Thuật toán tham lam

(GREEDY ALGORITHM)

I

GIỚI THIỆU

1. Khái niệm
2. Các thành phần của thuật toán

II

MỘT SỐ BÀI TOÁN ĐẶC TRƯNG

1. Huffman Coding
2. Bài toán xếp lịch cho các hoạt động
3. Bài toán cây khung nhỏ nhất sử dụng thuật toán Prim
4. Bài toán cái túi

III

ƯU - NHƯỢC ĐIỂM

GIỚI THIỆU

I.

Khái niệm

Thuật toán tham lam là mô hình thuật toán giải quyết vấn đề bằng cách lựa chọn hướng đem lại lợi ích tức thời trong từng giai đoạn.

8000đ ?


NGÂN HÀNG NHÀ NƯỚC VIỆT NAM
1000
ĐỒNG


NGÂN HÀNG NHÀ NƯỚC VIỆT NAM
2000
ĐỒNG


NGÂN HÀNG NHÀ NƯỚC VIỆT NAM
5000
ĐỒNG

```
getOptimal(Item, arr[], int n)
  1) Initialize empty result : result = {}
  2) While (All items are not considered)

      // We make a greedy choice to select
      // an item.
      i = SelectAnItem()

      // If i is feasible, add i to the
      // result
      if (feasible(i))
        result = result ∪ i
  3) return result
```

MỘT SỐ BÀI TOÁN ĐẶC TRƯNG

II.

01. Bài toán xếp lịch cho các hoạt động (Activity - Selection problem)

MÔ TẢ BÀI TOÁN

- **Bài toán :** Cho n công việc, thời gian bắt đầu và kết thúc của từng công việc. Tìm số công việc tối đa một người có thể hoàn thành, biết rằng người đó chỉ có thể hoàn thành một công việc trong một khoảng thời gian nhất định.

- **Ví dụ :**

Công việc	A1	A2	A3
Bắt đầu	12	10	20
Kết thúc	25	20	30

A2 ⇨ A3

TỔNG : 2

CÁC BƯỚC GIẢI QUYẾT

1

Sáp xếp các công việc theo thời gian kết thúc tăng dần

2

Chọn và in ra công việc đầu tiên của dãy

3

Tiếp tục xét các công việc còn lại, nếu công việc có thời gian bắt đầu lớn hơn hoặc bằng thời gian kết thúc công việc trước thì chọn

1. Activity – Selection Problem

- **Ví dụ :**

Công việc	A1	A2	A3	A4	A5	A6
Bắt đầu	0	3	1	5	5	8
Kết thúc	6	4	2	9	7	9

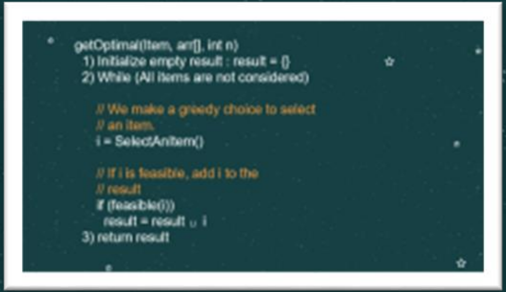
BƯỚC 1:

Công việc	A3	A2	A1	A5	A4	A6
Bắt đầu	1	3	0	5	5	8
Kết thúc	2	4	6	7	9	9

BƯỚC 2: **A3**

BƯỚC 3: **A3 ⇨ A2 ⇨ A5 ⇨ A6** **TỔNG : 4**

CÀI ĐẶT CHƯƠNG TRÌNH



```
getOptimal(Item, arr[], int n)
  1) Initialize empty result : result = {}
  2) While (All items are not considered)

     // We make a greedy choice to select
     // an item.
     i = SelectAnitem()

     // If i is feasible, add i to the
     // result
     if (feasible(i))
       result = result ∪ i
  3) return result
```

```
def MaxActivities(arr, n):
    selected = []

    # B1: Sắp xếp công việc theo thời gian kết thúc tăng dần
    Activity.sort(key = lambda x : x[1])

    # B2: Chọn ra công việc đầu tiên
    i = 0
    selected.append(arr[i])

    for j in range(1, n):

 #B3: Nếu công việc này có thời gian bắt đầu lớn hơn hoặc bằng thời gian kết
#thúc của công việc đã được chọn trước đó thì thêm công việc vào dãy selected
      if arr[j][0] >= arr[i][1]:
          selected.append(arr[j])
          i = j
    return selected
```

