

HƯỚNG DẪN THỰC HÀNH TÌM CÂY KHUNG NHỎ NHẤT - KRUSKAL

1 Thuật toán

Cho $G=(X,E)$ là một đồ thị liên thông có trọng số gồm n đỉnh. Thuật toán Prim được dùng để tìm ra cây khung nhỏ nhất của G .

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

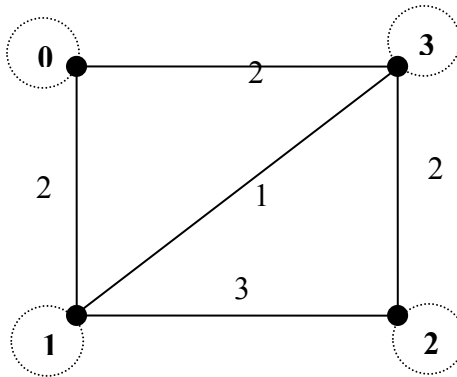
Bước 2: Lấy cạnh e ở đầu danh sách đã sắp xếp (có trọng nhỏ nhất). Nếu $T + \{e\}$ không chứa chu trình thì gán $T := T + \{e\}$. Loại cạnh e khỏi danh sách.

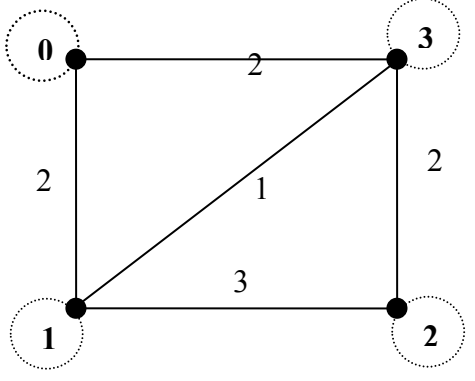
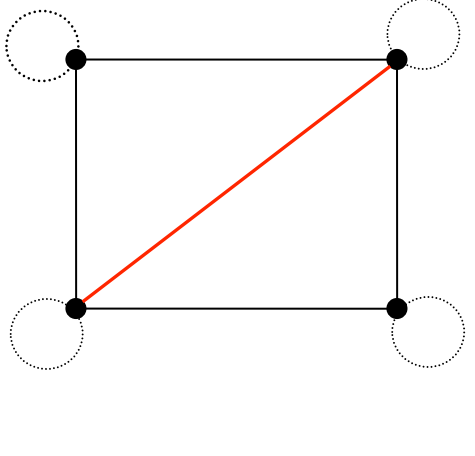
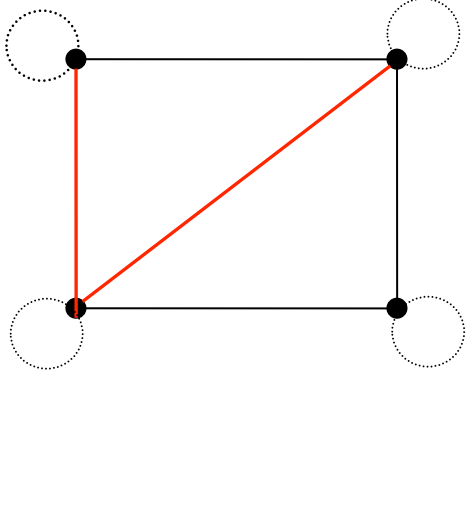
Bước 3: Nếu T đủ $n-1$ phần tử thì dừng, ngược lại làm tiếp tục bước 2

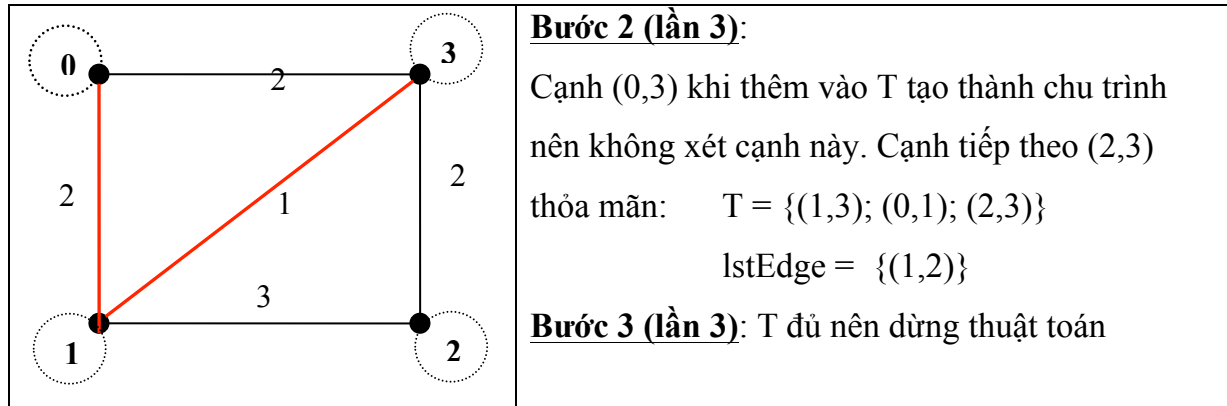
Chú ý: trong các thuật toán tìm khung nhỏ nhất chúng ta có thể bỏ đi hướng các cạnh và các khuyên; đối với cạnh song song thì có thể bỏ đi và chỉ để lại một cạnh trọng lượng nhỏ nhất trong chúng.

2 Ví dụ

Tìm cây khung nhỏ nhất của đồ thị sau:



	<p>Bước 1: Sắp xếp các cạnh theo thứ tự trọng lượng tăng dần và khởi tạo $T := \emptyset$.</p> <p>Gọi lstEdge là tập các cạnh đã sắp thứ tự trọng tăng:</p> $\text{lstEdge} = \{(1,3); (0,1); (0,3); (2,3); (1,2);\}$ $T = \emptyset.$
	<p>Bước 2: Lấy cạnh e ở đầu danh sách đã sắp xếp (có trọng nhỏ nhất). Nếu $T + \{e\}$ không chứa chu trình thì gán $T := T + \{e\}$. Loại cạnh e khỏi danh sách.</p> <p>Vì cạnh $(1, 3)$ bổ sung vào T ($T = \emptyset$) không tạo thành chu trình nên: $T = \{(1, 3)\}$.</p> <p>Danh sách cạnh:</p> $\text{lstEdge} = \{(0,1); (0,3); (2,3); (1,2);\}$
	<p>Bước 3: Nếu T đủ $n-1$ phần tử thì dừng, ngược lại làm tiếp tục bước 2.</p> <p>T chỉ có 1 phần tử $< n - 1 = 4 - 1 = 3$ nên thuật toán chưa dừng.</p> <p>Bước 2 (lần 2):</p> <p>Cạnh $(0,1)$ không tạo chu trình khi thêm vào T, vậy: $T = \{(1,3); (0,1)\}$</p> $\text{lstEdge} = \{(0,3); (2,3); (1,2); \}$ <p>Bước 3 (lần 2): T chưa đủ nên tiếp tục.</p>



3 Tổ chức lớp

Lớp **EDGE** thể hiện một cạnh của đồ thị.

```
class EDGE{
private:
    int v1; // Đỉnh thứ nhất
    int v2; // Đỉnh thứ 2

    char weight; // Trọng số của cạnh
};
```

Bổ sung thêm một số thuộc tính và phương thức cho lớp Graph

```
class Graph
{
private:
    Matrix _matran; // