách phương án tạo từ danh sách các chuỗi nhị phân với module như sau:

```
>>> def tim_phuongan(thoigian, gioihan):
    search = []   # su dung de tra ve
    cac_phuongan = lietke_chuoinhiphan(len(thoigian))
    for i in range(len(cac_phuongan)):
        dc_kg, diem = kiemtra_chuoi(cac_phuongan[i], thoigian, gioihan)
        #print cac_phuongan[i], dc_kg, diem
        if dc_kg == True:  # chi xet cac phuong an dat
            print (cac_phuongan[i], dc_kg, diem)
            search.append(cac_phuongan[i])
    return search
```

Thử nghiệm với dữ liệu trong bảng dữ liệu du lịch bên trên:

>>> songay_dulich = 1.5

>>> tim_phuongan(thoigian, songay_dulich)

…….. ← Sinh viên mô tả về kết quả hiểu được. Ví dụ:

000000001 True 0.6

000000010 True 0.3

000000011 True 0.8999999999999999

000000100 True 0.3

000000101 True 0.8999999999999999

000000110 True 0.6

……
……
……
……
……
……
……

3.3. Bổ sung yêu cầu về tour du lịch

Trên thực tế, việc xây dựng tour du lịch sẽ phức tạp hơn với những ràng buộc và yêu cầu từ khách hàng. Thông thường, nhiều thông tin khác sẽ được bổ sung khi chọn địa điểm bao gồm:

- Xu hướng giảm các loại địa điểm trùng.
- Xu hướng tăng các địa điểm được đánh giá cao (như theo một tiêu chí điểm: bằng lượt Review của khách hàng).
- Lựa chọn những vị trí thay thế những vị trí đang trùng tu, bảo dưỡng hợp lí,…

Sinh viên vui lòng xem phần Bài tập 1. Sinh viên có nhiệm vụ nghiên cứu thực hiện sau buổi học và nộp bài đúng quy định/yêu cầu của Giảng viên phụ trách.

4. Nguyên lý chuồng bồ câu

4.1. Nguyên lý

Nguyên lý chuồng bồ câu (Pigeonhole Principle) là một nguyên lý đơn giản. Tuy vậy, các ứng dụng của nguyên lý rất phong phú và trãi dài ở nhiều lĩnh vực như: xây dựng game trực tuyến, phân tích/nền tảng cho các bài toán toán học,... Ví dụ về nột số phát biểu của nguyên lý và ứng dụng thường thấy:

- Với dãy (n*n+1) số, luôn tìm được dãy con có (n+1) số tăng hoặc giảm.
- Ít nhất có 2 người trên thế giới có cùng số lượng sợi tóc!

Dưới đây là một ứng dụng đơn giản về áp dụng nguyên lý chuồng bồ câu trong thực tế:

4.2. Đọc thêm và bài tập nâng cao: Bài toán ứng dụng 2: Ảo thuật "Tìm Lá bài thứ 5"

Một ảo thuật gia mời một khán giả chọn 5 lá bài bất kì trong bộ bài 52 lá. Ảo thuật gia không được phép xem 5 lá bài được chọn. Tuy nhiên, người trợ giúp cho ảo thuật gia biết 5 lá bài và người trợ giúp sẽ lấy 4/5 lá bài cho ảo thuật gia. Ngay lập tức, ảo thuật gia nhận diện được lá bài thứ 5!

Thông tin về bộ bài: Một bộ bài gồm 52 lá bài được chia thành 4 "chất" Cơ (heart), Rô (diamond), Chuồn (club), Bích (spade). Mỗi "chất" có 13 lá bài từ 1 (con ách, A), 2, 3, 4, 5, 6, 7, 8, 9, 10 và 11 (Jack), 12 (Queen), 13 (King).

Để nhận diện được lá bài thứ 5, ảo thuật gia phải xử lý để có 2 nhóm thông tin: "chất" và số thứ tự của lá bài trong "chất" đó.

03 nhận xét quan trọng:

- *Nhận xét 1*: ***Theo nguyên lý Chuồng bồ câu***: Lá bài thứ 5 phải cùng "chất" với ít nhất 1 lá trong 4 lá bài còn lại. Suy ra, thông tin về "chất" có khả năng được giải mã thông qua thứ tự chọn lá bài của người trợ giúp. Ví dụ: Người trợ giúp chọn lá bài đầu tiên để thông tin về "chất" cho lá bài thứ 5.
- *Nhận xét 2*: **Khoảng cách ngắn nhất giữa 2 lá bài cùng một "chất":** Giả sử thứ tự của các lá bài lần lượt là 1 đến 13 và xoay vòng lại theo đồng hồ. Và gọi khoảng cách giữa 2 lá bài x và y là hàm Distance(x,y) là số lượng lá bài tối thiểu để đi từ lá bài từ lá bài x đến lá bài y. Minh họa: khoảng cách giữa 12 (Q) và 2 là 3 đơn vị (vì … → 12 (Q) → 13 (K) → 1 → 2 → …), kí hiệu là Distance(12, 2)=3. Ví dụ khác: Distance(3, 12) = 9. Như vậy, khoảng cách ngắn nhất giữa hai lá bài 3 và 12 là 3.

  Tương tự (một số ví dụ khác để dễ hiểu hơn):

  - Distance(11, 3) = 5 vì … → 11 (jack) → 12 → 13 → 1 → 2 → 3 → …
  - Distance(1, 7) = 6 vì … → 1 (ách) → 2 → 3 → 4 → 5 → 6 → 7 → …
  - Distance(7, 1) = 7 vì … → 7 → 8 → 9 → 10 → 11 (J) → 12 (Q) → 13 (K) → 1.

  Nhận xét: Khoảng cách ngắn nhất giữa hai lá bài không vượt quá **6** (<=6).

  Chứng minh: Do chọn cách bố trí xoay vòng nên không mất tính tổng quát, ta chọn 1 lá bài là 1 và lá bài còn lại là x. Dưới đây là hai biện luận:

  + Với 1<x<=7 thì Distance(1, x) < = Distance(1, 7) = 6

  + Với 7 < x < 13 thì Distance(x, 1) < Distance(7, 1) = 7. (điều phải chứng minh)

- *Nhận xét 3*: Với **3**/5 lá bài còn lại, chúng ta có 6 cách bố trí chúng (3 lá bài mà có 6 cách bố trí là nguyên lý…… ← sinh viên điền vào). Giả sử **thứ tự theo "hạng"** của

chúng tăng dần được mã hóa là 123, thì các cách bố trí theo ***thứ tự từ điển*** của chúng là: 123, 132, 213, 231, 312 và 321. Cụ thể:

Thứ tự từ điển	123	132	213	231	312	321
Hạng	1	2	3	4	5	6

Từ 3 nhận xét trên, giả định chúng ta có "hạng" của 52 lá bài như sau:

"Chất"	**Lá bài**												
	1 (A)	**K**	**Q**	**J**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Bích	*1*	*5*	*9*	*13*	*17*	*21*	*25*	*29*	*33*	*37*	*41*	*45*	*49*
Cơ	*2*	*6*	*10*	*14*	*18*	*22*	*26*	*30*	*34*	*38*	*42*	*46*	*50*
Chuồn	*3*	*7*	*11*	*15*	*19*	*23*	*27*	*31*	*35*	*39*	*43*	*47*	*51*
Rô	*4*	*8*	*12*	*16*	*20*	*24*	*28*	*32*	*36*	*40*	*44*	*48*	*52*

Ví dụ: Với 5 lá bài theo thứ tự được khán giả chọn: 3 Cơ, 5 Bích, 6 Chuồn, 7 Cơ, 2 Rô.

Người trợ giúp có các thông tin và xử lý như sau:

- **Chất** của từng lá bài theo thứ tự tương ứng: ['Cơ', 'Bích', 'Chuồn', 'Cơ', 'Rô']
- Khoảng cách giữa **2 lá cùng "chất":** Distance(3 Cơ, 7 Cơ) = 4.
- Hạng của từng lá bài theo thứ tự tương ứng: 46 (3 Cơ), 37 (5 Bích), 35 (6 Chuồn), 30 (7 Cơ), 52 (2 Rô).

"Chất"	***1 (A)***	***K***	***Q***	***J***	***10***	***9***	***8***	***7***	***6***	***5***	***4***	***3***	***2***
Bích	1	5	9	13	17	21	25	29	33	**37**	41	45	49
Cơ	2	6	10	14	18	22	26	**30**	34	38	42	**46**	50
Chuồn	3	7	11	15	19	23	27	31	**35**	39	43	47	51
Rô	4	8	12	16	20	24	28	32	36	40	44	48	**52**

- Sau khi chọn 2 lá bài cùng chất là 46 (3 Cơ) và 30 (7 Cơ), thứ tự sắp xếp tăng dần theo hạng của 3 lá bài còn lại là: 35, 37, 52 (xếp theo hạng: 35 < 37 < 52), nghĩa là tương ứng: 6 Chuồn, 5 Bích, 2 Rô. Do đó, nếu sắp xếp theo **thứ tự thứ 4 để tương ứng với distance=4** (theo nhận xét 3), nghĩa là thứ tự 231 thì chúng ta có: 5 Bích, 2 Rô, 6 Chuồn.

Như vậy, người trợ giúp ảo thuật gia sẽ đưa ra thứ tự của 4 lá bài như sau:

- Lá bài thứ 1: 3 Cơ ➔ Để suy ra lá thứ 5 có "chất" là Cơ
- 3 lá bài còn lại sẽ lần lượt xuất hiện là: 5 Bích, 2 Rô, 6 Chuồn. Đây là thứ tự 231 (thứ tự thứ **4**) theo bảng xếp "hạng" của 3 lá bài (37, 52, 35). Từ đó, ảo thuật gia có thể suy ra lá bài thứ 5 có giá trị là 3 + 4 = **7**.

Vậy ảo thuật gia sẽ kết luận: Lá bài thứ 5 là "**7 Cơ**".

BÀI TẬP CHƯƠNG 3

Bài 1: Mở rộng bài toán chọn địa điểm du lịch cho tour:

Đoàn du khách yêu cầu bổ sung: các địa điểm tương tự (nghĩa là cùng loại hình) thì chỉ cần chọn 1 để tham quan. Nghĩa là không đi 2 nơi Dã ngoại.

Tiêu chí	**Danh sách địa điểm**								
Địa điểm	Cần Giờ	Củ Chi	Chợ Bến Thành	Sở thú	BV Thẩm mỹ răng	Chí Hòa	Dinh Độc Lập	Bảo Tàng	Trường Văn Lang
Thời gian (ngày)	1.0	1.0	0.6	0.4	1.5	0.3	0.3	0.3	0.6
Loại hình	Dã ngoại	Dã ngoại/ lịch sử	Văn hóa /muasắm	Dã ngoại/ Trẻ em	Y tế	Lịch sử	Lịch sử	Lịch sử/ văn hóa	Văn hóa

Gợi ý:

- Thêm một danh sách loại_hình. Ví dụ: dữ liệu cho 2 địa điểm Cần Giờ và Củ Chi như bảng được thêm là: loaihinh = [[“Dã ngoai”], [“Dã ngoại”, “Lịch sử”]].
- Với mỗi điểm được chọn, xây dựng một tập hợp loại hình được chọn(dạng dữ liệu set). Mỗi khi thêm một phần tử vào tập hợp loại hình thì kiểm soát số lượng phần tử của tập hợp. Nếu số lượng phần tử tăng là việc thêm vào tập hợp sẽ thành công. Ngược lại, phương án đó sẽ không được sử dụng do có trùng lặp về loại hình.

Bài 2: Mở rộng bài toán chọn địa điểm du lịch cho tour:

Thêm ràng buộc: số lượng địa điểm tối thiểu. Ví dụ: Số lượng địa điểm đi tham qua không ít hơn 2 nơi.

Bài 3* (Bài tập khó): Viết chương trình hoàn chỉnh cho bài toán “Tìm Lá bài thứ 5”.

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Phạm Trọng Nghĩa – Nguyễn Công Nhựt – Trần Ngọc Việt - Hoàng Thị Kiều Anh – Lê Ngọc Thành – Đỗ Đình Thủ – Nguyễn Hữu Trí Nhật – Lê Công Hiếu – Nguyễn Thị Thanh Bình – Nguyễn Thái Hải – Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 4: PHÉP ĐẾM – HOÁN VỊ, TỔ HỢP VÀ CHỈNH HỢP

Mục tiêu:

- *Hoán vị, tổ hợp và chỉnh hợp không lặp và lặp bằng itertool và viết mã lệnh.*
- *Giới thiệu các bài toán ứng dụng tổ hợp trong thực tế.*
- *Giới thiệu vấn đề ngẫu nhiên tác động đến tập nghiệm.*

Nội dung chính:

1. Giới thiêu về hoán vị, tổ hợp và chỉnh hợp

[Giảng viên và sinh viên cùng thảo luận về lý thuyết, ví dụ/minh họa về hoán vị, tổ hợp và chỉnh hợp đã được học. Mục 1 này giảng viên giúp sinh viên nhớ lại những kiến thức đã học].

Tổ hợp, hoán vị và chỉnh hợp là những bài toán giải tích tổ hợp các đối tượng trong một tập hợp có sẵn. Việc xử lý thường để tìm ra một tập hợp hoặc các tập hợp thỏa mãn những điều kiện cho trước.

Ví dụ: xét bài toán sau và tìm tất cả các phương án (nghiệm) có thể:

SẮP XẾP PHÒNG CÁCH LY TẠI TRUNG TÂM CÁCH LY XÃ HỘI TỪ THIỆN

Để hạn chế sự lây lan của bệnh dịch Covid19, một số biện pháp social distancing cần được triển khai. Từ đó, các trung tâm cách ly xã hội được hình thành. Mỗi trung tâm có một số lượng phòng để các ca nghi nhiễm, người đến từ vùng dịch thực hiện nghĩa vụ cách ly.
Tuy vậy, để việc cách ly tốt, một số vấn đề về y tế và sức khỏe cần được quan tâm như sau:

- *Số lượng người* có thể cách ly trong 1 phòng.
- *Sự yên tĩnh*: là yếu tố cần thiết đối với người già hoặc người có bệnh về tim mạch.
- *Giường tầng*: đối với thanh niên có thể sử dụng giường tầng nhưng đối với người già và trẻ em thì hạn chế (không đáp ứng).
- *Giường chuyên dụng*: là giường có chăm sóc y tế, ví dụ: người bị thương tật, người cần khử trùng, người nằm cần các thiết bị truyền dịch (nước biển,…)
- Ngoài ra, còn 1 yếu tố đối với phía các trung tâm là *chi phí vệ sinh/dọn phòng*: đây là chi phí mà trung tâm cách ly sẽ thanh toán cho các đơn vị vệ sinh y tế nhằm thực hiện các công tác về: vệ sinh, khử trùng,…

Vấn đề cần giải quyết: Một Trung tâm cách ly xã hội từ thiện hiện tại đang có 4 phòng trống, với thông tin như sau (lưu ý: chỉ nêu ưu điểm, đặc điểm không mô tả là không có)

- Phòng A: chứa ít hơn 10 người, 1 giường chuyên dụng, yên tĩnh, chi phí vệ sinh phòng: 1 triệu đồng/ngày.

- Phòng B: chứa từ 7 đến 9 người, 2 giường chuyên dụng, yên tĩnh, có giường tầng, chi phí vệ sinh: 1.5 triệu đồng/ngày.
- Phòng C: chứa ít hơn 7 người, có 2 giường chuyên dụng, yên tĩnh, có giường tầng, chi phí vệ sinh là 2.5 triệu đồng/ngày.
- Phòng D: chứa ít hơn 13 người, có 3 giường chuyên dụng, có giường tầng, chi phí vệ sinh là 3 triệu đồng/ngày.

Hiện tại, trung tâm đang tiếp nhận 03 nhóm người cần cách ly xã hội được chuyển đến với các yêu cầu được thu thập như sau:

- Nhóm 1: 6 người, cần 2 giường chuyên dụng, không yêu cầu yên tĩnh, có thể sử dụng giường tầng, yêu cầu cách ly trong 3 ngày.
- Nhóm 2: 10 người, không thể sử dụng giường tầng, không cần yên tĩnh, chỉ yêu cầu cách ly trong 4 ngày.
- Nhóm 3: 8 người, cần yên tĩnh, cách ly trong 8 ngày.

Vấn đề cơ bản nhất của một Cử nhân/Kỹ sư Công nghệ Thông tin:

- Hãy cho biết có bao nhiêu cách sắp xếp các nhóm vào các phòng?
- Cho biết cách xếp nào tốt nhất?
- Câu hỏi suy nghĩ thêm: Cho nhận xét/ý kiến trong trường hợp số lượng phòng và số lượng nhóm tăng? Nghĩa là hãy nêu quan điểm để giải quyết bài toán trên.

[Sinh viên hãy giải và gửi đáp án đến GV trong thời gian tối đa 5 phút]

Trong Python, thư viện itertools trong Python có hỗ trợ nhiều hàm xử lý các yêu cầu của các bài toán tổ hợp.

2. Sử dụng thư viện itertools với các phép toán hỗ trợ xử lý về tổ hợp

Lưu ý: để sử dụng, ta phải import thư viện itertools bằng lệnh, cụ thể là: **>>> import itertools**

Giới thiệu một số lệnh của **thư viện itertools**:

- **Tích tổng tích tụ (accumulate)**: Cho một tập A = {1,2,3,4,5,6,…}. Tổng tích tụ thứ *i* là tổng từ phần tử đầu tiên đến phần tử thứ *i*. Ví dụ:

```
>>> for i in itertools.accumulate([1,2,3,4,5,6,7,8,9,10]):
        print(i)
```

…… ← Sinh viên giải thích kết quả sau đó điền kết quả vào

- **Hàm lặp (repeat)**: Mỗi giá trị có thể được lặp lại nhiều lần

>>> for i in **itertools.repeat**('Red', 3):

    print (i)

…… ← Sinh viên giải thích kết quả sau đó điền kết quả vào

……..

- **Tích Descartes**: cho 2 tập hợp A và B, tích Descartes (còn gọi là Cartesian product) là các cặp giá trị với mỗi giá trị ở mỗi tập A và B.

- Phương pháp 1: Viết đoạn mã thuần bằng Python:

```
>>> for i in ((i,j) for i in [1,2] for j in [6,7,8,9]):
        print(i)
```

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

- Phương pháp 2: Sử dụng itertools:

```
>>> import itertools
>>> for i in itertools.product([1,2],[6, 7, 8, 9]):
        print(i)
```

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

Sinh viên cho biết các kết quả (sau khi đã có lệnh **>>> import itertools**)

>>> for i in itertools.**product**('AB', 'C', 'DEF'):

    print(i)

……..…… ← Sinh viên mô tả hoặc/điền kết quả vào

>>> for i in itertools.**product**('24', 'IT', repeat = 2):

    print (i)

……..…… ← Sinh viên mô tả/điền kết quả vào

- **Hoán vị (permutation)**: (tương ứng với chỉnh hợp chập n từ n phần tử)

```
>>> from itertools import permutations
>>> for i in permutations('ABC'):
        print (i)
```

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

- **Chỉnh hợp chập k từ n (permutation)**:

```
>>> from itertools import permutations
>>> for i in permutations('ABC', 2):
        print (i)
```

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

- **Tổ hợp (combinations)**:

```
>>> for i in itertools.combinations('ABCDE', 3):
        print (i)
```

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

Ví dụ khác:

>>> import itertools

>>> nhaccu = 'Đàn Trống Sáo Bo'.split()

>>> chonmua2mon = list(itertools.**combinations**(nhaccu, 2))

>>> chonmua2mon

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

- **Tổ hợp có lặp (combinations)**:

>>> for i in itertools.**combinations_with_replacement**('ABCDE', 3):

print (i)

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

- **Xoay vòng giá trị cần lấy**:

>>> for i in **itertools.repeat**('Red', 3):

print (i)

…… ← Sinh viên mô tả hoặc điền kết quả vào

……..

3. Ôn luyện cơ bản về Python: Hàm ngẫu nhiên toán học và lặp trong Python

3.1. Cơ bản về hàm ngẫu nhiên trong Python

Python hỗ trợ hàm cung cấp giá trị ngẫu nhiên với thư viện random:

+ Chọn ngẫu nhiên trong tập hợp có sẵn:

>>> import random

>>> random.**choice**(['Táo', 'Lê', 'Ổi', 'Chuối'])

…….. ← Sinh viên điền kết quả

+ Phát sinh số thực ngẫu nhiên trong khoảng số thực [0,1):

>>> random.**random**() **# random số thực (float)**

…….. ← Sinh viên điền kết quả

+ Phát sinh số thực ngẫu nhiên trong khoảng số thực [a,b):

>>> random. **uniform**(4.9, 10.0) **# a = 4.9 và b = 10, số ngẫu nhiên trong khoảng [4.9, 10.0)**

…….. ← Sinh viên điền kết quả

+ Phát sinh số ngẫu nhiên trong khoảng 0 đến 5:

>>> random.**randrange**(6)

…….. ← Sinh viên điền kết quả

+ Phát sinh số ngẫu nhiên trong khoảng 50 đến 500:

>>> random.**randrange**(50, 500)

…….. ← Sinh viên điền kết quả

+ Phát sinh số nguyên chẵn ngẫu nhiên trong khoảng 20 đến 100:

>>> random.**randrange**(20, 100, 2) **# số ngẫu nhiên chẵn do 20 là chẵn và bước nhảy bằng 2**

…….. ← Sinh viên điền kết quả

+ Phát sinh ngẫu nhiên 10 giá trị trong khoảng 0 đến 99:

>>> random.**sample**(**range**(100), 10) **# với Python 2.x câu lệnh là: xrange(100)**

…….. ← Sinh viên điền kết quả

+ Phát sinh ngẫu nhiên 15 giá trị trong khoảng 10 đến 99:

>>> random.**sample**(**range**(10, 100), 15)

…….. ← Sinh viên điền kết quả

+ Phát sinh ngẫu nhiên 5 giá trị trong danh sách [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’]:

>>> chars = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’]

>>> rand5_char = random.**sample**(**chars,** 5)

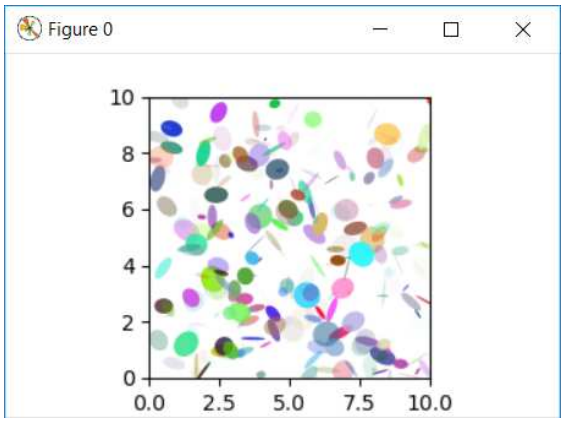
>>> print (rand5_char)

…….. ← Sinh viên điền kết quả

Mục 3.2 và 3.3 chủ yếu giới thiệu về các ứng dụng, bổ sung kiến thức Python và nền tảng toán học cho Sinh viên

3.2. Ứng dụng minh họa: Vẽ ngẫu nhiên Eclipse

Đoạn mã dưới đây sinh viên thử nghiệm về vẽ ngẫu nhiên 250 eclipse. Minh họa kết quả:



Lưu ý: Do hàm vẽ có sử dụng các thư viện đồ họa nên có thể một số máy tính cần cài đặt thêm các thư viện đồ họa.

Sinh viên có thể tạo một tập tin **ellipse.py** như sau để thử nghiệm, trong đó các tham số của các eclipse sẽ được lấy ngẫu nhiên theo hàm ngẫu nhiên **rand()**:

File Edit Format Run Options Windows Help

```
import matplotlib.pyplot as plt
import numpy.random as rnd
from matplotlib.patches import Ellipse

NUM = 250  # vẽ 250 eclipse ngẫu nhiên vị trí, góc:

ells = [Ellipse(xy=rnd.rand(2)*10, width=rnd.rand(),
                height=rnd.rand(), angle=rnd.rand()*360)
        for i in range(NUM)]

fig = plt.figure(0)
ax = fig.add_subplot(111, aspect='equal')
for e in ells:
    ax.add_artist(e)
    e.set_clip_box(ax.bbox)
    e.set_alpha(rnd.rand())
    e.set_facecolor(rnd.rand(3))

ax.set_xlim(0, 10)
ax.set_ylim(0, 10)

plt.show()
```

Lưu ý: Thực thi bằng việc vào menu Run → Run Module (hoặc F5)

3.3. Bài toán ứng dụng 1: Giới thiệu Roullete Wheel – "Chiếc nón kì diệu"

*[*Mục đích chính là **giới thiệu** khái niệm phát sinh tổ hợp trong điều kiện có xác suất].*

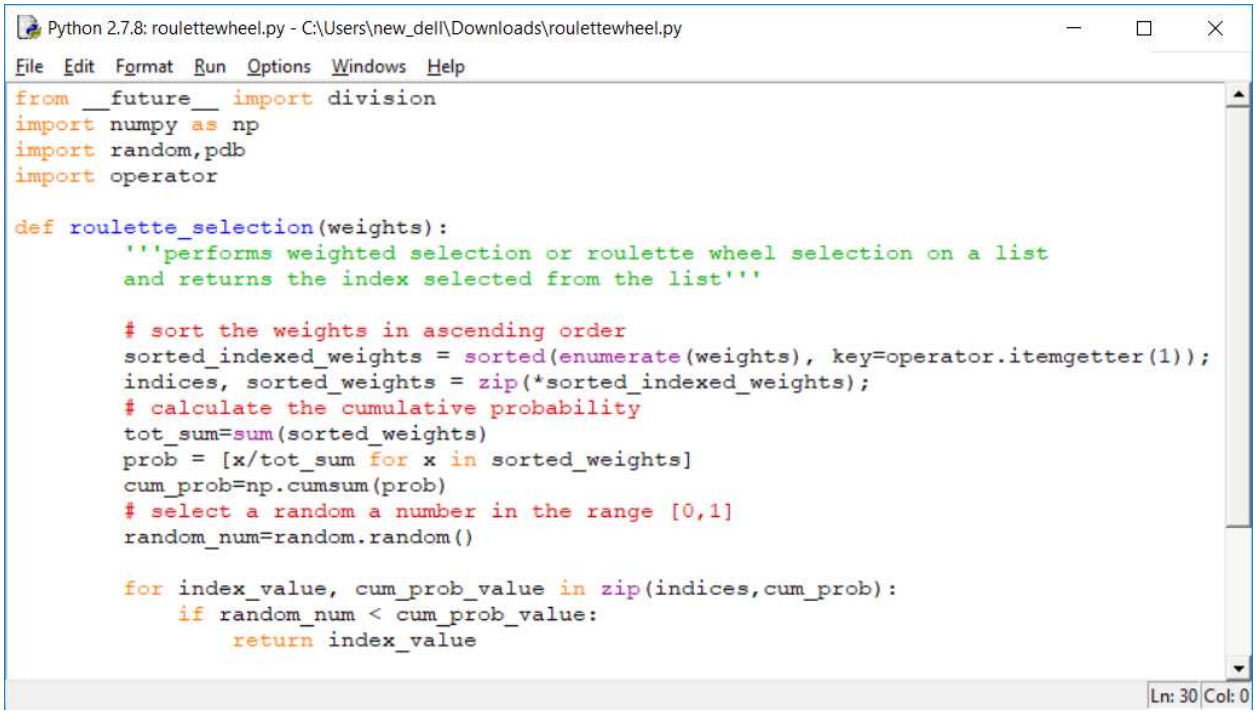
Trong các ví dụ bên trên, các tổ hợp được lấy ra từ tập hợp có sẵn. Do đó, việc tính toán sẽ tuân theo các nguyên lý cộng và nhân nào đó theo bài toán. Tuy nhiên, trong thực tế, một số bộ giá trị được hình thành từ sự kiện/biến cố ngẫu nhiên. Khi đó, tập hợp các giá trị sẽ phụ thuộc rất nhiều vào yếu tố (hay hàm) ngẫu nhiên của hiện tượng/sự việc đó. Dưới đây là minh họa:

Bài toán: Giao lộ Đại lộ Võ Văn Kiệt và An Dương Vương tại TP.HCM có 1 hệ thống đèn tín hiệu giao thông (đèn xanh – vàng – đỏ). Trên hướng đường chính Võ Văn Kiệt, bộ tham số (đèn xanh, đèn vàng, đèn đỏ) theo thời gian (giây) lần lượt là (87, 3, 20).

Vấn đề: Với 20 lần ngẫu nhiên đến giao lộ trên, hãy chô biết số lần gặp đèn xanh, đèn đỏ và đèn vàng là bao nhiêu?

Như vậy, việc gặp tín hiệu đèn xanh, đỏ, vàng có thể xem như là một việc hoàn toàn ngẫu nhiên với người đi trên đường. Tuy vậy, khả năng (hay xác suất) gặp từng loại sẽ khác nhau. Nếu mã hóa thành 3 loại: 0: xanh, 1: vàng và 2: đỏ, thì chúng ta sẽ có các chuỗi ngẫu nhiên như: [0,0,1,2,0,0,2,…], [0,1,1,0,0,2,2,2,…] hoàn toàn khác nhau ứng với mỗi người (hoặc lần thực thi chương trình). Hiển nhiên, chuỗi ngẫu nhiên nhưng sẽ ưu tiên những khoảng giá trị lớn. Ví dụ: đèn xanh chiếm nhiều thời gian hơn thì giá trị 0 sẽ xuất hiện nhiều hơn trong các lần mô phỏng.

Dưới đây là một cài đặt về thuật toán Roullete wheel để các bạn sinh viên thử nghiệm và hiểu hơn về hiện tượng này. Chi tiết về nguyên lý thú vị **Roullete wheel**, còn gọi là bánh xe Roullete – như chiếc nón kì diệu, sẽ được mô tả rõ hơn trong các bài học về thống kê hoặc sinh viên có thể đọc thêm trên mạng để nghiên cứu, tìm hiểu.



Sau đó, các sinh viên sẽ thử nghiệm thực thi:

>>> **xanhdo** =[87, 3, 20] **# khởi tạo các giá trị**

Sinh viên có thể thử nghiệm nhiều lần đoạn lệnh sau và nhận xét (lưu ý: 0 tương ứng với đèn xanh, 1 tương ứng với đènvàng và 20 tương ứng với đèn đỏ):

>>> for i in range(20):

print (**roulette_selection**(xanhdo))

…..

Sau đó, sinh viên có thể thử nghiệm điều chỉnh vector thời gian của đèn xanh/vàng/đỏ như sau:

>>> **xanhdo** =[27, 3, 30] **# khởi tạo các giá trị: đèn xanh gần bằng đèn đỏ**

Và nhận xét các kết quả từ câu lệnh dưới đây với câu lệnh bên trên (về tỉ lệ "gặp" các đèn):

>>> for i in range(20):

print (**roulette_selection**(xanhdo))

…..

Sinh viên có thể tìm hiểu thêm tại: https://vi.wikipedia.org/wiki/Roulette

4. Tập phủ trùm tối thiểu (Set Cover Problem)

Ý tưởng của tập phủ trùm tối thiểu là lựa chọn số lượng các tập con của một tập hợp gồm nhiều tập con để mỗi phần tử của của tập đó đều xuất hiện.

Hơn thế nữa, nếu các phần tử có trọng số thì chúng ta có thể cực tiểu chi phí của tập chọn.

Ví dụ:

Tập $\mathcal{A}$ gồm 10 phần tử {a, b, c, d, e, f, g, h, i, j} chia thành 4 tập con S_1, S_2, S_3, S_4 như sau: $S_1 = \{a, b, i\}, S_2 = \{a, i, j, g, h\}, S_3 = \{d, f, b\}, S_4 = \{c, e, i\}$

Như vậy, chỉ cần 3 tập S_2, S_3 , S_4 là chúng ta có thể có mọi phần tử của tập $\mathcal{A}$.

5. Bài toán ứng dụng 2: Chọn tập các camera quan sát hội chợ

Giả định bài toán như sau: Sân vận động Quân khu 7 tổ chức hội chợ. Sân vận động được chia thành 25 khu vực, mỗi khu vực được đánh số từ 1 đến 25. Trước đó, sân vận động đã lắp đặt 12 camera hỗ trợ an ninh lần lượt được đánh thứ tự là: I, II, III, IV, V, VI, VII, VIII, IX, X, XI và XII. Mỗi camera phụ trách được các khu vực của hội chợ như sau:

Vị trí camera	Khu vực hội chợ triển lãm
I	1, 3, 4, 6, 7
II	4, 7, 8, 12
III	2, 5, 9, 11, 13
IV	1, 2, 18, 19, 21
V	3, 6, 10, 12, 14
VI	8, 14, 15, 16, 17
VII	18, 21, 24, 25
VIII	2, 10, 16, 23
IX	1, 6, 11
X	20, 22, 24, 25
XI	2, 4, 6, 8
XII	1, 6, 12, 17

Bài toán: Là người đi thuê camera để giám sát an ninh, bạn hãy chọn các phương án thuê ít nhất các camera để phủ toàn bộ 25 khu vực hội chợ. Ví dụ: Nhìn vào bảng trên, chúng ta thấy, nếu đã thuê 2 camera **I** và **III** thì sẽ không cần thiết thuê thêm camera **IX**. Vì khu vực 1, 6, 11 đã được giám sát bởi camera I và III.

Phân tích: Bài toán trên có thể giải bằng vét cạn, nghĩa là chúng ta sẽ chọn và thử tất cả các nghiệm, tất cả các tổ hợp của 12 camera. Như việc sử dụng phương pháp xây dựng chuỗi nhị phân và liên tục cộng 1 đơn vị để sau đó kiểm tra chuỗi. Với quy tắc đếm, số lượng trường hợp chúng ta cần phải thử là: $2^{12} = 4096$ trường hợp. Phương pháp vét cạn sẽ tìm được tất cả các

nghiệm. Tuy nhiên, thời gian tìm kiếm và thách thức tính toán sẽ là một vấn đề khi tập hợp tăng lên (như các tập hợp gồm vài ngàn phần tử từ sẽ gặp trường hợp gọi là bùng nổ tổ hợp như để tính $2^k \; với \; k > 1000$).

Một giải pháp khác chúng ta có thể sử dụng là phương pháp Heuristic, nghĩa là phương pháp thử nghiệm gần đúng. Phương pháp Heuristics chỉ đảm bảo việc tìm nghiệm tương đối, nghĩa là thỏa mãn được nghiệm tìm được đúng nhưng việc tối ưu còn phụ thuộc vào chiến lược trong tìm kiếm.

Giả sử với chiến lược: ***"Tại mỗi bước, chọn 1 camera phủ nhiều khu vực chưa được phủ nhất!"***. Khi đó, chúng ta sẽ có các bước sau:

Bước	Camera đã chọn	Các khu vực được phủ	Các khu vực chưa được phủ
1	I	[Phủ được 5 khu vực] 1, 3, 4, 6, 7	2, 5, 8, 9, 10.. 25
2	I, III	[10] 1, 2, 3, 4, 5, 6, 7, 9, 11, 13	8, 10, 12, 14, 15..25
3	I, III, VI	[15] 1..9, 11, 13..17	10, 12, 18..25
4	I, III, VI, VII	[19] 1..9, 11, 13..18, 21, 24, 25	10, 12, 19, 20, 22, 23
5	I, III, VI, VII, VIII	[21] 1..11, 13..18, 21, 23, 24, 25	12, 19, 20, 22
6	I, III, VI, VII, VIII, X	[23] 1..11, 13..18, 20.. 25	12, 19
7	I, III, VI, VII, VIII, X, II	[24] 1..11, 13.. 25	19
8	I, III, VI, VII, VIII, X, II, IV	[25] 1..25	

Diễn giải: Tuy bước 1 và một số bước sau có nhiều lựa chọn khác, ở ví dụ này, chúng ta giả định bước 1 chọn camera I. Và tiếp đó là các bước 2..8 sẽ kế thừa kết quả chọn ở các bước trước để chọn.

Thực hành xây dựng các đoạn module hiện thực ý tưởng trên:

Để thực hiện ý tưởng trên, chúng ta cần những công việc như sau:

- Khởi tạo bảng dữ liệu quan sát các khu vực của 12 camera.
- Các hàm chọn camera sao cho nhiều khu vực chưa được quan sát.

+ Hàm tìm số lượng phần tử có trong tập A mà không có trong tập B:

```
>>> def difference(A, B):
    phantu_moi = 0 # phần tử có trong tập A mà không có trong tập B
    for x in A:
        if x not in B:
            phantu_moi = phantu_moi + 1
    return phantu_moi
```

+ Hàm thêm những phần tử trong tập A mà không có trong tập B vào tập B:

```
>>> def add(A,B):
        ketqua = B
        for x in A:
            if x not in B:
                ketqua = ketqua + [x]
        return ketqua
```

- **Thực hiện xử lý:**

+ Khai báo các camera phủ các khu vực:

```
>>> I = [1, 3, 4, 6, 7]
>>> II = [4, 7, 8, 12]
>>> III = [2, 5, 9, 11, 13]
>>> IV = [1, 2, 18, 19, 21]
>>> V = [3, 6, 10, 12, 14]
>>> VI = [8, 14, 15, 16, 17]
>>> VII = [18, 21, 24, 25]
>>> VIII = [2, 10, 16, 23]
>>> IX = [1, 6, 11]
>>> X = [20, 22, 24, 25]
>>> XI = [2, 4, 6, 8]
>>> XII =[1, 6, 12, 17]
```

+ Khai báo tập các hợp camera và tập hợp các khu vực:

>>> cameras = [I, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII]

>>> khuvuc = range(1,26) # 25 khu vực từ 1 đến 25

+ Xử lý chính:

>>> tapphu = [] # Ban đầu tập phủ là tập rỗng.

>>> while len(tapphu)<len(khuvuc):

    max_new_cameras = 0 # số các camera mới nhiều nhất có thể thêm vào tập phủ

    for camera in cameras:

        max_new_cameras = max(difference(camera, tapphu), max_new_cameras)

    luachon = 0 # duyệt lần nữa để tìm vị trí camera có số khu vực phủ nhiều nhất như trên

    while difference(cameras[luachon], tapphu) < max_new_cameras:

        luachon = luachon +1# vi tri camera se duoc chon

    if (luachon < len(cameras)): # để đảm bảo vị trí cameras là tồn tại

        tapphu = add(cameras[luachon], tapphu)

        print ("Camera: ", luachon+1, cameras[luachon], tapphu) # kết quả các bước

…….. ← Sinh viên điền kết quả

…….. # và so sánh với kết quả

…….. # trong bảng bên trên

+ In ra kết quả và kiểm lại:

>>> tapphu.sort()

>>> print (tapphu)

BÀI TẬP CHƯƠNG 4

Câu 1: Thử nghiệm chạy chương trình Roullete Wheel với nhiều bộ tham số đèn xanh/vàngđỏ khác nhau, như:

[47, 3, 50], [17, 3, 50], [77, 3, 50]

Lưu ý: mỗi bộ dữ liệu thực thi chương trình nhiều lần (khoảng 10 lần).

Câu 2: Phân tích và viết lại chương trình Set Covering bằng:

- Sử dụng dạng tập hợp (set) trong Python.
- Kỹ thuật đệ quy
- In ra nhiều nghiệm (tất cả các nghiệm có thể).

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Phạm Trọng Nghĩa – Nguyễn Công Nhựt – Trần Ngọc Việt - Hoàng Thị Kiều Anh – Lê Ngọc Thành – Đỗ Đình Thủ – Nguyễn Hữu Trí Nhật – Lê Công Hiếu – Nguyễn Thị Thanh Bình – Nguyễn Thái Hải – Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 5: QUAN HỆ TRONG TẬP HỢP

Mục tiêu:

- *Tìm hiểu về các loại quan hệ giữa hai ngôi trên một tập hợp*
- *Biểu diễn quan hệ, suy luận bằng toán học và Python*
- *Các ứng dụng sử dụng quan hệ toán học để xử lý dữ liệu*

Nội dung chính:

1. Dẫn nhập: Bài toán nói dối – nói thật: Knights và Knaves

Bài toán dẫn nhập cho thấy các tập dữ liệu luận lý đôi khi có sự liên hệ (quan hệ với nhau). Mở rộng ra, với các dữ liệu khác, sự liên hệ giữa các tập có thể tồn tại nhiều mối quan hệ. Trong phần dẫn nhập này, chúng ta xét đến quan hệ giữa 2 tập sự kiện có chung đặc điểm liên hệ về thời gian như sau:

Hai ông James và Jonathan đều nói dối vào những ngày nhất định.

James nói dối vào thứ Sáu, thứ Bảy và Chủ Nhật, nhưng nói thật vào tất cả những ngày còn lại.

Jonathan nói dối vào thứ Ba, thứ Tư và thứ Năm, nhưng nói thật vào tất cả những ngày còn lại.

Thế thì vào ngày nào trong tuần cả hai đều nói "**Ngày mai, tôi sẽ nói dối?**"

Các nhận xét:

- Nhận xét 1: Gọi nói dối là Fasle, nói sai là True. Như vậy ta sẽ lập được 1 hàm trả về trị đúng sai (như một hàm Bool với "nói dối" được xem như "mạch đảo", nghĩa là ngược lại).
- Nhận xét 2: Hàm Bool sẽ nhận vào 1 giá trị đầu tiên là Sai (nói dối). Tuy nhiên, giá trị sẽ được tính toán phụ thuộc vào giá trị ngày. Cụ thể: Nếu hôm nay là ngày nói dối (False) thì với giá trị đầu tiên là False thì kết quả sẽ là True. Ngược lại, nếu hôm nay là True thì mai sẽ là False.
- Nhận xét 3: Nghĩa là hàm Bool sẽ có 2 giai đoạn: bước 1: tìm dự báo ngày mai từ giá trị hôm nay và bước 2 là xác định sự đúng đắn từ dự báo và thực tiễn.
- Nhận xét 4: với 1 tuần có 7 ngày và xoay vòng, nếu chọn ngày đầu tiên của tuần là Thứ 2 thì chúng ta cần xét đến ngày Chủ Nhật. Như vậy, với mỗi người James hoặc Jonathan, chúng ta cần mô tả tình trạng nói dối của họ với 7+1 giá trị (để xoay vòng)
- Nhận xét 5: James và Johnathan là 2 bộ dữ liệu khác nhau.

Sinh viên thực hành tìm ngày cả 2 cùng nói câu: "Ngày mai, tôi sẽ nói dối!"!

Bảng "chân trị" của hàm "nói dối":

Sự thật **Hôm nay** *(1)*	Dự báo **Ngày mai** từ câu nói Hôm nay *(2, được suy từ 1)*	Sự thật **Ngày mai** *(3)*	Kết quả so sánh trùng khớp *(giữa 2 và 3)*
Thật	Dối	Thật	Sai
Thật	***Dối***	***Dối***	Đúng
Dối	***Thật***	***Thật***	Đúng
Dối	Thật	Dối	Sai

Bảng chân trị về câu nói của James và Jonathan:

	Thứ 2	Thứ 3	Thứ 4	Thứ 5	Thứ 6	Thứ 7	Chủ nhật	Thứ 2
James	Thật	Thật	Thật	Thật	Dối	Dối	Dối	Thật
Câu nói của James	**Sai**	**Sai**	**Sai**	**Đúng**	**Sai**	**Sai**	**Đúng**	
Jonathan	Thật	Dối	Dối	Dối	Thật	Thật	Thật	Thật
Câu nói của Jonathan	**Đúng**	**Sai**	**Sai**	**Đúng**	**Sai**	**Sai**	**Sai**	

Diễn giải bảng trên: Với James, thứ 2 và thứ 3 đều nói thật. Do đó, nếu thứ 2 James nói: "Ngày mai nói dối" nghĩa là mệnh đề này sai. Tương tự với Chủ nhật và Thứ 2, James nói dối mà nói "Mai nói dối" nghĩa là mệnh đề đó sẽ Đúng.

Đoạn mã Python như sau:

>>> James = [True, True, True, True, False, False, False, True]

>>> Jonathan =[True, True, True, True, False, False, False, True]

>>> ngay_trong_tuan = ["Thu 2", "Thu 3", "Thu 4", "Thu 5", "Thu 6", "Thu 7", "Chu nhat", "Thu 2"]

- *Code chưa rút gọn:*

```
>>> for i in range(0, 6):
        if ((James[i] == True and James[i+1] == False)
            or ((James[i] == False and James[i+1] == True))):
                if ((Jonathan[i] == True and Jonathan[i+1] == False)
                    or ((Jonathan[i] == False and Jonathan[i+1] == True))):
                        print (ngay_trong_tuan[i])
```

…… ← sinh viên điền kết quả tìm được.

- *Code rút gọn:*

Tìm ra quy luật là hôm nay và ngày mai phải khác nhau.

```
>>> for i in range(0,6):
        if ((James[i] and not James[i+1]) and (Jonathan[i] and not Jonathan[i+1])):
            print (ngay_trong_tuan[i])
```

…… ← sinh viên điền kết quả tìm được.

2. Tóm lược: Quan hệ trên các tập hợp và biểu diễn quan hệ

Dẫn nhập: Thông thường, với các phần tử (như số liệu) được cho, ta gọi chúng là một tập hợp, chúng ta cần tìm mối quan hệ giữa các phần tử đó. Có nhiều loại quan hệ giữa hai phần tử với nhau, như quan hệ toán học chia hết, đồng dư (cùng số dư khi chia cho 1 số), lớn hơn, nhỏ hơn,…

Ví dụ: *Cho ngẫu nhiên một dãy số S có thứ tự gồm 10 số tự nhiên từ 1 đến 99. Sau đó, ứng với mỗi số, hãy tìm số lượng các số phía sau nó lớn hơn nó.*

Sinh viên thực tập các câu lệnh sau:

```
>>> import random
>>> S = random.sample(range(1, 100), 10)  # lấy mẫu 10 số tự nhiên trong [1, 99]
>>> print (S)
……
>>> tS = []      # dãy tS lưu trữ số lượng số nằm ở phía sau Si mà lớn hơn Si, i=0,n-1
>>> for i in range(len(S)):
        tS.append(0)
        for j in range(i, len(S)):
            if S[i] < S[j]:
```

tS[i] = tS[i] + 1

>>> print (tS)

……

Quan hệ trên tập hợp:

- Định nghĩa: Một quan hệ hai ngôi từ tập A đến tập B là tập con của tích Descarte $R \subseteq A \times B$.
- Kí hiệu: a R b [thay cho $(a, b) \in R$].

Ví dụ:

- Quan hệ giữa 2 tập: A là tập sinh viên; B là tập lớp học. Khi đó
$$R = \{(a, b) | \text{ sinh } viên\ a\ trong\ lớp\ học\ b\}$$
- Quan hệ ước số: Tập $A = \{1,2,3,4,5,6\}, R = \{(a, b) | a\ là\ ước\ số\ của\ b\}$.Khi đó:

$R = \{(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (2,2), (2,4), (2,6), (3,3), (3,6), (4,4), (5,5), (6,6)\}$

Tính chất:

Cho quan hệ R trên A, quan hệ R được gọi là:

- **Phản xạ**: nếu $\forall a \in A, a\ R\ a$.
  Ví dụ: Quan hệ ước số vì mọi số đều là ước số của nó. Ví dụ: 10 là ước số của 10.
- **Đối xứng**: nếu $\forall a \in A, \forall b \in A: (a\ R\ b) \rightarrow (b\ R\ a)$
  Ví dụ: [Quan hệ đồng dư] $a, b, k \in \mathbb{Z}, a \equiv b\ (mod\ k) \rightarrow b \equiv a\ (mod\ k)$
- **Phản xứng**: nếu $\forall a \in A, \forall b \in A: (a\ R\ b) \wedge (b\ R\ a) \rightarrow (a = b)$
  Ví dụ: Quan hệ $\leq$ trên trường $\mathbb{Z}$ là phản xứng vì với 2 số $a, b \in \mathbb{Z}: (a \leq b) \wedge (b \leq a) \rightarrow (a = b)$.
- **Bắc cầu**: nếu $\forall a, b, c \in A, \forall b \in A: (a\ R\ b) \wedge (b\ R\ c) \rightarrow a\ R\ c$
  Ví dụ: Quan hệ $<$ trong trường số $\mathbb{Z}$, i.e. nếu a nhỏ hơn b và b nhỏ hơn c thì a nhỏ hơn c.

Chúng ta có thể biểu diễn quan hệ của các phần tử trong một tập hợp bằng ít nhất 2 cách:

- Phương pháp 1: Sử dụng ma trận, dãy các vector để mô tả các quan hệ trong một tập hợp.
- Phương pháp 2: Thông qua một hàm (module) có 2 tham số là 2 phần tử của tập hợp. Nếu hai phần tử có quan hệ thì hàm (module) sẽ trả về giá trị là True (đúng), ngược lại là False (sai).

Phương pháp 1: **Biểu diễn quan hệ R của tập A** với tập B bằng ma trận

- Biểu diễn bằng ma trận, gọi là ma trận M với các phần tử là m_{ij} với i biểu thị dòng và j biểu thị cột.
- Mỗi dòng là từng phần tử của tập A.

- Mỗi cột là từng phần tử của tập B.
- Hai phần tử có quan hệ được mô tả bằng giá trị 1, ngược lại là giá trị 0.

Sinh viên thực hiện các lệnh thực tập sau:

>>> import numpy as np

Sinh viên tạo 1 tập tin Excel có tên là monhoc.xlsx và lưu vào ổ đĩa D: có nội dung như sau:

Môn học (A1)	**Lập trình cơ bản**	**Đại số**	**Toán rời rạc**	**Vật lý**	**Hóa học**	**Thể dục**
Môn học (A2)	*(Lưu ý: giá trị 1: là môn ở dòng cần được học trước môn ở cột)*					
Lập trình căn bản	0	1	1	0	0	0
Đại số	0	0	0	0	0	0
Toán rời rạc	0	0	0	0	0	1
Vật lý	0	0	0	0	0	0
Hóa học	0	0	0	0	0	0
Thể dục	1	1	1	1	1	0

Từ đó, GV hướng dẫn sinh viên đọc dữ liệu từ Excel thành ma trận các quan hệ theo bảng trên:

```
>>> import pandas
>>> df = pandas.read_excel('D:\\monhoc.xlsx')
>>> print (df.columns)  # liệt kê các cột dữ liệu
……
>>> giatri = df['<tên cột>'].values     # đọc 1 cột dữ liệu
……
```

Sau đó thực hiện các lệnh đọc dữ liệu vào mảng/ma trận của Python/numpy để xử lý tiếp tục các yêu cầu bên dưới, nghĩa là *xem xét các tính chất ma trận*.

Suy ra [*cách kiểm chứng ma trận quan hệ với các tính chất*]:

- Quan hệ phản xạ: ma trận vuông (do $A = B$) và đường chéo chính (\) bằng 1, i.e. $m_{ii} = 1, \forall i$
- Quan hệ đối xứng: các trị 0, 1 đối xứng qua đường chéo chính (\), i.e. $m_{ij} = m_{ji}, \forall i, j$
- Quan hệ phản xứng: ma trận vuông (do $A = B$) và các trị 0, 1 đối xứng qua đường chéo phụ (/), i.e. $m_{ij} = 0 \vee m_{ji} = 0 \rightarrow i \neq j$
- Quan hệ bắc cầu: $m_{ij} = 1, m_{jk} = 1 \rightarrow m_{ik} = 1$

Hãy viết chương trình bằng Python kiểm tra các mối quan hệ (cho trong một ma trận) có bị chồng chéo nhau hay không? Ví dụ: i phụ thuộc j, j phụ thuộc k và k phụ thuộc i.

Gợi ý: SV có thể xem lại Bài toán loan tin (Chương 3, Thực hành Đại số Tuyến tính)

Quan hệ tương đương: là quan hệ khi có các tính chất **phản xạ**, **đối xứng** và **bắc cầu**.

Khái niệm lớp tương đương: Ví dụ lớp tương đương modulo 8 chứa 3.

Quan hệ thứ tự: là quan hệ khi có các tính chất: phản xạ, phản xứng, bắt cầu.

Ví dụ: quan hệ $\leq$ trong số thực, số nguyên; quan hệ ước số trong số nguyên.

Với tập số nguyên dương:

- Quan hệ $\leq$ là quan hệ thứ tự toàn phần trong tập. Vì mọi số đều có quan hệ.
- Quan hệ "ước số" không phải quan hệ thứ tự nhưng không toàn phần vì tồn tại các phần tử không phải ước số của nhau.

Thứ tự từ điển:

Cho $(A, \preccurlyeq)$ và $(B, \preccurlyeq)$ là 2 tập thứ tự toàn phần. Thứ tự từ điển $\prec$ trên $A \times B$ như sau:

$$(a_1, b_1) \prec (a_2, b_2) \; nếu \; (a_1 < a_2) \vee (a_1 = a_2 \wedge b_1 < b_2)$$

Ví dụ:

Tập bảng chữ cái $\Sigma = \{a, b, c\}, với\ a < b < c$ và $\lambda \in \Sigma^*$ là chuỗi rỗng thì thứ tự từ điển là: $\Sigma^* = \{\lambda, a, aa, ab, ac, aaa, aab, aac, aba, abb, \dots, b,\ ba, bb, bc, \dots, c, ca, cb, cc, \dots\}$

Hoặc: $\Sigma = \{0,1\}$ với $0 < 1$ thì $0111110 \prec 10$

PHẦN 3 VÀ 4 SINH VIÊN ĐỌC ĐỂ CÓ KIẾN THỨC (BIẾT VỀ BÀI TOÁN)

3. [Đọc thêm] Bài toán ứng dụng 1: Biểu diễn cơ sở dữ liệu

Trong thực tế, quan hệ giữa các tập hợp được ứng dụng trong các mô hình lưu trữ dữ liệu. Ví dụ: hai tập hợp Sinh viên và Môn học sẽ có mối quan hệ với nhau. Cụ thể là mỗi ***Sinh viên*** sẽ có

quan hệ ***Học*** các ***Môn học***: **Học(Sinh viên, Môn học)**. Với mô tả này, chúng ta sẽ có 2 tập hợp cơ bản là Sinh viên và Môn học. Và dưới đây là một bảng dữ liệu minh họa các mô tả quan hệ Sinh viên và Môn học:

Môn học:	**Lập trình cơ bản**	**Đại số**	**Toán rời rạc**	**Vật lý**	**Hóa học**	**Thể dục**
Sinh viên	*(Lưu ý: giá trị 1: Sinh viên có học môn tương ứng; 0: ngược lại)*					
Văn Hậu	1	1	0	1	1	0
Quang Hải	0	0	1	1	0	1
Xuân Trường	1	1	0	0	1	1
Văn Đức	0	0	1	1	1	1
Đức Chinh	1	0	0	1	1	0
Văn Toàn	1	1	0	1	1	0

Tất nhiên, đối với mỗi tập, những quan hệ trong chính tập đó cũng là một thành phần cần biết. Với dữ liệu thực tế, chúng ta phải xem xét các quan hệ mô tả với sự hợp lý. Ví dụ: với Môn học, chúng ta có thể có quan hệ giữa các môn học là: ***Tiên quyết*** (môn cần học trước), như sau:

Môn học:	**Lập trình cơ bản**	**Đại số**	**Toán rời rạc**	**Vật lý**	**Hóa học**	**Thể dục**
Môn học	*(Lưu ý: giá trị 1: là môn ở dòng cần được học trước môn ở cột)*					
Lập trình căn bản	0	1	1	0	0	0
Đại số	0	0	0	0	0	0
Toán rời rạc	0	0	0	0	0	1
Vật lý	0	0	0	0	0	0
Hóa học	0	0	0	0	0	0
Thể dục	1	1	1	1	1	0

Với bảng trên, chúng ta phải điều chỉnh lại giá trị trong quan hệ Tiên quyết giữa 2 môn Toán rời rạc và Thể dục. Vì ở dòng môn 'Toán rời rạc' yêu cầu học trước môn 'Thể dục'. Tuy nhiên, ở dòng cuối môn 'Thể dục' cũng yêu cầu cần phải học trước môn 'Toán rời rạc'! Điều này có nghĩa là: ***Quan hệ môn Tiên quyết là quan hệ có thứ tự***. Tiên quyết để chỉ đến thời gian học trước và sau. Do vậy, 2 môn trên cần sửa đổi thứ tự để có điều kiện **phản xứng**! (Giả định điều chỉnh vị trí dòng Toán rời rạc, cột Thể dục sang giá trị 0)

Toán rời rạc	0	0	0	0	0	0

Hơn thế nữa, một số quan hệ sẽ mô tả sự liên quan của nhiều tập hợp. Ví dụ: Quan hệ Học được bổ sung thêm Học kỳ để biết được chính xác Sinh viên học môn học đó ở Học kỳ (bao gồm học kỳ, niên khóa), cụ thể quan hệ được tạo thành từ bộ 3 tập hợp: Học(Sinh viên, Môn học, Học kỳ).

Từ dữ liệu trên, chúng ta có thể tìm kiếm được những thông tin cần thiết. Dưới đây là một số truy vấn cơ bản trên dữ liệu mà chúng ta có thể xây dựng được:

- Sinh viên X đã học được những môn gì trong học kì này.
- Kết hợp với môn tiên quyết để suy ra sinh viên X đã học được tất cả những môn gì.

Như vậy, về cơ bản, sinh viên sẽ hiểu được phương pháp lưu trữ dữ liệu là thiết lập những quan hệ của các tập dữ liệu. Hiển nhiên, để thể hiện dữ liệu trong cơ sở dữ liệu, chúng ta còn nhiều kỹ thuật khác để giải quyết các vấn đề gặp phải trong: tìm kiếm, lưu trữ, thống kê,.... Các môn học như thiết kế cơ sở dữ liệu ở các học kỳ sau sẽ giúp các bạn sinh viên nắm rõ hơn. Ví dụ: các mô hình dữ liệu tuân theo các chuẩn 1, 2, 3,…; mô hình dữ liệu Nosql,…

4. [Đọc thêm] Bài toán ứng dụng 2: Hợp lý hóa điều kiện tìm kiếm trên dữ liệu

Việc hiểu rõ về cấu trúc dữ liệu, những quan hệ trong dữ liệu và bằng các biến đổi luận lý, chúng ta dễ dàng thay đổi các điều kiện tìm kiếm trên dữ liệu nhằm tăng tốc tính toán. Ứng dụng dưới đây liên quan đến luật De Morgan nổi tiếng:

$$\overline{A \cup B} = \bar{A} \cap \bar{B}$$

$$\overline{A \cap B} = \bar{A} \cup \bar{B}$$

với $\bar{A} = \{x \in U, x \notin A\}, A \subset U$

Chúng ta xét 2 bài toán sau:

- ***Kiểm tra sự "Tách rời" (disjoint) của hai tập đối tượng:***

Cho 2 đối tượng, mỗi đối tượng là một tập hợp. Phép kiểm hai đối tượng ***tách rời*** được định nghĩa như sau:

$$disjoint(A, B) \leftrightarrow \nexists x: (x \in A \wedge x \in B) \; với \; A, B \; là \; 2 \; tập \; hợp$$

Bài toán minh họa: Cho 3 tập dữ liệu Trẻ em (gọi là tập A), Phụ nữ/Giới nữ (B) và Bệnh nhân sởi (C). Hãy tìm ***tập những người không phải là bệnh nhân sởi mà là trẻ em hoặc phụ nữ***. Biểu thức toán học chúng ta cần tìm như sau:

$$!\,((A \cup B) \cap C)$$

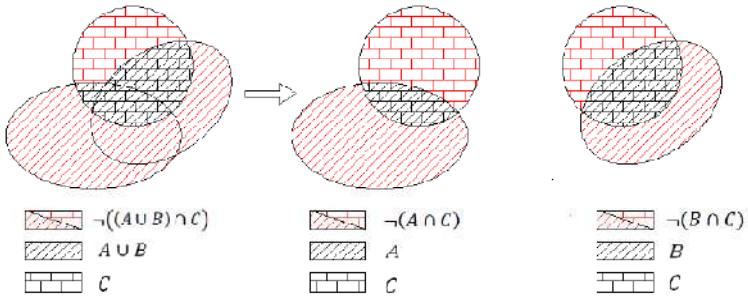
Biến đổi bằng luật phân phối, ta có:

$$!\,\big((A \cup B) \cap C\big) = !\,((A \cap C) \cup (B \cap C))$$

Áp dụng định luật De Morgan:

$$\ldots = !\,\big((A \cap C) \cup (B \cap C)\big) = !\,(A \cap C) \cap !\,(B \cap C)$$

Như vậy, về luận lý, chúng ta thấy rằng hai biểu thức: biểu thức đầu tiên $!(A \cup B) \cap C)$ và biểu thức thứ 2 $!(A \cap C) \cap !(B \cap C)$ là như nhau.



Tuy nhiên, trong thực tế, tùy thuộc vào dữ liệu, việc xử lý các biểu thức sẽ có độ phức tạp khác nhau và dẫn đến tốc độ xử lý khác nhau vì các lí do và giải thích sau:

- Việc tính toán tập $(A \cup B)$ sẽ tốn nhiều thời gian để xử lý do trên thực tế tập trẻ em và tập phụ nữ rất lớn. Khả năng chiếm trên 60% dữ liệu. Sau đó, với lượng dữ liệu đó, chúng ta thực hiện phép $\cap$ với tập C là tập bệnh nhân là một tập không lớn trên thực tế. Việc xử lý phải tuần tự từng bước.
- Hai phép xử lý $(A \cap C)$, $(B \cap C)$ và sau đó là phép lấy phủ định có thể độc lập với nhau nên việc xử lý hoàn toàn tách biệt. Các hệ thống có thể chia thành 2 tiến trình/luồng cùng lúc thực thi.

- ***Phép toán kiểm tra sự "Khác nhau" (difference) của hai tập đối tượng:***

Phép kiểm ***khác nhau*** được định nghĩa như sau:

$$difference(A, B) \leftrightarrow (x \in A \backslash B) \leftrightarrow (x \in A \wedge x \notin B) \text{ với } A, B \text{ là 2 tập hợp}$$

Bài toán minh họa: Cho 3 tập dữ liệu Trẻ em (gọi là tập A), Phụ nữ/Giới nữ (B) và Bệnh nhân sởi (C). Hãy tìm tập những Trẻ em không là Giới nữ mà không nhiễm bệnh sởi. Nghĩa là chúng ta cần tìm tập hợp sau:

$$A \backslash (B \backslash C)$$

Thay thế phép hiệu \ bằng phép $\cap$, ta có:

$$A \backslash (B \backslash C) = A \backslash (B \cap \bar{C})$$

Áp dụng định luật De Morgan cho vế phải đẳng thức trên, ta có:

$$\ldots = A \backslash (B \cap \bar{C}) = (A \backslash B) \cup (A \backslash \bar{C})$$

Áp dụng lần nữa thay thế phép hiệu \ bằng phép $\cap$, ta có:

$$\ldots = (A \backslash B) \cup (A \backslash \bar{C}) = (A \backslash B) \cup (A \cap C)$$

Như vậy, tương tự như trên, về luận lý, chúng ta thấy rằng hai biểu thức: biểu thức $A \backslash (B \backslash C)$ và biểu thức $(A \backslash B) \cup (A \cap C)$ là như nhau. Tuy nhiên, trong thực tế, tùy thuộc vào dữ liệu, việc xử lý các biểu thức sẽ có độ phức tạp khác nhau và dẫn đến tốc độ xử lý khác nhau vì các lí do và giải thích sau:

- Việc tính toán tập $A \backslash (B \backslash C)$ sẽ phải tuần tự thực hiện các bước.
- Hai phép xử lý $(A \backslash B)$ và $(A \cap C)$ có thể xử lý cùng lúc khi thực thi.

PHẦN DƯỚI ĐÂY GIẢNG VIÊN CUNG CẤP KIẾN THỨC CƠ BẢN, SINH VIÊN THỰC HÀNH VÀ THỬ NGHIỆM NHƯ BÀI TẬP VỀ NHÀ:

5. Khái niệm về lập trình logic

*Lưu ý 1: Phần này tập trung **giới thiệu** sinh viên **sự tồn tại và các khái niệm cơ bản nhất về lập trình và ngôn ngữ lập trình logic**. Chi tiết hơn để lập trình logic trên các ngôn ngữ như Prolog cũng như khai thác chi tiết các hàm trong gói thư viện PySWIP sẽ được học ở các môn chuyên ngành hoặc sinh viên có thể tự khám phá thêm. Do đó, phần này là phần bổ sung trong lớp học, chỉ khuyến khích học sinh nghiên cứu, tìm hiểu và tham khảo thêm chứ không bắt buộc.*

Lưu ý 2: Sinh viên nghiên cứu nên tự xem phương pháp cài đặt các gói thư viện phần mềm để làm thêm ở nhà.

5.1. Giới thiệu về lập trình logic và gói PySWIP

- **Khái niệm chương trình logic**

Từ nguyên lý cơ bản: *"Mọi thuật giải có thể biểu diễn bằng tập quy tắc với 3 cấu trúc: tuần tự, chọn và lặp (đệ quy)"*, một nhánh lập trình gọi là lập trình logic được ra đời. Lập trình logic có thể xem như việc thể hiện tập các luật, mỗi luật là sự kết hợp của các mệnh đề mệnh đề và mỗi mệnh đề là một dạng tân từ. Từ đó, người lập trình chỉ việc khai báo, còn việc xử lý luận lý để ra kết quả là do hệ thống.

Ví dụ: người lập trình khai báo luật: tổng 3 góc tam giác bằng 180, sau đó cho biết 2 góc thì hệ thống sẽ suy luận ra giá trị góc thứ 3.

Hiện tại, ngôn ngữ Prolog là ngôn ngữ lâu đời và nổi tiếng về lập trình logic.

- **Các thành phần trong chương trình logic:**

Tân từ có dạng như sau: $f(a_1, a_2, \dots, a_n)$ *là tân từ* $\leftrightarrow f: D_1 \times D_2 \times \dots \times D_n \rightarrow \{True, False\}$

Các D_i là các miền cho trước và $a_i \in D_i$. Mệnh đề (clause) là dạng các tân từ.

Các **luật** (rule) là tập hợp các mệnh đề: $p(\dots) :- q_1(\dots), q_2(\dots), \dots, q_n(\dots)$

Chương trình logic: $\{Rule: p_1(\dots), p_2(\dots), \dots\}$

Ví dụ: 2 mô tả quan hệ Ông với sự trợ giúp của mô tả quan hệ Cha mẹ như sau:

Ông(X, Y) :- Chame(Z, Y)

Ông(X, Y) :- Cha(X, Z), Chame(Z, Y)

Giải thích: ở luật thứ 1: nếu X là ông của Y thì tồn tại một Z là Cha hoặc Mẹ của Y; đồng thời luật thứ 2 mô tả là: X là **Cha** của Z hoặc Z có quan hệ **Cha mẹ** của Y (dấu , ở luật thứ 2).

- **Cấu trúc một chương trình logic:**

Một chương trình logic thường gồm 3 phần:

- **Phần khai báo các mệnh đề** (Predicates), khai báo các hằng, biến, tân từ,...
- **Phần thân chương trình** (Clauses): khai báo cụ thể các quy luật, quy tắc (dữ liệu).
- **Phần đích** (kết luận, Goal).

5.2. Cài đặt gói pyswip để minh họa các suy diễn luận lý trong Python

- **Giới thiệu về gói phần mềm PySWIP:**

PySWIP là một gói phần mềm Python làm cầu nối để thực thi các chương trình ngôn ngữ Prolog với những dòng lệnh theo dạng ngôn ngữ Python. Lõi của PySWIP là gói SWI-Prolog, một mã nguồn mở được phát triển từ năm 1987 được sử dụng nhiều trong web ngữ nghĩa.

Yêu cầu cài đặt:

* Python phiên bản 2.3 hoặc cao hơn.

* ctypes phiên bản 1.0 hoặc cao hơn.

* SWI-Prolog phiên bản 5.6.x hoặc cao hơn (phù hợp với các phiên bản khác).

* Thư viện chia sẻ libpl.

* Trên hệ thống Linux và Win32, trên tất cả các hệ POSIX.

- **Chi tiết cài đặt:**

Cài đặt bằng lệnh pip install. Pip install là 1 lệnh trong thư mục Scripts:

Từ thư mục Anaconda3 được cài đặt, thực hiện lệnh: **Scripts\pip install pyswip**

Ví dụ: Anaconda được cài đặt ở ổ **C:\Anaconda3** thì:

C:\Anaconda3> Scripts\pip install pyswip

```
C:\Anaconda3>scripts\pip install pyswip
Collecting pyswip
  Downloading https://files.pythonhosted.org/packages/60/cd/7defc05e52763f142865
45331b8e865ff829304befcaf22c4001f5706b77/pyswip-0.2.8-py2.py3-none-any.whl
Installing collected packages: pyswip
Successfully installed pyswip-0.2.8
```

Lưu ý: Nếu tập tin pip đã cũ (có phiên bản cập nhật mới thì sử dụng lệnh Python để cập nhật):

- Tập tin pip.exe đã cũ và được yêu cầu cập nhật mới:

```
You are using pip version 9.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.
```

- Thực hiện việc cập nhật phiên bản mới:

```
C:\Anaconda3>python -m pip install --upgrade pip
Cache entry deserialization failed, entry ignored
Collecting pip
  Using cached https://files.pythonhosted.org/packages/d8/f3/413bab4ff08e1fc4828
dfc59996d721917df8e8583ea85385d51125dceff/pip-19.0.3-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 9.0.1
    Uninstalling pip-9.0.1:
      Successfully uninstalled pip-9.0.1
Successfully installed pip-19.0.3
```

Sinh viên có thể tham khảo thêm thông tin tại: https://github.com/f0ma/pyswip3

Trang web download: https://code.google.com/archive/p/pyswip/downloads

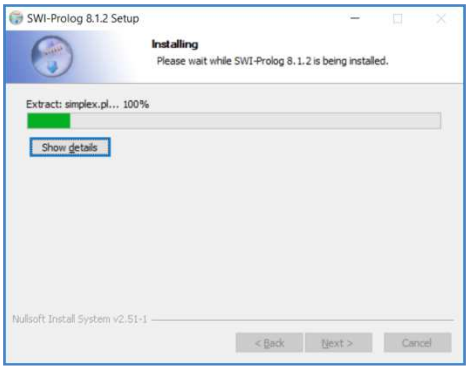
*Lưu ý: Trong Windows, gói pyswip có thể bị lỗi cài đặt (do thiếu thư viện libswipl), như sau:

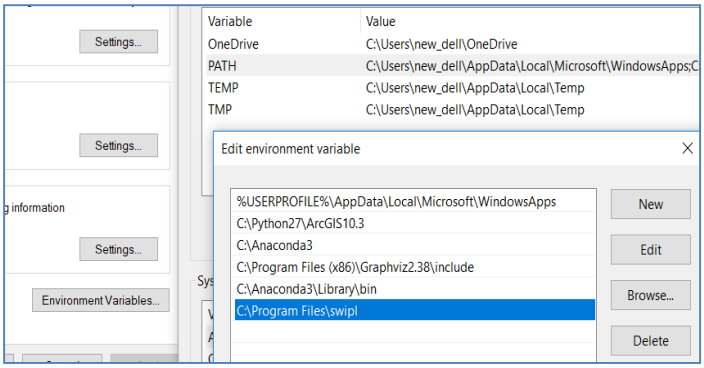
```
Command Prompt

C:\Anaconda3>Scripts\pip install pyswip
pip is configured with locations that require TLS/SSL, however the ssl module in Python is not available.
Requirement already satisfied: pyswip in c:\anaconda3\lib\site-packages (0.2.8)
pip is configured with locations that require TLS/SSL, however the ssl module in Python is not available.
Could not fetch URL https://pypi.org/simple/pip/: There was a problem confirming the ssl certificate: HTT
Max retries exceeded with url: /simple/pip/ (Caused by SSLError("Can't connect to HTTPS URL because the S
```

Khi đó, chúng ta nên kiểm tra việc thiết lập đường dẫn (path) đối với thư mục **Anaconda\Library\bin** và thực hiện việc cài đặt bổ sung gói **SWI-Prolog** tương thích với **Windows** tại địa chỉ: http://www.swi-prolog.org/download/devel (/devel hoặc /stable) cùng với việc thiết lập đường dẫn Path (tại Enviroment Variables cho thư mục cài đặt). Sau đó, chúng ta tắt và khởi động trình IDLE để tải lại các thư viện.

Hiển nhiên, chúng ta có thể xóa gói bằng lệnh **pip uninstall pyswip** và thực hiện cài đặt mới.





5.3. Minh họa sử dụng gói pyswip

Dưới đây là một số ví dụ đơn giản sử dụng gói pyswip.

Lưu ý:

- *Dữ liệu đều phải viết thường, không được viết hoa;*
- *Hạn chế khoảng trắng, ví dụ không có khoảng trắng: phaichet(X):-connguoi(X);*
- *Các từ khóa phải viết như mô tả, như: X, Y, What, Which.*

- **Ví dụ 1: Liệt kê dữ liệu tác giả và tác phẩm**

Sinh viên thực hành nhập các lệnh dưới đây:

>>> **from** pyswip **import** Prolog

>>> tgtph = **Prolog**() # khai báo quan hệ giữa tác giả tác phẩm là một đối tượng Prolog

>>> tgtph.**assertz**('tacgiatacpham(nguyendu,truyenkieu)') # nhập dữ liệu nguyendu, truyenkieu

>>> tgtph.**assertz**('tacgiatacpham(nguyendu,thanhhienthitap)')

>>> tgtph.**assertz**('tacgiatacpham(nguyendu,namtrungtapngam)')

>>> tgtph.**assertz**('tacgiatacpham(nguyendu,bachanhluctap)')

>>> list(tgtph.**query**('tacgiatacpham(nguyendu, **X**)')) # truy vấn kết quả đã nhập

…….. ← sinh viên điền kết quả.

……..

>>> for tg in tgtph.query('tacgiatacpham(X, Y)'): # truy vấn : liệt kê toàn bộ

print(tg["X"], ' viet ra tac pham: ', tg["Y"])

…….. ← sinh viên điền kết quả.

……..

……..

……..

- **Ví dụ 2: Suy luận tam đoạn luận:**

Tam đoạn luận là một cách suy luận trong diễn dịch.

Mệnh đề: 'Mọi người đều phải chết!'

Dữ liệu: Ông Micheal Jackson là người

Suy ra: ông Micheal Jackson phải chết

Sinh viên thực hiện các bước sau:

- Khai báo mệnh đề: phaichet(X) với X là đối tượng connguoi.
- Khai báo dữ liệu connguoi, như: micheal_jackson,… (bằng lệnh **assertz**)
- Tạo ra các truy vấn, nghĩa là kết quả: (Which) là con người? (What) phải chết?

```
>>> from pyswip import Prolog
>>> p = Prolog()
>>> p.assertz('phaichet(X):-connguoi(X)')
>>> p.assertz('connguoi(micheal_jackson)')
>>> p.assertz('connguoi(trinh_cong_son)')
>>> p.assertz('connguoi(vo_tong)')
>>> list(p.query('connguoi(Which)'))
```

…….. ← sinh viên điền kết quả.

```
>>> list(p.query('phaichet(What)'))
```

…….. ← sinh viên điền kết quả.

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Hoàng Thị Kiều Anh – Khưu Minh Cảnh – Phạm Trọng Nghĩa –Nguyễn Công Nhựt – Trần Ngọc Việt – Lê Ngọc Thành – Đỗ Đình Thủ – Nguyễn Hữu Trí Nhật – Lê Công Hiếu – Nguyễn Thị Thanh Bình – Nguyễn Thái Hải – Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 6: CƠ BẢN VỀ ĐẠI SỐ BOOL, FINITE STATE MACHINE

Mục tiêu:

- *Khái niệm về đại số Bool*
- *Biểu diễn Finite State Machine trong Python*

Nội dung chính:

1. Cơ bản về đại số Bool với Python

Tóm tắt lý thuyết: Cho $B = \{0,1\}$, biến x được gọi là biến bool nếu nó nhận giá trị 0 hoặc 1 trong B. Khi đó, một hàm số $f(x_1, x_2, \dots, x_n)$ được gọi là hàm bool nếu nó xác định trên tập

$$B^n = \{(x_1, x_2, \dots, x_n) | x_i \in B, i = \overline{1,n}\}$$

Các đặc tính: cho hai hàm số bool, phép toán hội ($\wedge$), phép toán tuyển ($\vee$) cho hai hàm số và phép toán phủ định cho một hàm số đều sẽ cho kết quả là một hàm số bool. Các kí hiệu:

- Hội của hai hàm bool: $(f \wedge g)$.
- Tuyển của hai hàm bool: $(f \vee g)$.
- Phủ định của hàm bool: $\bar{f}$.

Ví dụ 1: Cho hàm bool theo bảng sau và hãy xác định biểu thức bool và thể hiện bằng ngôn ngữ Python:

x	y	Hàm f(x,y)
1	1	0
1	0	1
0	1	0
0	0	0

Nhận xét: f(x,y)=1 khi và chỉ khi x=1, y=0 và f(x,y)=0 trong các trường hợp còn lại của giá trị x, y. Từ đó, chúng ta có thể kết luận biểu thức bool của hàm f(x,y) là $x \wedge \bar{y}$.

Với Python, chúng ta có thể viết các hàm thể hiện hàm bool. Một cách đơn giản là liệt kê tất cả các trường hợp tương ứng với giá trị của hàm bool cho trong bảng:

Sinh viên thực hiện thể hiện của hàm f như bên dưới:

```
>>> def bool_xy(x, y):
        kq = 0
        if (x==1) and (y==1):
```

```
        kq = 0
    if (x==1) and (y==0):
        kq = 1
    if (x==0) and (y==1):
        kq = 0
    if (x==0) and (y==0):
        kq = 0
    return kq
```

Thực hiện chạy chương trình:

```
>>> for x in [1,0]:
    for y in [1,0]:
        print (bool_xy(x,y))
```

……..

Cải tiến hàm theo phân tích:

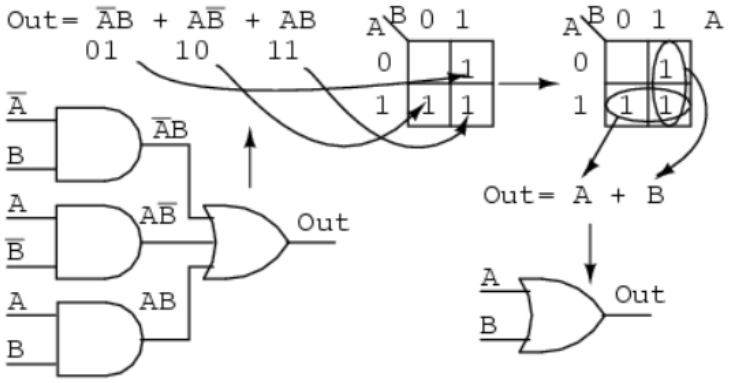
```
>>> def bool_xy1(x,y):
    kq = 0
    if (x == 1) and (y==0):
        kq = 1
    return kq
```

Thực hiện chạy chương trình lần nữa:

```
>>> for x in [1,0]:
    for y in [1,0]:
        print (bool_xy(x,y))
```

……..

Ví dụ 2: Trong hình bên dưới, hai mạch được thiết kế với cùng một chức năng. Trong đó, mạch bên trái nhiều linh kiện hơn mạch bên phải nhưng cùng một tính năng.



Sinh viên hãy viết chương trình bằng Python để mô tả hai mạch bên trên.

Lưu ý: Trong thực tế, mặc dù mong muốn ban đầu của đại số Bool là tối thiểu một thiết kế "mạch" nhưng việc thiết kế mạch trong thực tế là một vấn đề lớn. Vì việc thiết kế mạch sẽ ảnh hưởng đến độ bền của thiết bị, linh kiện, mức độ "chịu đựng" về điện áp hoặc các hiệu ứng nhiễu (trong các thiết bị âm thanh, thu phát sóng,…trong các môi trường khác nhau), cũng như năng lượng tiêu thụ và các yếu tố khác... Do đó, mặc dù việc sử dụng các giải pháp đại số Bool để làm đơn giản hóa mạch là một vấn đề khác. Đôi khi, với các thiết kế phức tạp thì mạch trở nên ổn định, bền và chống nhiễu khi đưa vào sử dụng.

2. Khái niệm về Máy trạng thái hữu hạn FSM (Finite State Machines)

Một "Finite State Machine" viết tắt là FSM còn được gọi là "máy trạng thái" hoặc "Finite State Automaton" là máy ảo gồm tập các trạng thái (bao gồm trạng thái khởi đầu và một hoặc nhiều trạng thái cuối), một tập các sự kiện nhập, 1 tập sự kiện xuất và 1 hàm chuyển đổi trạng thái. Hàm chuyển đổi trạng thái lấy state và sự kiện nhập là đầu vào và trả về tập các sự kiện xuất mới và trạng thái tiếp theo (trạng thái mới). Một số trạng thái được gọi là trạng thái "chấm dứt" (terminal states) khi không thể chuyển sang trạng thái khác.

Hoạt động của một FSM bắt đầu bằng một trạng thái gọi là trạng thái khởi đầu và xử lý xuyên suốt các dịch chuyển phụ thuộc vào các trạng thái khác nhau và thông thường kết thúc khi ở trạng thái cuối. Một trạng thái đánh dấu một dòng thành công của hoạt động được gọi là một trạng thái được chấp nhận (accept state). Kỹ thuật state machine hiện tại được sử dụng trong các ứng dụng: trò chơi (games), giao diện người sử dụng, giao thức mạng và các chương trình phân tích.

2.1. Mô hình toán học:

Một máy trạng thái hữu hạn xác định là một bộ 5: $(\Sigma, S, s_0, \delta, F)$ với:

- Σ là bảng chữ cái đầu vào (tập hữu hạn và không rỗng các kí hiệu).
- S là tập trạng thái hữu hạn và không rỗng.
- s_0 là trạng thái ban đầu, là 1 phần tử của S.
- δ là hàm chuyển đổi trạng thái, được mô tả như sau: $\delta: S \times \Sigma \to S$
  Trong máy trạng thái hữu hạn không xác định, hàm trên sẽ là $\delta: S \times \Sigma \to \wp(S)$, nghĩa là δ sẽ trả về 1 tập các trạng thái, trong đó $\wp(S)$ là power set (tập các tập con) của tập S.
- F là các trạng thái cuối (tập này có thể là tập rỗng), nghĩa là F là tập con của tập S.

Hệ máy trạng thái hữu hạn (FSM – Finite State Machines) có những ứng dụng như bộ điều khiển các bộ phận như động cơ, phanh… của tàu điện ngầm khi muốn dừng tàu khẩn cấp ở các vận tốc: 50km/h, 100km/h, 300km/h. Nhiều hệ FSM được cài đặt để đáp ứng nhu cầu sử dụng, đặc biệt trong các hệ thống nhúng, điều khiển, vi điều khiển để đảm bảo về an toàn và ổn định của một hệ thống.

2.2. Ví dụ: Mạch điện đèn điều khiển tín hiệu giao thông

Một mạch điện cho đèn xanh – đỏ giao thông sẽ gồm 3 trạng thái: s0 (xanh), s1 (vàng) và s2 (đỏ). Cứ mỗi giây, một tín hiệu điều khiển được đưa vào và trạng tái biến đổi được mô tả như sau:

Trạng thái hiện tại	Tín hiệu điều khiển vào	Trạng thái tương lai
S0	0	S0
S0	1	S1
S1	0	S1
S1	1	S2
S2	0	S2
S2	1	S0

Rõ ràng, với hệ thống mạch đèn giao thông là một minh họa cho state machine khi thỏa 2 điều kiện: có thể biết được trạng thái biến đổi khi biết dữ liệu và có lưu trạng thái hiện tại.

*Lưu ý: biến **global** trong Python có nghĩa là sử dụng biến toàn cục*

```
>>> trang_thai = "s0"

>>> def chuyen( tin_hieu):
        global trang_thai
        if trang_thai == "s0":
                if tin_hieu == 0:
                        print ("0 - s0")
                        trang_thai = "s0"
                elif tin_hieu == 1:
                        print ("1 - s1")
                        trang_thai = "s1"
        elif trang_thai == "s1":
                if tin_hieu == 0:
                        print ("0 - s1")
                        trang_thai = "s1"
                elif tin_hieu == 1:
                        print ("1 - s2")
                        trang_thai = "s2"
        elif trang_thai == "s2":
                if tin_hieu == 0:
                        print ("0 - s2")
                        trang_thai = "s2"
                elif tin_hieu == 1:
                        print ("1 - s0")
                        trang_thai = "s0"
```

Sau đó, sinh viên có thể thử nghiệm biến đổi trạng thái:

>>> tap_tinhieu = [1, 0, 1, 0, 1, 1, 0 , 0, 1] # thiết lập tập dữ liệu đầu vào

>>> for i in tap_tinhieu:

chuyen(i)

……

Bên cạnh việc xây dựng chương trình như trên, Python hỗ trợ chúng ta phương thức viết đơn giản và hiệu quả hơn. Chúng ta hãy xem xét đoạn mã dưới đây:

```
>>> trang_thai = "s0"

>>> trangthai_ke = { 's0': ['s0', 's1'],
                     's1': ['s1', 's2'],
                     's2': ['s2', 's0']}
```

```
>>> ket_qua = { 's0': [ 0, 1],
                's1': [ 0, 1],
                's2': [ 0, 1]}
```

```
>>> def chuyen_trangthai( tin_hieu):
        global trang_thai
        chuoi_in = str(ket_qua[trang_thai][tin_hieu])
        chuoi_in = chuoi_in + " - " + trangthai_ke[trang_thai][tin_hieu]
        print (chuoi_in)
        trang_thai = trangthai_ke[trang_thai][tin_hieu]
```

Sau đó, chúng ta có thể thử nghiệm:

>>> tap_tinhieu = [1, 0, 1, 0, 1, 1, 0 , 0, 1]

>>> for i in tap_tinhieu:

chuyen_trangthai(i)

…….. # so sánh kết quả với chương trình bên trên.

3. Xây dựng chương trình kiểm tra ngữ pháp đơn giản

Giả định chúng ta muốn nhận diện nghĩa của mỗi câu nhỏ với một bộ từ vựng và ngữ pháp tối giản theo quy tắc như sau:

- Những câu phải bắt đầu với từ “Python is” và tiếp theo sau là:
- Một tính từ, hoặc từ “not” (không) cùng với một tính từ.

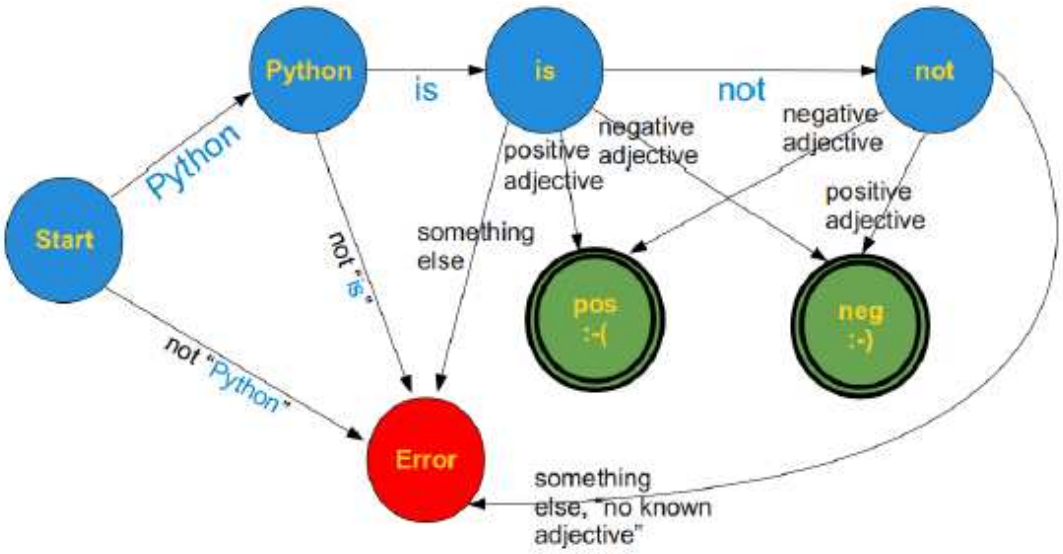
Ví dụ:

- “Python is great” (nghĩa tích cực) [Python vĩ đại]
- “Python is stupid” (nghĩa tiêu cực) [Python ngu ngốc]
- “Python is not ugly” (nghĩa tích cực) [Python không xấu xí]

Nếu không phải mẫu trên, nghĩa là câu đã viết sai!

Từ đó, chúng ta sẽ có mô hình xử lý như sau:

Trả về lỗi Sai (Error): khi bắt đầu câu không phải là chữ "Python" hoặc chữ kế tiếp không phải chữ "is"/"is not" hoặc chữ tiếp theo sau chữ "is"/"is not" không phải chữ "not" hoặc không phải một tính từ (adjective).



Chương trình trong Python

Để cài đặt ví dụ trên, chúng ta xây dựng chương trình **Finite State Machine** như sau:

```
class StateMachine:
    def __init__(self):
        self.tap_xuly = {}
        self.trangthaiBatdau = None
        self.trangthaiKetThuc = []

    def them_Trangthai(self, trangthai, xuly, trangthai_ketthuc = 0):
        trangthai = trangthai.upper()
        self.tap_xuly[trangthai] = xuly
        if trangthai_ketthuc:
            self.trangthaiKetThuc.append(trangthai)

    def thietlap_TrangthaiBatdau(self, trangthai):
        self.trangthaiBatdau = trangthai.upper()

    def thucthi(self, dauvao):
        try:
            xuly = self.tap_xuly[self.trangthaiBatdau]
        except:
            raise InitializationError("Phai gĩi .thietlap_TrangthaiBatdau() truoc khi goi .thucthi() ")

        if not self.trangthaiKetThuc:
            raise InitializationError("It nhat 1 trang thai phai la trang thai ket thuc")

        while True:
            (TrangThaiMoi, dauvao) = xuly(dauvao)
            if TrangThaiMoi.upper() in self.trangthaiKetThuc:
                print ("Den dich! ", TrangThaiMoi)
                break
            else:
                xuly = self.tap_xuly[TrangThaiMoi.upper()]
```

Thể hiện code:

class StateMachine:

def __init__(self):

self.tap_xuly = {}

self.trangthaiBatdau = None

self.trangthaiKetThuc = []

def them_Trangthai(self, trangthai, xuly, trangthai_ketthuc = 0):

trangthai = trangthai.upper()

self.tap_xuly[trangthai] = xuly

if trangthai_ketthuc:

self.trangthaiKetThuc.append(trangthai)

```
    def thietlap_TrangthaiBatdau(self, trangthai):
        self.trangthaiBatdau = trangthai.upper()

    def thucthi(self, dauvao):
        try:
            xuly = self.tap_xuly[self.trangthaiBatdau]
        except:
            raise InitializationError("Phai gĺi .thietlap_TrangthaiBatdau() truoc khi goi .thucthi() ")

        if not self.trangthaiKetThuc:
            raise InitializationError("It nhat 1 trang thai phai la trang thai ket thuc")

        while True:
            (TrangThaiMoi, dauvao) = xuly(dauvao)
            if TrangThaiMoi.upper() in self.trangthaiKetThuc:
                print ("Den dich! ", TrangThaiMoi)
                break
            else:
                xuly = self.tap_xuly[TrangThaiMoi.upper()]
```

Và chương trình sử dụng lớp trên FSM.py:

```
# Thuc thi chuong trinh nay su dung goi statemachine.py
```

- Đưa thư viện statemachine vào sử dụng

- Khởi tạo các danh sách "từ vựng"

```
# Thuc thi chuong trinh nay su dung goi statemachine.py


from statemachine import StateMachine

tinhtu_tichcuc = ["vi_dai", "sieu", "vui", "de", "giai_tri"]
tinhtu_tieucuc = ["chan", "kho", "xau", "kem"]
```

from statemachine import StateMachine

Khởi tạo các danh sách từ vựng:

tinhtu_tichcuc = ["vi_dai", "sieu", "vui", "de", "giai_tri"] # danh sách tính từ tích cực

tinhtu_tieucuc = ["chan", "kho", "xau", "kem"] # tính từ tiêu cực

- Khai báo trạng thái ban đầu:

```
def trangthai_baudau(txt):
    splitted_txt = txt.split(None, 1)
    tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")
    if tu == "Python":
        trangthaimoi = "Python_state"
    else:
        trangthaimoi = "error_state"
    return (trangthaimoi, txt)
```

def trangthai_baudau(txt):

splitted_txt = txt.split(None, 1)

tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

if tu == "Python":

trangthaimoi = "Python_state"

else:

trangthaimoi = "error_state"

return (trangthaimoi, txt)

Các đoạn code tương tự:

- Khai báo trạng thái python
- Và các trạng thái tiếp theo (is, is not, tính từ)

```
def trangthai_python(txt):
    splitted_txt = txt.split(None, 1)
    tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

    if tu == "is":
        trangthaimoi = "is_state"
    else:
        trangthaimoi = "error_state"
    return (trangthaimoi, txt)
```

def trangthai_python(txt):

splitted_txt = txt.split(None, 1)

tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

if tu == "is": # nếu từ là 'is' thì

trangthaimoi = "is_state" # trạng thái mới là is_state

else:

trangthaimoi = "error_state" # ngược lại trạng thái mới là lỗi (error_state)

return (trangthaimoi, txt)

```
def trangthai_is_chuyentrangthai(txt):
    splitted_txt = txt.split(None, 1)
    tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

    if tu == "not":
        trangthaimoi = "not_state"
    elif tu in tinhtu_tichcuc:
        trangthaimoi = "pos_state"
    elif tu in tinhtu_tieucuc:
        trangthaimoi = "neg_state"
    else:
        trangthaimoi = "error_state"
    return (trangthaimoi, txt)
```

def trangthai_is_chuyentrangthai(txt):

splitted_txt = txt.split(None, 1)

tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

if tu == "not":

trangthaimoi = "not_state"

elif tu in tinhtu_tichcuc:

trangthaimoi = "pos_state"

elif tu in tinhtu_tieucuc:

trangthaimoi = "neg_state"

else:

trangthaimoi = "error_state"

return (trangthaimoi, txt)

```
def trangthai_isnot_chuyentrangthai(txt):
    splitted_txt = txt.split(None, 1)
    tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

    if tu in tinhtu_tichcuc:
        trangthaimoi = "pos_state"
    elif tu in tinhtu_tieucuc:
        trangthaimoi = "neg_state"
    else:
        trangthaimoi = "error_state"
    return (trangthaimoi, txt)
```

def trangthai_isnot_chuyentrangthai(txt):

splitted_txt = txt.split(None, 1)

tu, txt = splitted_txt if len(splitted_txt) > 1 else (txt, "")

if tu in tinhtu_tichcuc:

trangthaimoi = "pos_state"

elif tu in tinhtu_tieucuc:

trangthaimoi = "neg_state"

else:

trangthaimoi = "error_state"

return (trangthaimoi, txt)

- Và khai báo trạng thái tiêu cực

```
def neg_state(txt):
    print ("Chao!")
    return ("neg_state","")
```

def neg_state(txt):

print ("Chao!")

return ("neg_state","")

- Và hàm main để gọi:

```
if __name__ == "__main__":
    m = StateMachine()
    # add_state
    m.them_Trangthai("Start", trangthai_baudau)
    m.them_Trangthai("Python_state", trangthai_python)
    m.them_Trangthai("is_state", trangthai_is_chuyentrangthai)
    m.them_Trangthai("not_state", trangthai_isnot_chuyentrangthai)
    m.them_Trangthai("neg_state", None, trangthai_ketthuc = 1)
    m.them_Trangthai("pos_state", None, trangthai_ketthuc = 1)
    m.them_Trangthai("error_state", None, trangthai_ketthuc = 1)
    # set_start
    m.thietlap_TrangthaiBatdau("Start")
    # run
    m.thucthi("Python is vi_dai")
    m.thucthi("Python is kho")
    m.thucthi("Python is xau")
    m.thucthi("Python is xao")
```

if __name__ == "__main__":

m = StateMachine()

```
# add_state
m.them_Trangthai("Start", trangthai_baudau)
m.them_Trangthai("Python_state", trangthai_python)
m.them_Trangthai("is_state", trangthai_is_chuyentrangthai)
m.them_Trangthai("not_state", trangthai_isnot_chuyentrangthai)
m.them_Trangthai("neg_state", None, trangthai_ketthuc = 1)
m.them_Trangthai("pos_state", None, trangthai_ketthuc = 1)
m.them_Trangthai("error_state", None, trangthai_ketthuc = 1)
# set_start
m.thietlap_TrangthaiBatdau("Start")
# run
m.thucthi("Python is vi_dai")
m.thucthi("Python is kho")
m.thucthi("Python is xau")
m.thucthi("Python is xao")
```

Kết quả thực thi là (khi đến đích nghĩa là câu nói đã đúng ngữ pháp!)

```
========== RESTART: C:/Anaconda3/Scripts/FSM.py =
Den dich!  pos_state
Den dich!  neg_state
Den dich!  neg_state
Den dich!  error_state
>>>
```

BÀI TẬP CHƯƠNG 6

Câu 1: Thiết kế thang máy! Hệ máy trạng thái hữu hạn trong Python

Trong điều khiển thang máy, vị trí tại mỗi tầng sẽ là một trạng thái của thang máy. Sau đó, thang máy sẽ nhận được các lệnh đi lên hoặc xuống tùy theo người sử dụng và hệ thống điều khiển thang máy sẽ thực hiện các tác vụ theo trạng thái. Cụ thể hơn, chúng ta có bài toán như sau:

Nhà ga metro gồm 2 tầng (tầng Trệt và tầng 1). Nhà ga có 1 thang máy dành cho người hạn chế lên xuống cầu thang (người già, người khuyết tật, người bị thương ở chân) sử dụng với 1 nút bấm điều khiển để đi giữa hai tầng. Đèn tín hiệu của thang là: màu Tím khi thang ở tầng Trệt và màu Hồng khi ở tầng 1.

Tại mỗi thời điểm, bộ điều khiển sẽ kiểm tra trạng thái của thang để đi lên hoặc xuống và thay đổi tín hiệu đèn.

Bạn được giao nhiệm vụ lập trình một đoạn chương trình cho con chip nhúng để điều khiển thang máy có 1 nút nhấn mỗi tầng. Hãy viết chương trình bằng Python để điều khiển của thang.

Thực hiện:

- Vẽ lược đồ FSM cho hệ thống.

Lưu ý: Lược đồ là hình ảnh tổng quan của hệ thống. Các trạng thái của hệ thống được thể hiện bằng những đường tròn. Các mũi tên chỉ hướng thay đổi trạng thái của hệ thống.

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Hoàng Thị Kiều Anh – Lê Ngọc Thành – Phạm Trọng Nghĩa –Nguyễn Công Nhựt – Trần Ngọc Việt – Đỗ Đình Thủ – Nguyễn Hữu Trí Nhật – Lê Công Hiếu – Nguyễn Thị Thanh Bình – Nguyễn Thái Hải – Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 7: ĐỒ THỊ VÀ CÁC TÍNH CHẤT CỦA ĐỒ THỊ

Mục tiêu:

- *Khái niệm về biểu diễn đồ thị trong Python*
- *Định nghĩa về sự liên thông của đồ thị.*
- *Xử lý về đường đi, chu trình của đồ thị bằng Python.*
- *Khai phá các tính chất của đồ thị thông qua gói networkx của Python*

Nội dung chính:

1. Biểu diễn đồ thị trong Python

Hiện tại, một đồ thị được biểu diễn bằng nhiều dạng trong các ngôn ngữ lập trình. Dưới đây, hai dạng cơ bản trong ngôn ngữ Python được giới thiệu.

1.1. Biểu diễn đồ thị bằng ma trận kề

Ma trận kề là hình thức biểu diễn đồ thị đơn giản nhất. Ma trận kề là ma trận vuông nxn với n là số đỉnh của đồ thị. Các quy tắc biểu diễn một ma trận kề $A = \{a_{ij}\}, 1 \leq i \leq n, 1 \leq j \leq n$

- Ma trận gồm n dòng và n cột tương ứng với đồ thị n đỉnh.
- Giá trị của a_{ij} thể hiện một hoặc nhiều dạng thông tin dưới đây:
- Sự kết nối (đối với đồ thị chỉ quan tâm đến kết nối): giá trị khác 0 để cho biết có sự kết nối từ đỉnh i đến đỉnh j.
- Chiều dài/giá trị/hàm phạt/khả năng tiếp cận từ đỉnh i đến đỉnh j (giá trị vô cùng xem như hai đỉnh không có sự kết nối trực tiếp; giá trị 0 cho thấy hai đỉnh trùng nhau; giá trị âm cho thấy đây là kết nối ngược chiều);
- Dạng đồ thị: đồ thị có hướng thường các cạnh ngược hướng sẽ có giá trị âm (nghĩa là: $a_{ij} = -a_{ji}$).

Tuy nhiên, nhược điểm của phương pháp thể hiện này là: việc thể hiện đồ thị lớn và đặc biệt là thưa sẽ tốn rất nhiều tài nguyên. Do ma trận tăng mũ 2 theo kích thước của số đỉnh.

1.2. [Đọc thêm] Sử dụng cấu trúc dữ liệu của Python

Sinh viên tham khảo thêm và thực hành trực tuyến tại đây với sự hỗ trợ của giảng viên:

https://www.python-course.eu/graphs_python.php

2. Một số đặc trưng và tính chất của đồ thị

Dưới đây là một số chỉ số về đặc trưng cũng như các tính chất của một đồ thị/mạng.

2.1. Các chỉ số và tính chất cơ bản về đỉnh và cạnh, loại đồ thị

Để đánh giá mức độ phức tạp và sự kết nối trong cục bộ của một đồ thị/mạng:

- Chỉ số χ: là tỉ số giữa số liên kết thực sự và số liên kết có thể có trong mạng. Công thức:

$$\chi = \frac{l}{l_{max}} = \frac{l}{3(n-2)}, n: số\ đỉnh\ hoặc\ nút\ của\ mạng$$

Nhận xét:

- (ngưỡng ít liên kết) $0 \leq \chi \leq 1$ (ngưỡng nhiều liên kết)

- Chỉ số α: là tỉ số giữa số lượng vòng thực sự và số lượng vòng cực đại trong mạng (Vòng liên kết giữa 3 đỉnh, không tính vòng bao trùm).

$$\alpha = \frac{c}{c_{max}} = \frac{c}{(2n-5)}$$

Nhận xét: với mạng ít liên kết, chỉ số α không thích hợp để đánh giá. Với mạng có chỉ số α lớn (α gần bằng 1) thì mạng có sự kết nối tốt. Ứng dụng trong việc phân tích kết nối giao thông. Tính toán xem nơi nào "giao thông thuận tiện"

2.2. Các thuộc tính của đồ thị

Cho đồ thị $G = (V, E)$ với V là tập các n đỉnh và E là tập các cạnh.

Gọi khoảng cách giữa hai đỉnh u, v $(u, v \in E)$ trong đồ thị G là $d(u, v)$.

- **Phủ*** của một đỉnh (eccentricity)

Phủ của một đỉnh kí hiệu là $ecc(v)$ là khoảng cách xa nhất đến các đỉnh khác. Như vậy, theo định nghĩa, đó là:

$$ecc(v) = \max_{x \in V}\{d(v, x)\}, (v \in V)$$

(là từ tạm dịch)*

- Đường kính của đồ thị (diameter), kí hiệu là diam(G):

Đường kính của đồ thị được định nghĩa là độ dài **lớn nhất** trong các đường đi ngắn nhất giữa hai điểm bất kì trong G. Như vậy, theo định nghĩa, đường kính của đồ thị G là $Diam(G)$:

$$Diam(G) = \max\{ecc(v)|\}$$

Ứng dụng: Xét thời gian chậm nhất để loan 1 tin. Ví dụ: sáng 8 giờ 00 hằng ngày, tất cả các ngân hàng, cửa hàng đều được cập nhật giá ngoại tệ, giá cổ phiếu. Do đó, trước đó, một đoạn mã script webservice được cài đặt để chuyển đến các máy. Máy chậm nhất trong hệ thống cũng phải được chuyển đến trước thời điểm 8 giờ 00.

- Bán kính của đồ thị (radius), kí hiệu là radius(G):

Ngược với đường kính của đồ thị, bán kính của đồ thị được định nghĩa là giá trị **nhỏ nhất** của phủ của đồ thị. Như vậy, bán kính của đồ thị được định nghĩa là:

$$radius(G) = \min\{ecc(v)|\}$$

Ứng dụng: Trong một thành phố, nếu các khu trung tâm được xem là một điểm của đồ thị (có các cạnh là các đường kết nối) thì việc tìm bán kính đồ thị là tìm vị trí để đặt các tiện ích xã hội như: trạm y tế, trạm chữa cháy. Các đỉnh có ecc bằng với bán kính có thể được xem là các đỉnh trung tâm của đồ thị.

3. Sử dụng gói networkx để giải các bài toán đồ thị

3.1. Giới thiệu gói networkx

Thư viện **networkx** là một thư viện để xử lý mạng/đồ thị trong Python. Networkx có các khả năng:

- Phân tích mạng cơ bản:
- Thể hiện các khái niệm cơ bản của đồ thị.
- Thuộc tính về loại và cấu trúc của đồ thị.
- Nhận diện các node trung tâm của đồ thị.
- Truyền thông trong mạng:
- Nhóm và phân hoạch đồ thị.
- Tìm kiếm các "cộng đồng" trong mạng/đồ thị tĩnh và động.
- Các ứng dụng về phân tích mạng

Hiện tại, gói networkx được tích hợp trong gói phần mềm Anaconda. Các thành phần căn bản của gói networkx bao gồm:

- Đồ thị: có thể là đồ thị vô hướng (Graph) hoặc hữu (có) hướng (DiGraph).
- Node: các nốt của đồ thị.
- Edge: các cạnh của đồ thị

Để sử dụng gói networkx, chúng ta phải tham chiếu thư viện

>>> import networkx as nx # sử dụng với tên gọi là nx hoặc **>>> import networkx**

3.2. Tạo lập đồ thị vô hướng với networkx

```
>>> g = nx.Graph()
>>> g.add_node('TP.HCM')
>>> g.add_node('Dong Nai')
>>> g.add_node('Ba Ria Vung Tau')
>>> g.add_node('Lam Dong')
>>> g.add_node('Can Tho')
>>> g.add_node('Long An')
>>> g.add_edge('TP.HCM', 'Dong Nai')
>>> g.add_edge('TP.HCM', 'Ba Ria Vung Tau')
>>> g.add_edge('TP.HCM', 'Long An')
>>> g.add_edge('Dong Nai', 'Lam Dong')
>>> g.add_edge('Dong Nai', 'Ba Ria Vung Tau')
>>> print (g.number_of_nodes())
…… ← sinh viên điền vào
>>> print (g.number_of_edges())
…… ← sinh viên điền vào
>>> print (g.nodes())
…… ← sinh viên điền vào
>>> print (g.edges())
…… ← sinh viên điền vào
……
>>> print (g.degree('TP.HCM'))
…… ← sinh viên điền vào
>>> print (g.degree())
```

…… ← sinh viên điền vào

>>> print (list(g.neighbors('TP.HCM')))

…… ← sinh viên điền vào

>>> g.**has_edge**('Lam Dong', 'Long An')

…… ← sinh viên điền vào

>>> nx.**shortest_path**(g, 'Lam Dong', 'Long An') # đã có mạng lưới đường đi

…… ← sinh viên điền vào

>>> nx.**shortest_path**(g, 'Lam Dong', 'Can Tho') # hiện tại chưa xây dựng mạng đường đi

…… ← sinh viên điền tên **Exception**

>>> g.add_node('Tien Giang')

>>> g.add_edge('Tien Giang', 'Long An')

>>> g.add_edge('Tien Giang', 'Can Tho')

>>> nx.**shortest_path**(g, 'Lam Dong', 'Can Tho') # hiện tại đã bổ sung thêm đường đi

…… ← sinh viên điền vào

>>> nx.**shortest_path_length**(g, 'Lam Dong', 'Ba Ria Vung Tau')

…… ← sinh viên điền vào

>>> nx.**shortest_path_length**(g, 'Dong Nai', 'Ba Ria Vung Tau')

…… ← sinh viên điền vào

>>> nx.**shortest_path_length**(g, 'Lam Dong', 'Long An')

…… ← sinh viên điền vào

BÀI TẬP CHƯƠNG 7

Câu 1: Cài đặt các hàm tính toán cấu trúc mạng:

- Chỉ số χ của một đồ thị/mạng.
- Chỉ số α của một đồ thị/mạng.

Câu 2: Cài đặt thuật toán Dijsktra theo cấu trúc dữ liệu ma trận kề.

Cho một đồ thị có trọng số $G = (X, E)$ với X, E lần lượt là tập đỉnh và tập cạnh. Xét ánh xạ:

$$W: E \to \mathbb{R}$$

$$e \mapsto w(e)$$

Ta có các định nghĩa sau:

(i). $w(e)$ là trọng số/trọng lượng của cạnh/cung e

(ii). Cho 2 đỉnh $a, b \in X$. Đặt $\wp = \{L | L \text{ đường đi từ } a \text{ đến } b\}$.

Giả sử tồn tại $L_0 = \min\{w(L): L \in \wp\}$. Khi đó, L_0 gọi là đường đi ngắn nhất từ $a \to b$.

Với bài toán tìm đường đi ngắn nhất giữa hai đỉnh, yêu cầu của đồ thị là đồ thị không có mạch âm (vì nếu có mạch âm thì càng đi con đường càng ngắn do trọng số/trọng lượng là số âm).

Nhắc lại thuật toán Dijsktra:

Dijkstra là thuật toán tìm đường đi ngắn nhất từ **đỉnh đầu tiên (0) đến một đỉnh khác** trong đồ thị với trọng số mọi đỉnh đều dương, nghĩa là: $w(u) \geq 0, \forall u$.

Thuật toán:

Đặt:

$$L = (L_{ij}) \text{ với } L_{ij} = \begin{cases} w(u), & \text{nếu } u = u_{ij} \in U \\ \infty, & \text{nếu } u = u_{ij} \notin U \\ 0, & \text{nếu } i = j \end{cases}$$

Gọi $\pi(i)$ là độ dài đường đi ngắn nhất từ một đỉnh khác đến đỉnh $i, i \in X = \{0,2, \dots, n-1\}$; lưu ý: tập đỉnh bắt đầu tính từ 0 cho phù hợp với ngôn ngữ Python.

$\pi^*(i)$ là giá trị tạm (tính) của $\pi(i)$.

Thuật toán thực hiện các bước sau:

Bước 1: $\pi^*(0) = 0$; $\pi^*(i) = \infty, \forall i \neq 0$; S=$X$. Tập S là tập các đỉnh cần được tính toán. Nếu đỉnh nào được loại nơi đây, nghĩa là độ dài đường đi từ đỉnh đó đến đỉnh 0 đã xác định.

Bước 2: Chọn $j \in S$ thỏa: $\pi^*(j) = \min \pi^*(i), i \in S$. Đặt, nghĩa là cập nhật độ dài đường đi từ đỉnh 0 đến đỉnh j và loại bỏ j ra khỏi tập S:

$$\pi(j) \leftarrow \pi^*(j)$$

$$S \leftarrow S - \{j\}$$

Nếu $S = \emptyset$: dừng thuật toán.

Bước 3: Với mọi $i \in \{lân\ cận\ j\} \cap S$, đặt $P = \pi(j) + L_{ji}$. Khi đó, đặt: $\pi^*(i) = \min\{\pi^*(i), P\}$. Nếu $\pi^*(i) = P$ thì đánh dấu đỉnh $i(\pi^*(i), j)$

Sau đó trở lại bước 2.

Ví dụ: Cho đồ thị sau:

Đỉnh\Đỉnh	**0**	**1**	**2**	**3**	**4**	**5**
0	0	7	1	∞	∞	∞
1	7	0	4	4	2	∞
2	1	4	0	∞	6	∞
3	∞	4	∞	0	2	5
4	∞	2	6	2	0	3
5	∞	∞	∞	5	3	0

Thực hiện các bước của thuật toán, ta được như sau:

Bước 1: $\pi^*(0) = 0; \pi^*(i) = \infty, \forall i \neq 0, S = \{0,1,2,3,4,5\}$

$\pi^*(i)$	0	1	2	3	4	5
Độ dài	**0**	∞	∞	∞	∞	∞

Bước 2: $j = 0;\ \pi^*(0) = 0; S \leftarrow \{0,1,2,3,4,5\}\backslash\{0\} = \{1,2,3,4,5\}$

Bước 3: $i \in \{lân\ cận\ 0\} \cap S = \{1,2\} \cap \{1,2,3,4,5\} = \{1,2\}$. Thực hiện gán lại:

$$\pi^*(1) \leftarrow \min\{\pi^*(1), \pi(0) + L_{01}\} = \min\{\infty, 7\} = 7$$

Tương tự:

$$\pi^*(2) \leftarrow \min\{\pi^*(2), \pi(0) + L_{02}\} = \min\{\infty, 1\} = 1$$

$\pi^*(i)$	0	1	2	3	4	5
Độ dài	0	∞	∞	∞	∞	∞
	0	(đỉnh**0**, độ dài**7**)	(**0**, **1**)	∞	∞	∞

Bước 2: $j = 2;\ \pi^*(2) = 1; S \leftarrow \{1,2,3,4,5\}\backslash\{2\} = \{1,3,4,5\}$

Bước 3: $i \in \{lân\ cận\ 2\} \cap S = \{1,4,5\} \cap \{1,3,4,5\} = \{1,4,5\}$. Thực hiện gán lại:

$$\pi^*(1) \leftarrow \min\{\pi^*(1), \pi(0) + L_{21}\} = \min\{7, 1+4\} = 5$$

$$\pi^*(4) \leftarrow \min\{\pi^*(4), \pi(0) + L_{24}\} = \min\{\infty, 1+6\} = 7$$

$$\pi^*(5) \leftarrow \min\{\pi^*(5), \pi(0) + L_{25}\} = \min\{\infty, 1+2\} = 3$$

$\pi^*(i)$	0	1	2	3	4	5
Độ dài	0	(0,7)	(0,1)	∞	∞	∞
	0	(2,5)	(0,1)	∞	(2,7)	(2,3)

…

Sinh viên tự thực hiện các bước sau và lặp lại đến khi thuật toán dừng:

Bước 2:…

Bước 3:…

Bước 2:…

Bước 3:…

THỰC HÀNH TOÁN RỜI RẠC

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Hoàng Thị Kiều Anh – Lê Ngọc Thành – Phạm Trọng Nghĩa –Nguyễn Công Nhựt – Trần Ngọc Việt – Đỗ Đình Thủ – Nguyễn Hữu Trí Nhật – Lê Công Hiếu – Nguyễn Thị Thanh Bình – Nguyễn Thái Hải – Huỳnh Thái Học và các Giảng viên khác

TP.HCM – Năm 2020

MỤC LỤC

CHƯƠNG 8: ĐỒ THỊ DẠNG CÂY

Mục tiêu:

- *Tìm hiểu về đồ thị cây: định nghĩa, tính chất, các loại cây, các thuộc tính của cây.*
- *Các thuật toán xử lý cây: duyệt cây, cây khung và cây khung tối thiểu.*
- *Giới thiệu ứng dụng cây trong thực tiễn xử lý bằng Python.*
- *Các thao tác lệnh bổ sung với gói NetworkX.*

Nội dung chính:

1. Đồ thị cây (Tree)

Bài này giới thiệu về một loại đồ thị đặc biệt, đó là cây. Cây là một dạng đồ thị đặc biệt nên nhìn chung cây sẽ áp dụng được tất cả các thuật toán xử lý của đồ thị như tìm đường đi ngắn nhất,… Ngoài ra, cây có riêng những tính chất và các bài toán riêng.

1.1. Định nghĩa, tính chất

- Cây (tree): là một đồ thị liên thông và không có chu trình.
- Rừng (forest): một rừng có p cây. Mỗi cây là một đồ thị liên thông, do đó, rừng là đồ thị có p thành phần liên thông. Mỗi thành phần liên thông là 1 cây.
- Cây có hướng là một đồ thị có hướng. Trong cây có hướng, một đỉnh được gọi là rễ (root) nếu từ đó có thể có đường đi đến đến các đỉnh còn lại.

1.2. Định lý cơ bản về cây

Những điều sau đây là tương đương:

i. G là cây.
ii. Giữa 2 cặp đỉnh bất kỳ có 1 dây chuyền duy nhất nối chúng với nhau.
iii. G liên thông tối tiểu, nghĩa là nếu xóa đi 1 cạnh của G thì không còn liên thông nữa.
iv. Thêm một cạnh vào giữa 2 đỉnh không kề nhau thì ta sẽ có một chu trình sơ cấp duy nhất.
v. G liên thông và có n-1 cạnh.
vi. G không có chu trình và có n-1 cạnh.

1.3. Cây khung và cây khung tối thiểu

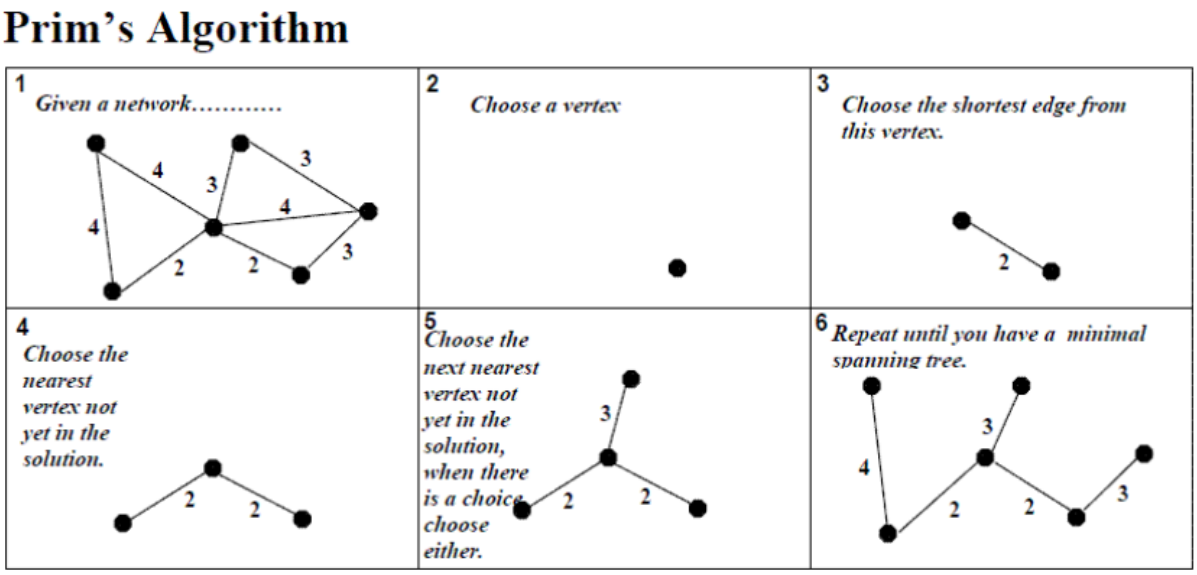
Cây khung hay còn gọi là cây tối đại (cây bao trùm/chùm): Cho một đồ thị $G = (V, E)$, một đồ thị cây $K = (V, U)$ được gọi là cây khung của G nếu K là đồ thị con của đồ thị G: có mọi đỉnh của đồ thị G và $U \subset E$.

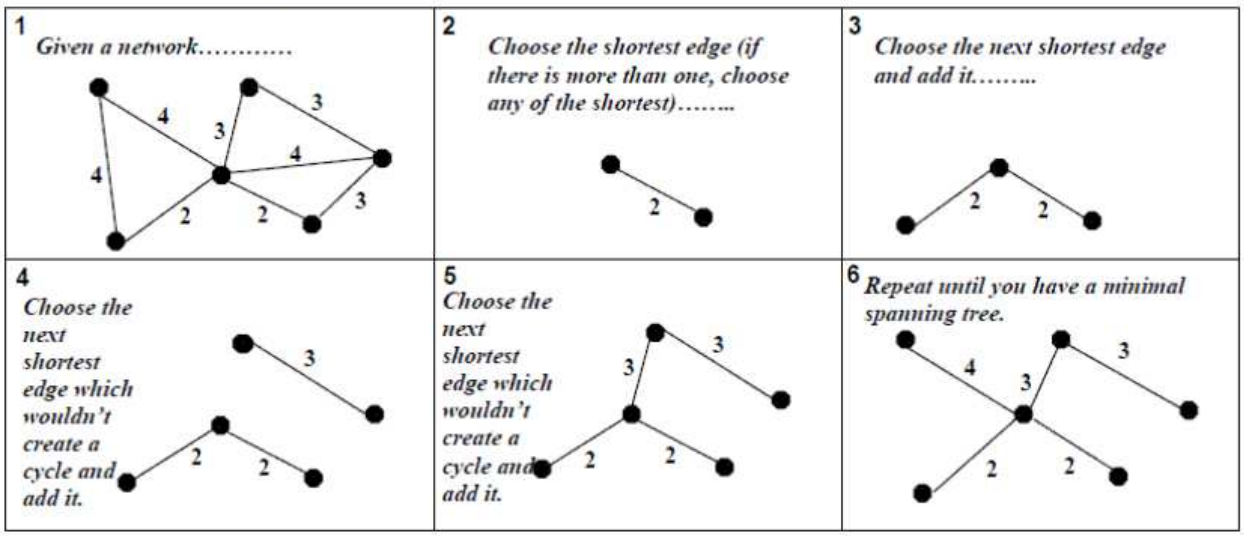
Cây khung nhỏ nhất: Xét G có trọng số cạnh, khi đó, nếu tổng các cạnh của cây K là nhỏ nhất thì đó là cây khung của đồ thị G. Cây khung nhỏ nhất được minh họa với các ứng dụng như: xây dựng mạng lưới ống nước/dây điện ngắn nhất tại các thành phố hoặc khu vực dân cư.

Từ một đồ thị G, hiện nay có nhiều thuật toán để xác định cây khung nhỏ nhất như: Prim, Kruskal, Boruvka. Trong đó, phổ biến là 2 thuật toán Prim và Kruskal như sau:

- Prim: tiếp cận chiều sâu (depth search) với ý tưởng bước đầu tiên chọn **điểm** vì và cạnh ngắn nhất từ đỉnh đó để "loang" rộng ra các đỉnh còn lại chưa được xét của đồ thị cùng với cạnh ngắn nhất mà không lặp thành vòng.
- Kruskal: tiếp cận chiều rộng (width search) với ý tưởng bước đầu tiên chọn **cạnh** ngắn nhất của đồ thị trước vì nhận định: cạnh ngắn nhất của đồ thị luôn nằm trong

Hình minh họa 2 thuật toán: [Giảng viên có thể giải thích thêm]





Gói networkx hỗ trợ việc tính cây khung/cây cực đại tối thiểu như sau:

Sinh viên có thể tham khảo tại đây:

https://networkx.github.io/documentation/latest/reference/algorithms/generated/networkx.algorithms.tree.mst.maximum_spanning_tree.html

Cụ thể xét đồ thị các tỉnh thành phố như sau:

```
>>> import networkx as nx
>>> g = nx.Graph()
>>> g.add_node('TP.HCM')
>>> g.add_node('Dong Nai')
>>> g.add_node('Ba Ria Vung Tau')
>>> g.add_node('Lam Dong')
>>> g.add_node('Can Tho')
>>> g.add_node('Long An')
>>> g.add_node('Tien Giang')
>>> g.add_edge('TP.HCM', 'Dong Nai', weight = 50)
>>> g.add_edge('TP.HCM', 'Ba Ria Vung Tau', weight = 120)
>>> g.add_edge('TP.HCM', 'Long An', weight = 40)
>>> g.add_edge('Dong Nai', 'Lam Dong', weight = 230)
>>> g.add_edge('Dong Nai', 'Ba Ria Vung Tau', weight = 60)
>>> g.add_edge('Tien Giang', '29')     # lệnh gõ nhầm
>>> g.remove_edge('Tien Giang', '29')    # xóa lệnh gõ nhầm
>>> g.add_edge('Tien Giang', 'Long An')    #lệnh gõ thiếu chiều dài (trọng số, weight)
>>> g.remove_edge('Tien Giang', 'Long An')   # xóa lệnh gõ thiếu chiều dài
>>> g.add_edge('Tien Giang', 'Long An', weight = 29)
>>> g.add_edge('Tien Giang', 'Can Tho', weight = 200)
>>> g.add_edge('Long An', 'Dong Nai', weight = 70)
```

>>> g.remove_edge('Tien Giang', '29') # lệnh sẽ báo lỗi vì cạnh này đã được xóa trước đó.

…… # sinh viên ghi nhận exception

Để xem đồ thị, chúng ta có thể xem: các đỉnh:

>>> g.nodes()

……

Tuy nhiên, chúng ta vẫn phải xóa đỉnh (node) '29' do lệnh tạo ra:

>>> g.remove_node('29')

>>> **g.nodes()** # đã xóa đỉnh '29'

……

Thể hiện dữ liệu các kết nối và có sắp xếp các cạnh nối theo tên cạnh của đồ thị g ban đầu:

>>> sorted(**g**.edges(data=True))

……

……

……

Giả sử, cần xây dựng đường truyền Internet với số lượng dây là ngắn nhất giữa các thành phố bên trên, chúng ta có thể xem xét xây dựng cây khung tối thiểu như sau:

>>> T = nx.maximum_spanning_tree(g)

Thể hiện dữ liệu và các kết nối của cây tối đại

>>> sorted(T.edges(data=True)) # tương tự thử nghiệm với lệnh >>> T.nodes()

……

……

……

Giảng viên cùng sinh viên vẽ đồ thị g ban đầu và đồ thị cây khung T được tạo thành.

Tài liệu tham khảo: Sinh viên có thể tham khảo thêm tại:

https://networkx.github.io/documentation/latest/reference/algorithms/generated/networkx.algorithms.tree.mst.maximum_spanning_tree.html

2. Một số tham khảo về hỗ trợ của gói Networkx để xử lý mạng đồ thị và cây:

Đứng ở góc độ các một chuyên gia về khoa học dữ liệu, bên cạnh việc tìm hiểu yêu cầu bài toán và thuật toán xử lý, khai thác công cụ phần mềm là sự cần thiết và yêu cầu như một kỹ năng. Theo đó, gói networkx là một thư viện với nhiều cài đặt để xử lý các bài toán mà sinh viên cần nắm rõ sử dụng. Dưới đây là liệt kê một số bài toán về đồ thị và cây cơ bản được xử lý bằng gói networkx:

- **Phân tích Pagerank (chỉ số kết nối):**

Giả định sinh viên đã tìm hiểu về phân tích pagerank (trong Thực hành đại số tuyến tính về ứng dụng trị riêng/vector riêng). Sinh viên có thể sử dụng hàm trong gói thư viện networkx để phân tích với giả định các liên kết trên là các liên kết để "xếp hạng". Lưu ý: đồ thị được xét bên trên là đồ thị vô hướng (xem như các liên kết là 2 chiều).

