

TÌM CÂY KHUNG NHỎ NHẤT

1. Giải thuật Prim

Cho $G = (V, E)$ là đồ thị vô hướng liên thông có trọng số gồm n đỉnh. Giải thuật Prim được dùng để tìm ra cây khung nhỏ nhất của G .

Bước 1: Chọn tùy ý $v \in X$. Khởi tạo $Y := \{v\}$ và $T := \emptyset$. V là tập đỉnh của đồ thị, Y là tập đỉnh được chọn vào cây khung nhỏ nhất và T là tập cạnh của cây khung nhỏ nhất.

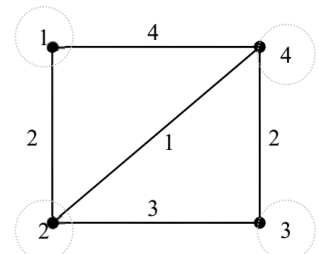
Bước 2: Trong số những cạnh e nối đỉnh $v \in Y$ với đỉnh $w \in V \setminus Y$, ta chọn cạnh có trọng số nhỏ nhất.

Bước 3: Gán $Y := Y \cup \{w\}$ và $T := T \cup \{e\}$

Bước 4: Nếu T đủ $n - 1$ phần tử thì dừng, ngược lại làm tiếp **Bước 2**.

Nhận xét: có thể thấy trong khi tìm khung ngắn nhất, ta không xét cạnh khuyên vì một đỉnh không thể vừa thuộc Y vừa thuộc $V \setminus Y$, tương tự, ta chỉ chọn cạnh có trọng số nhỏ nhất trong số cạnh bội có cùng đỉnh bắt đầu và đỉnh kết thúc.

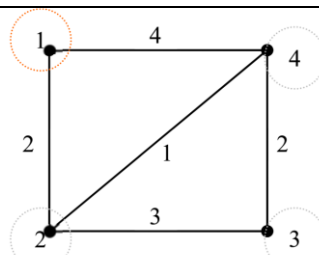
Ví dụ, cho đồ thị như hình bên, tìm cây khung nhỏ nhất



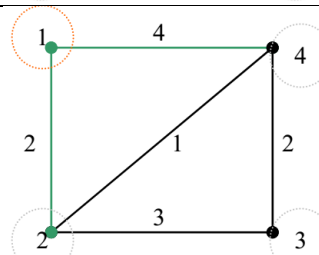
Bước 1: Chọn tùy ý $v \in V = \{1,2,3,4\}$, trong ví dụ này ta chọn đỉnh 1 làm đỉnh xét đầu tiên.

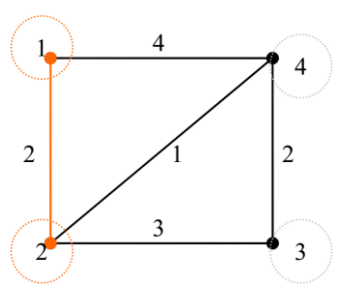
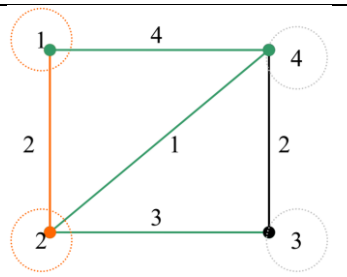
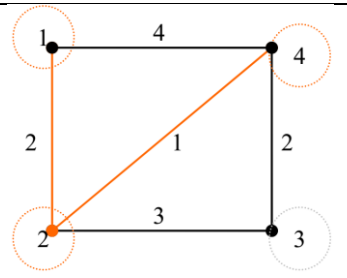
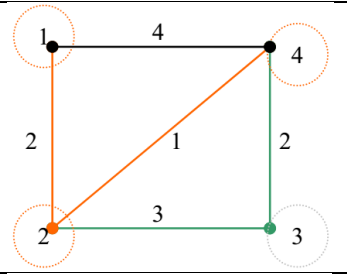
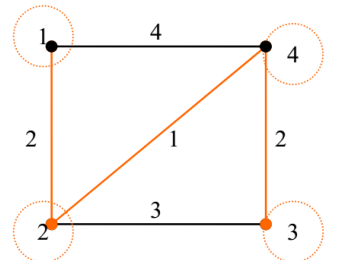
$Y := \{1\}$, như vậy $V \setminus Y = \{2,3,4\}$

$T := \emptyset$



Bước 2: Cạnh $(1,2)$ và $(1,4)$ lần lượt nối đỉnh $1 \in Y$ với đỉnh $2,4 \in V \setminus Y$. Chọn cạnh $(1,2)$ vì trọng số nhỏ hơn.



<p>Bước 3: Gán $Y := Y \cup \{2\}$ và $T := T \cup \{(1,2)\}$</p> <p>$Y := \{1,2\}$, như vậy $V \setminus Y = \{3,4\}$</p> <p>$T := \{(1,2)\}$</p> <p>Bước 4: T có 1 phần tử $< n - 1 = 3$ nên làm tiếp.</p>	
<p>Bước 2 (lần 2): Cạnh $(1,4)$, $(2,3)$ và $(2,4)$ lần lượt nối đỉnh $1,2 \in Y$ với đỉnh $3,4 \in V \setminus Y$. Chọn cạnh $(2,4)$ có trọng số nhỏ nhất.</p>	
<p>Bước 3 (lần 2): Gán $Y := Y \cup \{4\}$ và $T := T \cup \{(2,4)\}$</p> <p>$Y := \{1,2,4\}$, như vậy $V \setminus Y = \{3\}$</p> <p>$T := \{(1,2), (2,4)\}$</p> <p>Bước 4 (lần 2): T có 2 phần tử $< n - 1 = 3$ nên làm tiếp.</p>	
<p>Bước 2 (lần 3): Cạnh $(2,3)$ và $(4,3)$ lần lượt nối đỉnh $2,3 \in Y$ với đỉnh $3 \in V \setminus Y$. Chọn cạnh $(4,3)$ có trọng số nhỏ hơn.</p>	
<p>Bước 3 (lần 3): Gán $Y := Y \cup \{3\}$ và $T := T \cup \{(4,3)\}$</p> <p>$Y := \{1,2,4,3\}$, như vậy $V \setminus Y = \emptyset$</p> <p>$T := \{(1,2), (2,4), (4,3)\}$</p> <p>Bước 4 (lần 3): T đã có đủ 3 cạnh. Dừng giải thuật.</p>	

Định nghĩa lớp **AdjacencyMatrix** để biểu diễn **đồ thị vô hướng có trọng số** bằng ma trận kề (đã có hướng dẫn). Trong ma trận kề, gọi $[i, j]$ là giá trị tại dòng i cột j ($i, j = 0, \dots, n-1$). $[i, j] = 0$: không có cạnh nối từ đỉnh i đến đỉnh j , $[i, j] = x > 0$: có cạnh nối từ i đến j với trọng số x dương.

```
class AdjacencyMatrix {
    public int n;
    public int[,] a;
    public bool readAdjacencyMatrix(string filename){
        // .....
    }
    public void showAdjacencyMatrix(){
        // .....
    }
}
```

Định nghĩa lớp **EDGE** để biểu diễn cạnh

```
class EDGE {  
    int v;           // Đỉnh bắt đầu  
    int w;           // Đỉnh kết thúc  
    int weight;      // Trọng số của cạnh v-w  
    // Hàm thành viên để truy cập biến của class  
    public int V { // ... }  
    public int W { // ... }  
    public int Weight { // ... }  
}
```

Khai báo các biến cần thiết cho giải thuật

```
EDGE []T;           // cạnh của cây khung ngắn nhất  
int nT;             // số cạnh của cây T  
bool []marked;      // đánh dấu đỉnh đã xét (true) và đỉnh chưa xét (false)  
// đóng vai trò như tập V và Y trong giải thuật trên
```

Giả sử đồ thị được biểu diễn bởi biến **g** kiểu **AdjacencyMatrix**. Khởi tạo các biến khai báo ở trên.

Mảng T có kích thước g.n-1	T = new EDGE [g.n-1];
Số cạnh của cây khung ban đầu là 0	nT = 0;
Mảng marked có kích thước g.n Khởi tạo nhãn của các đỉnh là chưa xét (false)	marked = new bool [g.n]; for (int i = 0; i < g.n ; ++ i) marked [i] = false ;
Gán nhãn đã xét (true) cho đỉnh bắt đầu	marked [source] = true ;

Định nghĩa hàm thực hiện giải thuật Prim trên **đồ thị vô hướng có trọng số**

```
public void Prim (AdjacencyMatrix g, int source) {  
    // khởi tạo các biến cần thiết  
    // ... ..  
    // thực hiện giải thuật  
    while (nT < ...){ // trong khi chưa đạt số cạnh tối đa của cây khung  
        EDGE edgeMin;  
        int nMinWeight = -1; // -1 nghĩa là chưa có cạnh min  
        // duyệt các đỉnh w thỏa điều kiện chưa xét  
        for (w...)  
            if (...){  
                // tìm v bất kỳ đã xét và có cạnh nối trực tiếp v - w  
                for (v...)  
                    if (... && ...){  
                        // nếu chưa có cạnh min (nMinWeight < 0)  
                        // hoặc trọng số (v, w) < nMinWeight  
                        // thì cập nhật lại edgeMin = (v, w)  
                        if (... || ...){  
                            edgeMin.v = v; edgeMin.w = w;  
                            nMinWeight = ...;  
                        }  
                    }  
            }  
    }  
}
```

```

T[nT++] = edgeMin;           // thêm cạnh edgeMin vào cây khung
// gán nhãn đã xét cho đỉnh w
...
}
...                           // in cây khung từ mảng T
}

```

2. Giải thuật Kruskal

Cho $G = (V, E)$ là một đồ thị liên thông có trọng số gồm n đỉnh.

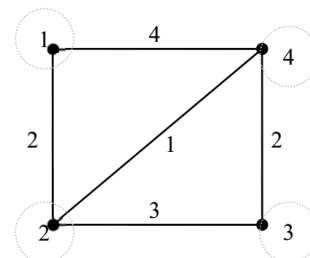
Bước 1: Sắp xếp các cạnh theo thứ tự trọng số tăng dần và khởi tạo $T := \emptyset$.

Bước 2: Lấy cạnh e đầu tiên trong danh sách đã sắp xếp (tức là cạnh có trọng số nhỏ nhất). Nếu $T \cup \{e\}$ không chứa chu trình thì gán $T := T \cup \{e\}$. Loại e khỏi danh sách.

Bước 3: Nếu T đủ $n - 1$ phần tử thì dừng, ngược lại làm tiếp **Bước 2**.

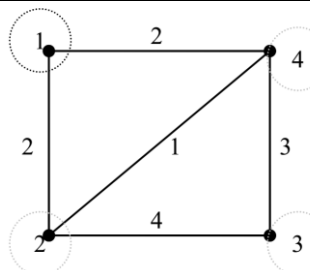
Giải thuật Prim	Giải thuật Kruskal
Bắt đầu xây dựng cây khung từ bất kỳ đỉnh nào trong đồ thị	Bắt đầu xây dựng cây khung từ đỉnh thuộc về cạnh có trọng số nhỏ nhất trong đồ thị
Duyệt mỗi đỉnh nhiều hơn một lần để xác định khoảng cách nhỏ nhất	Duyệt mỗi đỉnh chỉ một lần do đã sắp xếp trước
Độ phức tạp $O(V^2)$, với V là số đỉnh, có thể cải thiện thành $O(E + \log V)$ sử dụng Fibonacci heap	Độ phức tạp $O(E \log V)$, với V là số đỉnh
Chạy nhanh hơn Kruskal trên đồ thị dày	Chạy nhanh hơn Prim trên đồ thị thưa

Ví dụ, cho đồ thị như hình bên, tìm cây khung nhỏ nhất



Bước 1: Sắp xếp các cạnh theo thứ tự trọng số tăng dần và đưa vào $lstEdge$, như vậy $lstEdge = \{(2,4), (1,2), (1,4), (3,4), (2,3)\}$.

Khởi tạo $T = \emptyset$

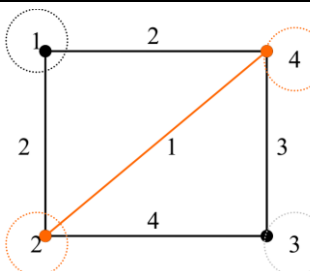


Bước 2: Thêm cạnh $e = (2,4)$ vào T vì $T \cup \{e\}$ không tạo chu trình

$lstEdge = \{(1,2), (1,4), (3,4), (2,3)\}$

$T = \{(2,4)\}$

Bước 3: T có 1 phần tử $< n - 1 = 3$ nên làm tiếp.