Select

feasible

ĐỘ PHỨC TẠP THUẬT TOÁN

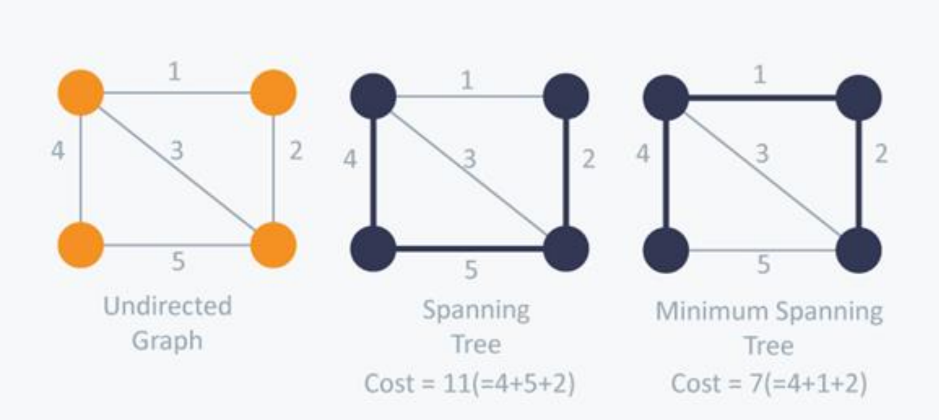
- Dãy đã được sắp xếp : O(n)
- Dãy chưa được sắp xếp : O(nlogn)

n = số công việc

02. Bài toán tìm cây khung nhỏ nhất sử dụng thuật toán Prim (Prim's Algorithm for MST)

NHẮC LẠI

- **Cây khung :** là một cây con chứa tất cả các đỉnh của đồ thị. Nói cách khác, cây khung là một tập hợp các cạnh của đồ thị, không chứa chu trình và kết nối tất cả các đỉnh của đồ thị.
- **Cây khung nhỏ nhất :** là cây khung có tổng trọng số các cạnh nhỏ nhất. Một đồ thị có thể có nhiều hơn một cây khung nhỏ nhất.
- **Ví dụ :**



2. Prim's Algorithm for MST

CÁC BƯỚC GIẢI QUYẾT

01

Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung

02

Tạo một mảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0

03

Nếu mstSet chưa chứa hết các đỉnh của đồ thị :

a.Chọn một đỉnh u chưa có trong mstS et và có giá trị key nhỏ nhất.

b. Thêm u vào mstSet.

c.Cập nhật lại key cho các đỉnh kề u: với v là đỉnh liền kề với u (v chưa có trong mstSet)
, nếu độ dài cạnh uv nhỏ hơn key của v
, gán key v = độ dài uv.

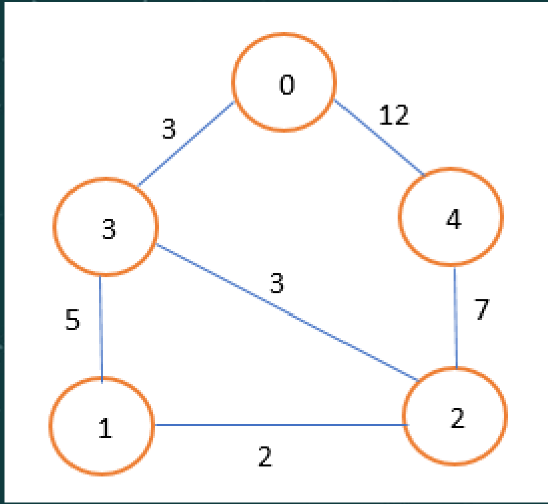
04

Kết thúc khi mstSet đã c hứa đầy đủ các đỉnh của đồ thị

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={}



Đỉnh	0	1	2	3	4
Key	0	∞	∞	∞	∞

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

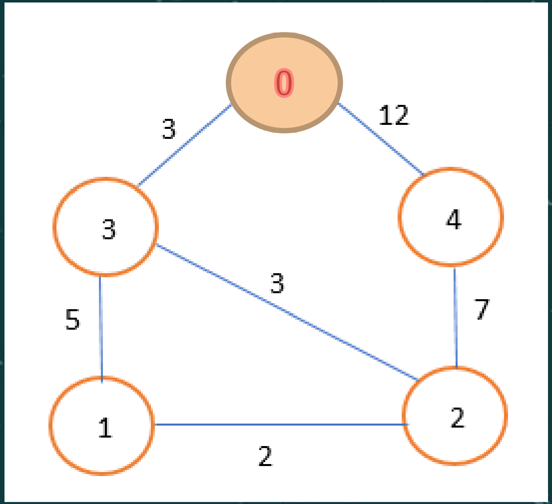
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0}



u

Đỉnh	0	1	2	3	4
Key	0	∞	∞	∞	∞

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

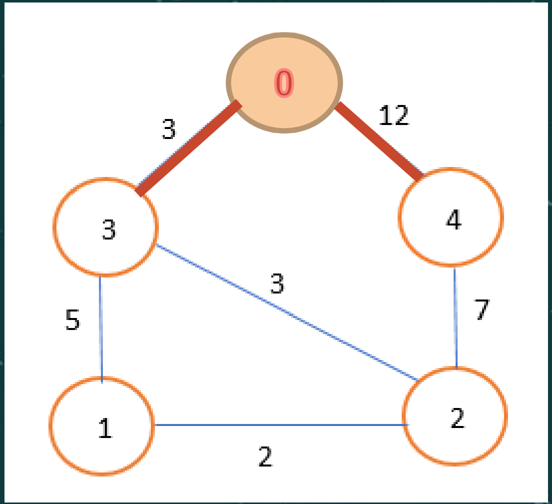
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0}



u ⬇ (0) v ⬇ (3) v ⬇ (4)

Đỉnh	0	1	2	3	4
Key	0	∞	∞	3	12

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

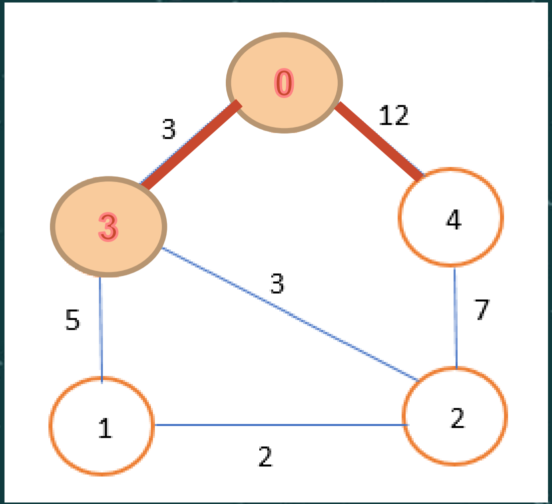
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3}



u

Đỉnh	0	1	2	3	4
Key	0	∞	∞	3	12

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

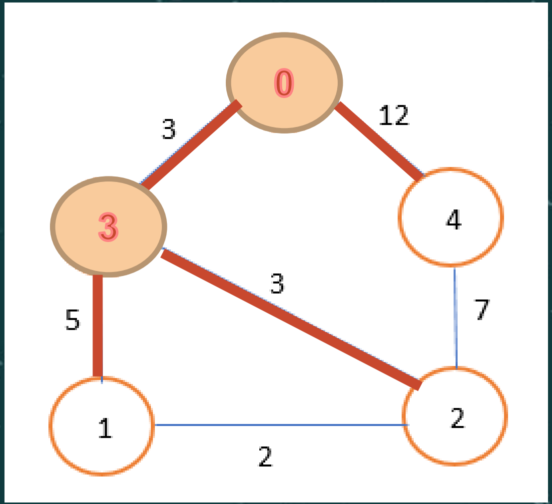
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3}



v v u

Đỉnh	0	1	2	3	4
Key	0	5	3	3	12

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

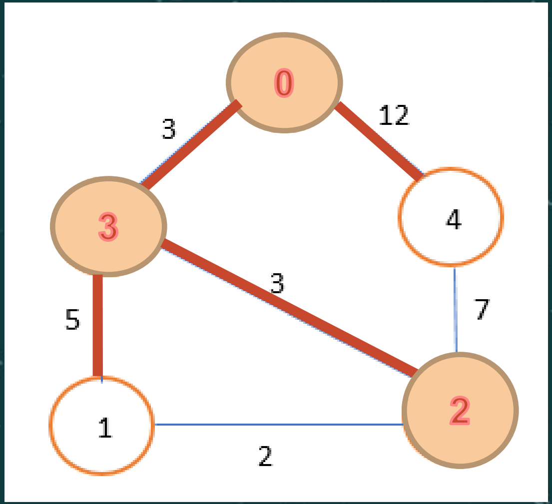
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3, 2}



u

Đỉnh	0	1	2	3	4
Key	0	5	3	3	12

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

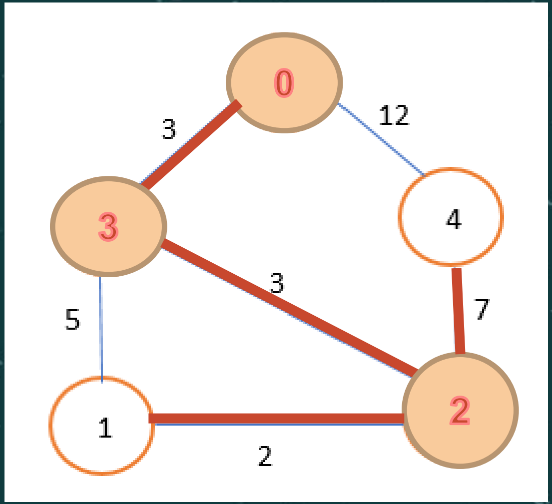
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3, 2}



v u v

Đỉnh	**0**	1	2	3	4
Key	**0**	2	3	3	7

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

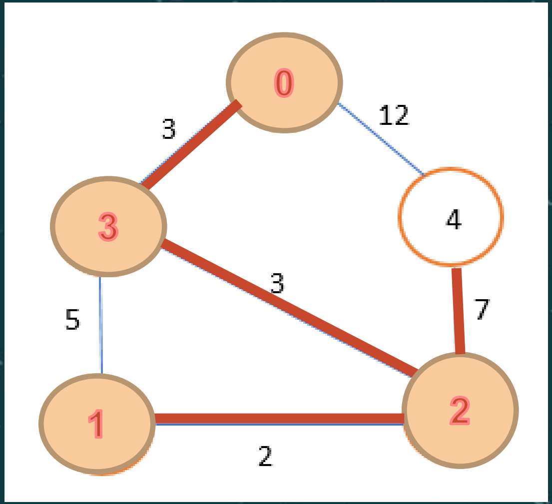
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3, 2, 1}