>>> nx.pagerank(g, 0.85)

{'TP.HCM': 0.13445880738149718, 'Dong Nai': 0.2351507400853221, 'Ba Ria Vung Tau': 0.11598739208998513, 'Lam Dong': 0.13355497428234894, 'Can Tho': 0.13462563996889287, 'Long An': 0.09373705822226845, 'Tien Giang': 0.15248538796968533}

Tham khảo: https://networkx.github.io/documentation/latest/reference/algorithms/link_analysis.html

Và các bài toán khác như: [giảng viên cung cấp thông tin thêm]

- Đồ thị hai hướng (bipartite) – giải các bài toán về "ghép đôi"

Tham khảo: https://networkx.github.io/documentation/latest/reference/algorithms/bipartite.html

- Bài toán tìm phủ ngắn nhất (covering) – ý tưởng như bài toán tập hợp phủ (set covering)

Lệnh sau để phân các vùng gần nhau:

>>> nx.min_edge_cover(g)

{('Can Tho', 'Tien Giang'), ('Ba Ria Vung Tau', 'TP.HCM'), ('Long An', 'TP.HCM'), ('TP.HCM', 'Long An'), ('Dong Nai', 'Lam Dong')}

Tham khảo: https://networkx.github.io/documentation/latest/reference/algorithms/covering.html

- Các bài toán về đường đi/chu trình (tournament)

Bài toán đường đi Hamilton:

Tham khảo: https://networkx.github.io/documentation/latest/reference/algorithms/tournament.html

…

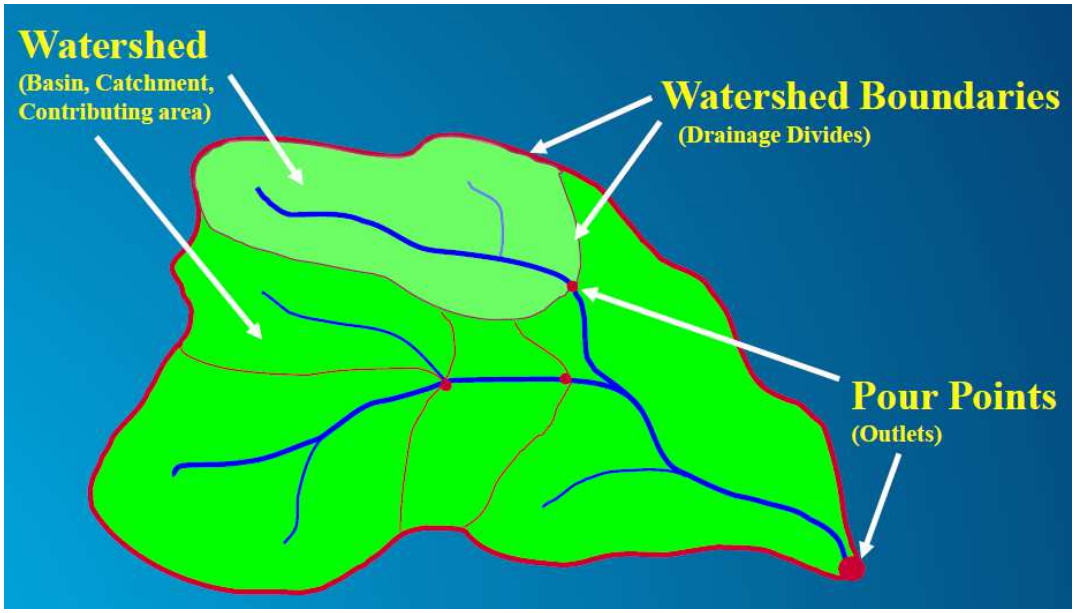
3. Bài toán ứng dụng 2: Bài toán tích lũy dòng chảy – Câu chuyện ngập khi mưa tại đô thị

Dưới đây là một ứng dụng của cây đồ thị trong việc tính toán tích lũy dòng chảy.

3.1. Giới thiệu mô hình tích lũy dòng chảy đơn dòng (single flow), thuật toán D8

Các khái niệm cơ bản:

- Hệ thống thoát nước bề mặt:



Bao gồm:

- Các lưu vực (watershed): là mỗi vùng nước chảy độc lập. TP. Hồ Chí Minh phân thành nhiều lưu vực lớn khác nhau như: Tân Hóa – Lò Gốm, Bắc Nhiêu Lộc, Nam Nhiêu Lộc,… Và mỗi lưu vực lớn đó bao gồm nhiều lưu vực nhỏ hơn.
- Các biên giới của lưu vực (watershed boudaries).
- Những điểm tiếp nhận nước (pour points).

Giả định quy luật của dòng chảy là: dòng chảy sẽ chảy từ nơi cao sang nơi thấp nhất và không có trường hợp chảy ngược lại từ nơi thấp sang nơi cao.

- Quy trình tính toán:

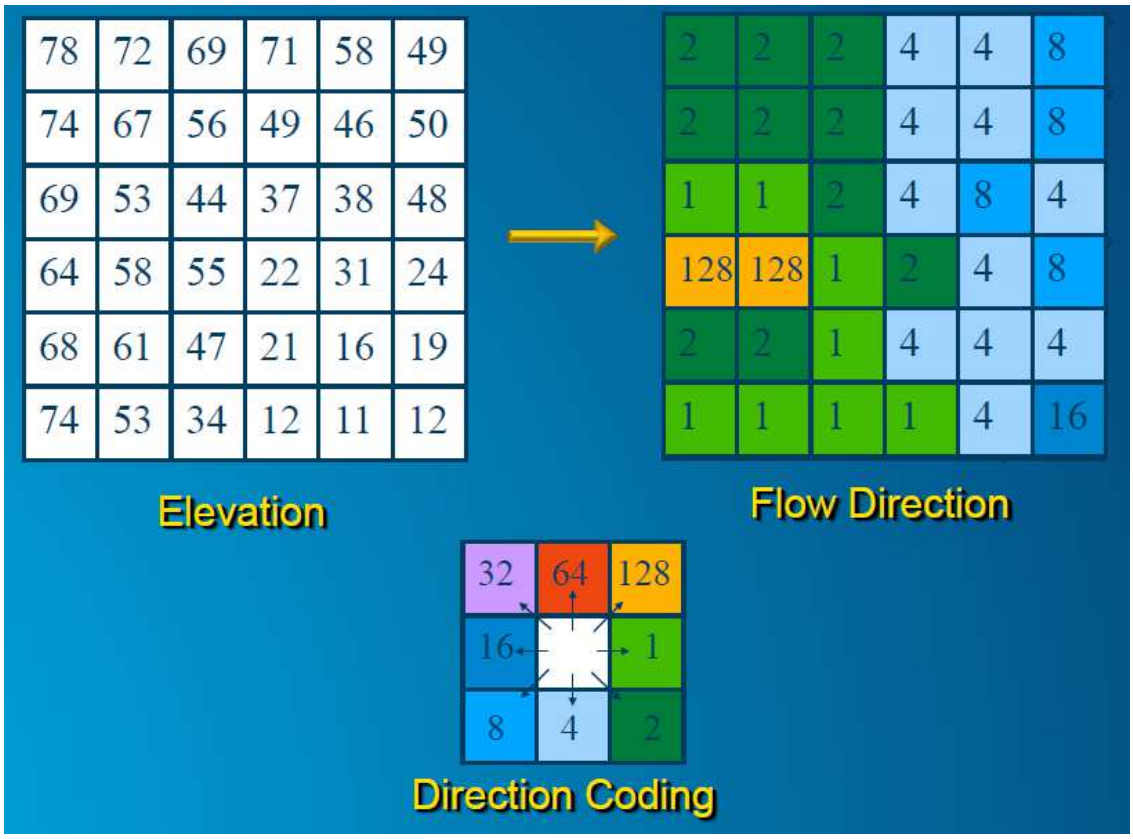
Để tính toán dòng chảy nước, 2 bước CHÍNH được xử lý:

- Bước 1: Hướng dòng chảy (FLOW DIRECTION): là tính toán hướng di chuyển của dòng (nước) tại mỗi điểm (vị trí địa lý).
- Bước 2: Tích lũy dòng chảy (FLOW ACCUMULATION): là tính giá trị tích lũy tại mỗi điểm.

Lưu ý: Các bước xử lý chuyên ngành sẽ được mô tả trong các bài giảng khác.

- Tính toán hướng dòng chảy và tích lũy dòng chảy (với mô hình đơn dòng):

Có nhiều thuật toán tính toán hướng dòng chảy. Ở đây, ta xét mô hình D8, một mô hình về dòng chảy đơn dòng, nghĩa là tại 1 vị trí chỉ có thể chảy đến 1 trong 8 vị trí lân cận có độ dốc thấp nhất. Ví dụ:



Lưu ý:

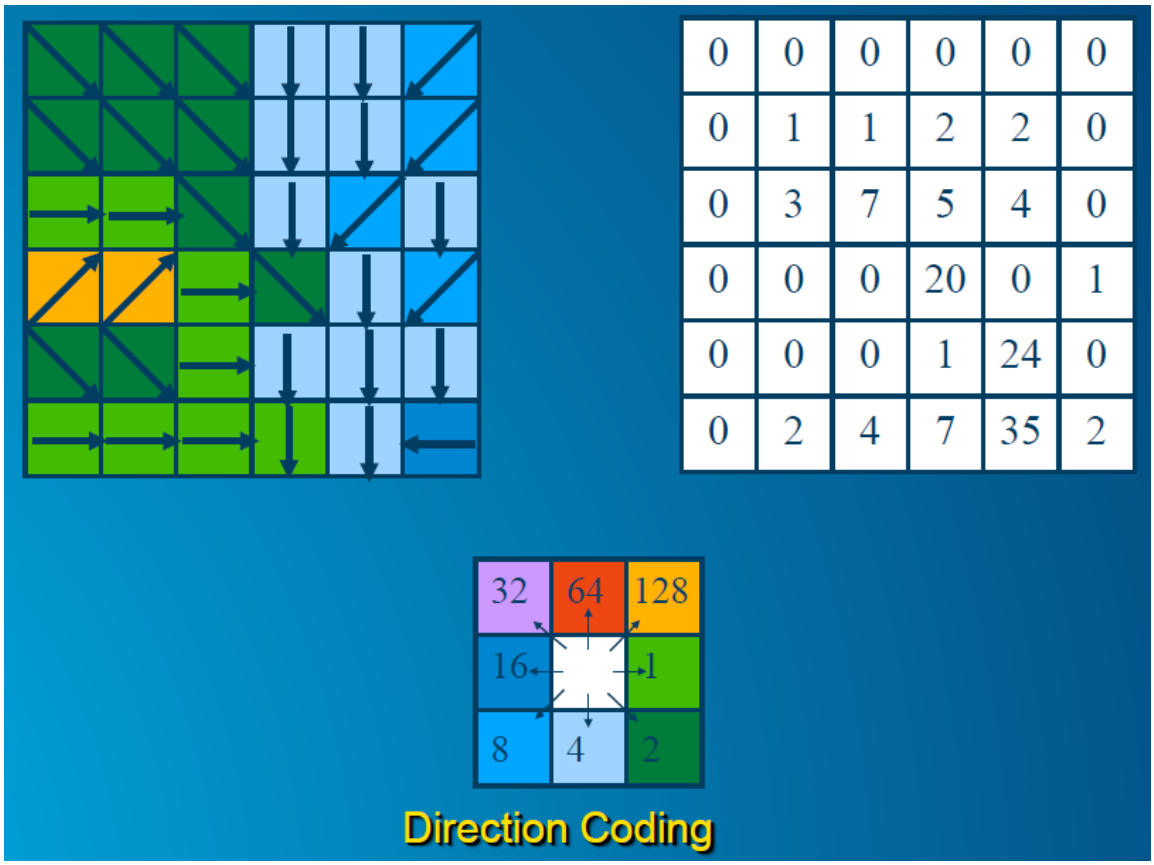
- Elevation hoặc DEM: là độ cao của địa hình thực tế.
- Flow Direction là hướng dòng chảy được mã hóa theo các mã hướng (direction coding). Tại 1 vị trí sẽ có 8 vị trí ứng cử viên để dòng chảy đến.

Cụ thể tính toán bảng trên: Từ bảng độ cao (elevation) hay còn gọi là mô hình độ cao số (DEM – digital elevation model), ta sẽ tính toán ra bảng Flow Direction với quy luật: **Tìm hướng có độ dốc cao nhất**. Ví dụ: ô cuối cùng mang giá trị 12 có 3 ô xung quanh là 16, 19 và 11. Như vậy, ***độ dốc cao nhất trong trường hợp này là ô 11***. Vì chúng ta có các tính toán như sau:

Xét ô 11: độ dốc từ ô 12 đổ về là: $\frac{12-11}{k} = 1$, với k là khoảng cách giữa 2 ô (thẳng hàng nhau).

Xét ô 16 và 19: chắc chắn không có dòng ngược từ độ cao 12 sang 16 hay ô 19. Để thuyết phục, ta thực hiện tính toán: $\frac{12-16}{k\sqrt{2}} < 0$, giả định k là khoảng cách giữa 2 ô. Ở đây ô 16 chéo nên có $\sqrt{2}$.

Từ đó, chúng ta có thể lập được hướng dòng chảy theo quy định về hướng như bảng mã hướng. Kết quả được như sau:



Từ đây chúng ta có thể xây dựng được cây dòng chảy.

3.2. Bước chuẩn bị cho việc xử lý

Các bước thực hiện:

- Giảng viên hướng dẫn sinh viên đọc dữ liệu từ ma trận hoặc tập tin hoặc bằng Excel.

- Xây dựng hàm tạo đồ thị cây theo các đỉnh. Giả định mỗi độ cao của đỉnh được cho là một số thực. Nếu giá trị độ cao là -9999 thì không xét hướng cho điểm tại vị trí đó.
- Xây dựng hàm tính toán hướng chảy của dòng tại mỗi đỉnh.
- Xây dựng cây (đồ thị) theo hướng dòng chảy tại mỗi đỉnh.
- Tính toán lượng tích trữ dòng tại mỗi đỉnh bằng cách đếm số node cha của đỉnh/node.

3.3. [Đọc thêm] Cài đặt thuật toán D8

Dưới đây là một hiện thực cho thuật toán D8 mà hàm tính tích lũy không phải dạng cây đồ thị.

Sinh viên có thể đọc hiểu và điều chỉnh theo hướng xây dựng đồ thị với mục đích chính là tính toán tích lũy dòng tại mỗi vị trí.

```
import math
import numpy as np

def tinh_huong(dem, m_dong, n_cot, vitri):
    huong = np.zeros((m_dong, n_cot))
    for dong in range(m_dong):
        for cot in range(n_cot):
            xx, yy, mymax = 0, 0, 0
            for i in range(-1,2):
                for j in range(-1,2):
                    if (i*i+j*j>0) and \
                      (dong+i>=0) and (dong+i<m_dong) and \
                      (cot+j>=0) and (cot+j<n_cot) and \
                  (mymax*math.sqrt(i*i+j*j) < dem[dong][cot] - dem[dong+i][cot+j]):
                        mymax = dem[dong][cot] - dem[dong+i][cot+j]
                        xx = i
```

```
                                    yy = j
                        huong[dong][cot] = vitri[xx+1][yy+1]
    return huong


def tinh_tichluy(p, q, m_dong, n_cot):
    for i in range(-1, 2):
        for j in range(-1,2):
            if (i*i+j*j>0) and \
               (p+i>=0) and (p+i<m_dong) and \
              (q+j>=0) and (q+j<n_cot) and \
             (huong[p+i][q+j] == vitri[-(i-1)][-(j-1)]):
                t1.append(1)
                tinh_tichluy(p+i, q+j, m_dong, n_cot)
    return len(t1)


vitri = np.array([ [32,64,128],[16,0,1],[8,4,2]])


#dem = [ [0 for j in range(n_cot)] for i in range(m_dong)]
# Giả định dữ liệu địa hình dem là một bảng 6x6 dưới đây:
dem = np.array([    [78,72,69,71,58,49],
                    [74,67,56,49,46,50],
                    [69,53,44,37,38,48],
                    [64,58,55,22,31,24],
                    [68,61,47,21,16,19],
                    [74,53,34,12,11,12] ])
```

```
m_dong = dem.shape[0] #6
n_cot = dem.shape[1] #6

tichluy = np.zeros((m_dong, n_cot)) #[ [0 for j in range(n_cot)] for i in range(m_dong)]

huong = tinh_huong(dem, 6,6, vitri)

for i in range(m_dong):
        for j in range(n_cot):
                t1 = []
                tichluy[i][j] = tinh_tichluy(i, j, m_dong, n_cot)
print (dem)
print (huong)
print (tichluy)
```

Kết quả tính toán:

```
================ RESTART: C:/Anaconda3/Scripts/singleflow.py ================
>>> dem
array([[78, 72, 69, 71, 58, 49],
       [74, 67, 56, 49, 46, 50],
       [69, 53, 44, 37, 38, 48],
       [64, 58, 55, 22, 31, 24],
       [68, 61, 47, 21, 16, 19],
       [74, 53, 34, 12, 11, 12]])
```

>>> **huong**

```
array([[ 2.,  2.,  2.,  4.,  4.,  8.],
       [ 2.,  2.,  2.,  4.,  8.,  8.],
       [ 1.,  1.,  2.,  4.,  8.,  4.],
       [128., 128.,  1.,  2.,  4.,  8.],
       [ 2.,  2.,  1.,  4.,  4.,  8.],
       [ 1.,  1.,  1.,  1.,  0., 16.]])
```

>>> **tichluy**

```
array([[ 0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  1.,  2.,  2.,  0.],
       [ 0.,  3.,  7.,  8.,  1.,  0.],
       [ 0.,  0.,  0., 20.,  0.,  1.],
       [ 0.,  0.,  0.,  1., 24.,  0.],
       [ 0.,  2.,  4.,  7., 35.,  0.]])
```

*Gợi ý: nếu ma trận **dem** có m dòng và n cột thì chúng ta phải xây dựng cây đồ thị có hướng có mxn đỉnh. Những đỉnh liên kết với nhau sẽ theo giá trị hướng dòng chảy.*

Sau đó, việc tìm dòng tích lũy bằng thuật toán tìm tổng số các node cha của một node.