<p>Bước 2 (lần 2): Thêm $e = (1,2)$ vào T vì $T \cup \{e\}$ không tạo chu trình</p> <p>$lstEdge = \{(1,4), (3,4), (2,3)\}$</p> <p>$T = \{(2,4), (1,2)\}$</p> <p>Bước 3 (lần 2): T có 2 phần tử $< n-1 = 3$ nên làm tiếp.</p>	
<p>Bước 2 (lần 3): Không thêm $e = (1,4)$ vào T vì $T \cup \{e\}$ tạo chu trình</p> <p>$lstEdge = \{(3,4), (2,3)\}$</p> <p>$T = \{(2,4), (1,2)\}$</p> <p>Bước 3 (lần 3): T có 2 phần tử $< n-1 = 3$ nên làm tiếp.</p>	
<p>Bước 2 (lần 4): Thêm $e = (3,4)$ vào T vì $T \cup \{e\}$ không tạo chu trình</p> <p>$lstEdge = \{(2,3)\}$</p> <p>$T = \{(2,4), (1,2), (3,4)\}$</p> <p>Bước 3 (lần 4): T đã có đủ 3 cạnh. Dừng giải thuật.</p>	

Định nghĩa cấu trúc dữ liệu **AdjacencyMatrix** để biểu diễn **đồ thị vô hướng có trọng số** và định nghĩa cấu trúc dữ liệu **EDGE** để biểu diễn cạnh. Thực hiện tương tự như trong giải thuật Prim.

Khai báo các biến cần thiết cho giải thuật

```

EDGE []T;           // cạnh của cây khung ngắn nhất
int nT;             // số cạnh của cây T
int []label;        // nhãn của đỉnh đã xét, dùng để phát hiện chu trình
EDGE []lstEdges;    // danh sách cạnh của đồ thị (đã xếp theo trọng số)
int nEdges;         // số cạnh có trong lstEdge

```

Giả sử đồ thị được biểu diễn bởi biến g kiểu **AdjacencyMatrix**. Khởi tạo các biến khai báo ở trên.

Mảng T có kích thước $g.n-1$ Số cạnh của cây khung ban đầu là 0	$T = \text{new } \text{EDGE}[g.n-1];$ $nT = 0;$
Mảng $lstEdges$ có kích thước tối đa là $g.n*(g.n-1)$ Số cạnh có trong $lstEdges$ ban đầu là 0	$lstEdges = \text{new } \text{EDGE}[g.n*(g.n-1)];$ $nEdges = 0;$
Mảng $label$ có kích thước $g.n$ Khởi tạo nhãn của các đỉnh là chỉ mục của đỉnh	$label = \text{new } \text{int}[g.n];$ $\text{for } (\text{int } i = 0; i < g.n; ++i)$ $\quad label[i] = i;$

Định nghĩa các hàm tiện ích

```

/*Sắp xếp cạnh theo thứ tự tăng dần về trọng số*/
private void SortListEdge(AdjacencyMatrix g){
    // Áp dụng giải thuật sắp xếp
}

```

```

/*Khởi tạo mảng chứa mọi cạnh của đồ thị */
private void InitListEdge(AdjacencyMatrix g){
    nEdgeCount = 0;
    for (i...)          // Duyệt các cạnh của đồ thị
        for (j...)
            if (...){
                lstEdge[nEdgeCount].v = ...; lstEdge[nEdgeCount].w = ...;
                lstEdge[nEdgeCount].weight = ...;
                nEdgeCount++;
            }
}

```

```

/* Kiểm tra khi thêm một cạnh có chỉ số idx trong danh sách lstEdge có tạo
thành chu trình không */
private bool IsCircle(int idx){
    // Kiểm tra nhãn của hai đỉnh ứng với cạnh idx có giống nhau hay không
    // Nếu có, trả về true, cạnh này tạo thành chu trình.
    // Nếu không, gán nhãn nhỏ hơn cho biến lab1 và nhãn lớn hơn cho biến lab2
    ... ..
    // Cập nhật toàn bộ đỉnh có nhãn lab2 bằng giá trị lab1
    ... ..
    return ...;
}

```

Định nghĩa hàm thực hiện giải thuật Kruskal trên **đồ thị vô hướng có trọng số**

```

public void KruskalAlg(AdjacencyMatrix g){
    ... ..          // khởi tạo các biến cần thiết
    int eMinIndex = 0;      // chỉ mục của cạnh nhỏ nhất
    while (nT < ...){       // trong khi chưa đạt số cạnh tối đa của cây khung
        if (eMinIndex < nEdgeCount ) { // cạnh e nhỏ nhất chưa xét
            //Cạnh này có tạo thành chu trình khi thêm vào không?
            if (IsCircle(eMinIndex) == false)
                // thêm cạnh có chỉ mục eMinIndex vào cây
                T[nT++] = ...;
            eMinIndex++;
        }
        else{
            // Dừng giải thuật trong trường hợp thất bại
        }
    }
    ...              // in cây khung từ mảng T
}

```

3. Bài tập tự rèn luyện

3.1 Dùng tham số dòng lệnh để nhận tập tin chứa ma trận trọng số biểu diễn đồ thị từ người dùng. Xuất ra tập tin cây khung tìm được và giá trị độ lớn của cây khung.

Ví dụ: *mssv_tuanX.exe input.txt output.txt*

input.txt	output.txt
4	1 2
0 2 0 4	2 4
2 0 3 1	4 3
0 3 0 2	5
4 1 2 0	

3.2 Cho dạng đồ thị được biểu diễn trong tập tin **inp.txt** theo quy tắc sau:

- Dòng đầu là số đỉnh của đồ thị.
- Dòng thứ hai là số cạnh của đồ thị.
- Các dòng tiếp theo, mỗi dòng biểu diễn một cạnh của đồ thị và trọng số đi kèm theo định dạng *đỉnh_1 đỉnh_2 trọng_số*, với *đỉnh_1* và *đỉnh_2* là hai đầu của cạnh

Quy ước:

- Đỉnh được đánh nhãn từ 0, 1, 2,...
- Đồ thị không có khuyên và không có cạnh âm.

Áp dụng giải thuật Prim tìm cây khung nhỏ nhất và xuất ra tập tin **out.txt** với quy ước mỗi dòng là một cạnh của cây khung (không quan trọng thứ tự)

Lưu ý:

- Chương trình nhận tên tập tin dưới dạng tham số dòng lệnh.
- Giữa một cặp đỉnh có thể có nhiều cạnh nối.

Ví dụ

inp.txt	out.txt
4	3 0
5	1 3
0 1 9	1 2
1 2 8	
3 0 5	
0 3 6	
1 3 3	

3.3 Trong thành phố gồm N địa điểm có tọa độ thực (x_i, y_i) . Chi phí để lắp đặt dây liên lạc giữa địa điểm i và địa điểm j tỉ lệ với khoảng cách giữa hai điểm. Nếu đã có dây liên lạc giữa điểm i và điểm k, giữa điểm k và điểm j thì có thể liên lạc giữa địa điểm i và điểm j. Hãy chỉ ra cách nối dây để đảm bảo liên lạc giữa các địa điểm với chi phí thấp nhất.

Gợi ý: bài này được xem như là một đồ thị đủ, với trọng là khoảng cách giữa các đỉnh. Dữ liệu nhập vào từ tập tin **THANHPHO.INP** có định dạng:

- Dòng đầu tiên chứa số đỉnh ($n < 100$)
- N dòng tiếp theo chứa tọa độ (x_i, y_i) của đỉnh thứ i .

Dữ liệu xuất ra tập tin **THANHPHO.OUT** có định dạng:

- Dòng đầu tiên gồm 2 số m, k trong đó m là số cạnh, và k là chi phí.
- m dòng tiếp theo là danh sách các cạnh của cây khung.

Ví dụ

THANHPHO.INP		THANHPHO.OUT			
n		m	k		
x1	y1	xi1	yi1	xi2	yi2
x2	y2	...			
...	...				
xn	yn				

3.4 Trong một khu vực có N thành phố. Người ta muốn xây dựng một hệ thống các tuyến đường sắt nối các thành phố lại với nhau với tiêu chí tiết kiệm tối đa. Chi phí được tính dựa trên tổng chiều dài các tuyến. Giả sử rằng giữa 2 thành phố bất kỳ đều có đường đi, và đường đi này là thẳng (tức là tương đương đường chim bay). Bạn hãy thiết kế hệ thống đường sắt này.

Dữ liệu nhập vào từ tập tin **cities.txt** có định dạng:

- Dòng đầu tiên chứa số đỉnh
- N dòng tiếp theo chứa tọa độ $(x[i], y[i])$ của đỉnh thứ i (i là chỉ số bắt đầu từ 1) theo định dạng
 $i \quad x[i] \quad y[i]$

Dữ liệu xuất ra tập tin **result.txt** có định dạng:

- Dòng đầu tiên: tổng chiều dài các tuyến đường
- $N-1$ dòng tiếp theo, mỗi dòng thể hiện 1 tuyến đường trong hệ thống đường sắt theo định dạng $i \quad j \quad c[i,j]$, với $c[i,j]$ là khoảng cách từ thành phố thứ i đến thành phố thứ j .

Hướng dẫn: tuyến đường sắt cần xây dựng chính là cây khung nhỏ nhất của đồ thị có N đỉnh, mỗi đỉnh chính là một thành phố.