u

Đỉnh	**0**	1	2	3	4
Key	**0**	2	3	3	7

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

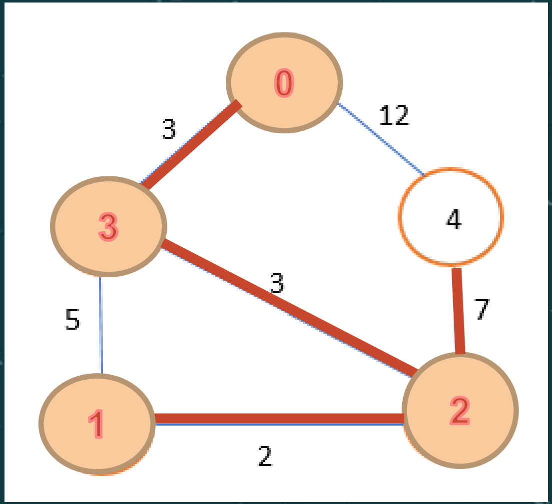
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3, 2, 1}



u

Đỉnh	0	1	2	3	4
Key	0	2	3	3	7

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

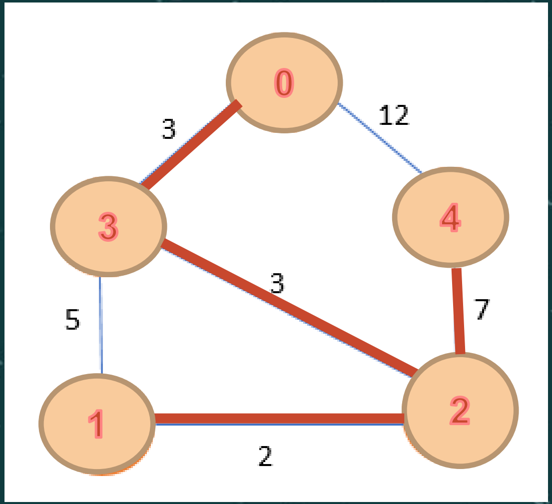
b) Thêm u vào mstSet.

c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :** mstSet ={0, 3, 2, 1, 4}



u

Đỉnh	0	1	2	3	4
Key	0	2	3	3	7

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

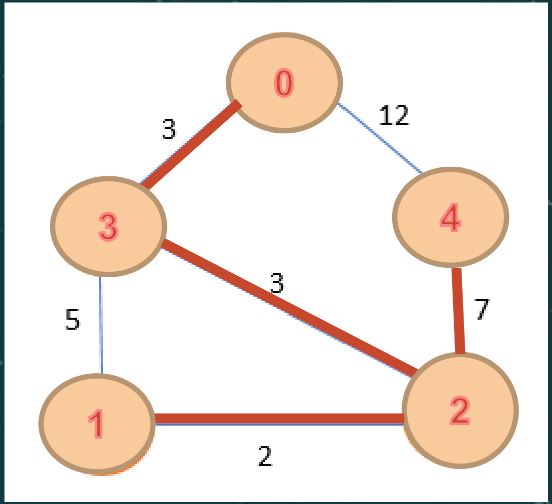
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3, 2, 1, 4}



u

Đỉnh	0	1	2	3	4
Key	0	2	3	3	7

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

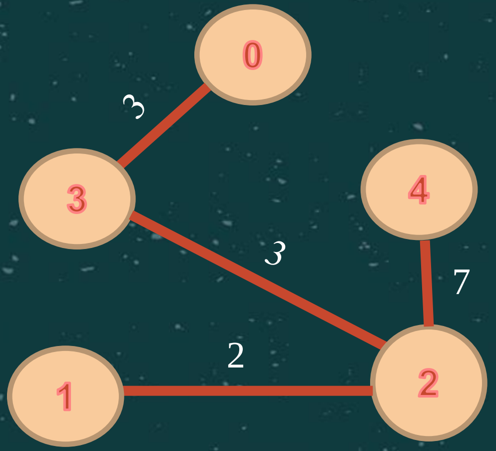
c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

- **Ví dụ :**

mstSet ={0, 3, 2, 1, 4}



Đỉnh	0	1	2	3	4
Key	0	2	3	3	7

Tổng = 15

1) Tạo một mảng mstSet lưu trữ các đỉnh đã có của cây khung.

2) Tạo một bảng lưu giá trị key tương ứng với mỗi đỉnh của đồ thị. Khởi tạo các key = ∞. Gán key của đỉnh đầu tiên = 0.

3) Nếu mstSet chưa chứa đầy đủ các đỉnh của đồ thị :

a) Chọn một đỉnh u chưa có trong mstSet và có giá trị key nhỏ nhất.

b) Thêm u vào mstSet.

c) Cập nhật lại key cho các đỉnh liền kề của u : với v là đỉnh liền kề với u, nếu độ dài cạnh uv nhỏ hơn key của v, gán key v = độ dài cạnh uv.

4) Kết thúc khi mstSet đã chứa đầy đủ các đỉnh của đồ thị.

2. Prim's Algorithm for MST

CÀI ĐẶT CHƯƠNG TRÌNH

```python
import sys

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    # Hàm in ra cây khung
    def printMST(self, parent):
        print ("Edge \tWeight")
        for i in range(1, self.V):
            print (parent[i], "-", i, "\t", self.graph[i][ parent[i] ] )

    # Hàm tìm đỉnh có key nhỏ nhất và chưa có trong mstSet
    def minKey(self, key, mstSet):

        #Khởi tạo giá trị min
        min = sys.maxsize

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v


        return min_index
```

select

```python
    def primMST(self):
        #B1 : Tạo mảng lưu đỉnh đã có trong mstSet
        mstSet = [False] * self.V

        # B2: Tạo mảng lưu giá trị key, key[0] = 0, còn lại = ∞
        key = [sys.maxsize] * self.V
        key[0] = 0
        # Để lưu đỉnh cha
        parent = [None] * self.V
        parent[0] = -1

        for cout in range(self.V):

            # B3a: Chọn u chưa có trong mstSet có key nhỏ nhất
            u = self.minKey(key, mstSet)

            # B3b: Thêm u vào mstSet
            mstSet[u] = True

            # B3c:
            for v in range(self.V):
                if self.graph[u][v] > 0 and mstSet[v] == False
                 and key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u

        self.printMST(parent)
```

feasible

ĐỘ PHỨC TẠP THUẬT TOÁN

- Nếu đồ thị đầu vào là dạng ma trận : O(V^2)
- Nếu đồ thị đầu vào là dạng danh sách liền kề : O(E log(V))

V = số đỉnh của đồ thị

03. Huffman Coding

MÔ TẢ BÀI TOÁN

- **Bài toán :** Cho tập A gồm các kí hiệu và tần suất xuất hiện của chúng. Tìm một bộ mã tiền tố với tổng độ dài mã hóa là nhỏ nhất.

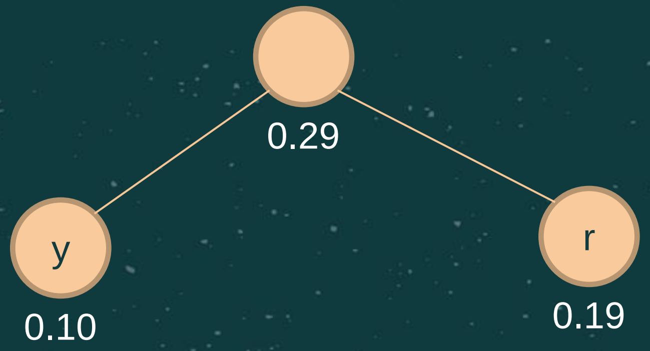
- *Mã tiền tố: Mã tiền tố là bộ các từ mã của một tập hợp các ký hiệu sao cho từ mã của mỗi ký hiệu không là tiền tố (phần đầu) của từ mã một ký hiệu khác trong bộ mã ấy.*
- *Từ mã: Các ký hiệu (kí tự, chữ số,…) được thay bằng các xâu nhị phân.*

- **Ví dụ : greedy gồm 5 chữ cái g, r, e, d, y tạo thành thì ta sẽ có g = 10, r = 01, e = 11, d = 001, y = 000 là bộ mã tiền tố.**

CÁC BƯỚC GIẢI QUYẾT

Bước 1: Tạo một rừng có n cây (n là số kí tự không trùng nhau).

Bước 2: Chọn 2 cây có trọng số nhỏ nhất, dùng một gốc để nối 2 cây lại, mỗi cây ban đầu sẽ là 1 con của cây mới.và trọng số của cây mới bằng tổng trọng số của 2 cây ban đầu.



Bước 3: Lặp lại bước 2 cho đến khi rừng chỉ còn 1 cây.

Từ tài liệu tham khảo 3, hãy chỉ ra chỗ nào trong code của Huffman coding tương ứng với phần nào trong code tổng quát slide 6.

ĐỘ PHỨC TẠP THUẬT TOÁN

O(nlogn) với n là số kí tự không trùng nhau.

04. Bài toán cái túi (Knapsack Problem)

MÔ TẢ BÀI TOÁN

Weight = w1

Value = v1

Weight = w2

Value = v2

…

Weight = w_n

Value = v_n

Tổng sức chứa = **W**

<u>Yêu cầu</u> : Lấp đầy túi sao cho tổng giá trị Value lớn nhất và tổng cân nặng Weight ≤ W

BÀI TOÁN CÁI TÚI DẠNG PHÂN SỐ
(Fractional Knapsack Problem)

- Các món đồ có thể cắt nhỏ.
- Ví dụ :

W = 10

Object	**1**	**2**	**3**	**4**	**5**
Value	21	16	80	14	10
Weight	3	4	5	7	1

Object	**1**	**2**	**3**	**4**	**5**
Value	21	16	80	14	10
Weight	3	4	5	7	1
Ratio(v/w)	7	4	16	2	10

Object	**3**	**5**	**1**	**2**	**4**
Value	80	10	21	16	14
Weight	5	1	3	4	7
Ratio(v/w)	16	10	7	4	2

Đáp án :

1) O3
- W còn lại = 5
- V = 80

2) O3 ➡ O5
- W còn lại = 4
- V = 90

3) O3 ➡ O5 ➡ O1
- W còn lại = 1
- V = 111

4) O3 ➡ O5 ➡ O1 ➡ ¼ O2
- W còn lại = 0
- V = 115

CÁC BƯỚC GIẢI QUYẾT

01
Tính tỉ lệ cho mỗi món đồ (Ratio = Value / Weight)

02
Sắp xếp các món đồ theo tỉ lệ giảm dần

03
Lần lượt chọn ra món đồ có tỉ lệ lớn nhất cho đến khi không thể lấy toàn bộ món đồ tiếp theo.

04
Cuối cùng, lấy phần có thể lấy của món đồ tiếp theo.

4. Knapsack Problem

CÀI ĐẶT CHƯƠNG TRÌNH

```python
class ItemValue:
    def __init__(self, wt, val, ind):
        self.wt = wt
        self.val = val
        self.ind = ind
        self.ratio = val // wt

    def __lt__(self, other):
        return self.ratio < other.ratio

class FractionalKnapSack:
    def getMaxValue(wt, val, capacity):
        iVal = []
        #B1 : Tính tỉ lệ ratio = val/wt
        for i in range(len(wt)):
            iVal.append(ItemValue(wt[i], val[i], i))
        # B2 : sắp xếp theo giá trị ratio
        iVal.sort(reverse=True)
```

Select

```python
        totalValue = 0
        for i in iVal:
            curWt = int(i.wt)
            curVal = int(i.val)
            # B3 : lấy nguyên các món đồ có tỉ lệ lớn
            if capacity - curWt >= 0:
                capacity -= curWt
                totalValue += curVal
            #B4 : khi không thể lấy nguyên món đồ, lấy phần có thể lấy
            else:
                fraction = capacity / curWt
                totalValue += curVal * fraction
                capacity = int(capacity - (curWt * fraction))
                break
        return totalValue
```

Feasible

ĐỘ PHỨC TẠP THUẬT TOÁN

T = O(nlogn)

n = số món đồ

BÀI TOÁN CÁI TÚI DẠNG 0/1
(0/1 Knapsack Problem)

- Các món đồ không thể cắt nhỏ.
- Ví dụ :

W = 10

Object	**1**	**2**	**3**	**4**	**5**
Value	2	6	8	12	1
Weight	3	4	5	7	1

Object	**4**	**3**	**2**	**1**	**5**
Value	12	8	6	2	1
Weight	7	5	4	3	1

- Đáp án: O4 ➡ O1 — W = 10, V = 14
- Trường hợp tốt hơn : O3 ➡ O2 ➡ O5 — W = 10, V = 15

➡ **Thuật toán tham lam không hoạt động hiệu quả trong bài toán cái túi dạng 0/1**

Tại sao giải thuật tham lam hoạt động hiệu quả cho bài toán cái túi dạng phân số nhưng không hoạt động hiệu quả trong bài toán cái túi dạng 0/1?

ĐẶC ĐIỂM NHẬN DẠNG

A

Có đặc tính tham lam:
Chọn giải pháp tốt nhất hiện tại, không xem xét lại lựa chọn của mình

B

Có cấu trúc con tối ưu:
Một bài toán có cấu trúc con tối ưu khi lời giải tối ưu của các bài toán con dẫn đến lời giải tối ưu của cả bài toán.

**Để hiểu hơn về 2 đặc điểm nhận dạng này, các bạn hãy tham khảo ở tài liệu tham khảo 1.*

ƯU ĐIỂM - NHƯỢC ĐIỂM

III.

ƯU ĐIỂM

+ Đơn giản, dễ tiếp cận và triển khai thực hiện vì việc chúng ta làm ở từng bước là lấy lựa chọn tốt nhất.
+ Độ phức tạp rõ ràng: từng bước giải của bài toán riệng lẻ nhau, không gối lên nhau.
+ So sánh 1 bài toán có thể giải bằng thuật toán tham lam hay nói cách khác là có đặc điểm nhận diện trên thì tham lam chạy nhanh hơn so với các thuật toán khác.

NHƯỢC ĐIỂM

- Vì thuật toán tham lam không suy xét lại những lựa chọn trước đó nên đôi khi sẽ không mang lại kết quả tối ưu toàn cục.
- Khó để chưng minh tính đúng đắn của thuật toán. [4]

Khái niệm
Thuật toán tham lam là mô hình thuật toán giải quyết vấn đề bằng cách lựa chọn hướng đem lại lợi ích tức thời trong từng giai đoạn.
getOptimal(Item, arr[], int n)
1) Initialize empty result : result = {}
2) While (All items are not considered)
// We make a greedy choice to select
// an item.
i = SelectAnItem()
// If i is feasible, add i to the
// result
if (feasible(i))
result = result ∪ i
3) return result
ĐẶC ĐIỂM NHẬN DẠNG
*Để hiểu hơn về 2 đặc điểm nhận dạng này, các bạn hãy tham khảo ở tài liệu tham khảo 1.
ƯU ĐIỂM
NHƯỢC ĐIỂM
Vì thuật toán tham lam không suy xét lại những lựa chọn trước đó nên nhiều khi sẽ không mang lại kết quả tối ưu toàn cục.
Khó để chứng minh tính đúng đắn của thuật toán. [4]

Thành viên

Nguyễn Phương Bảo Ngọc - 19521907
Phạm Đỗ Hoàng My - 19521863

Tài liệu tham khảo

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "16 Greedy Algorithms". Introduction To Algorithms. MIT Press. pp. 370–. ISBN 978-0-262-03293-3.
2. Greedy Algorithms (General Structure and Applications) – GeeksForGeeks (1/4/2021): https://www.geeksforgeeks.org/greedy-algorithms-general-structure-and-applications/
3. Aashish Barnwal – GeeksforGeeks, *Huffman Coding | Greedy Algo-3 : https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/*
4. Magnus Lie Hetland (2008), *Proof methods and greedy algorithms,* Lecture notes: https://folk.idi.ntnu.no/mlh/algkon/greedy.pdf

Thank you!

Do you have any questions?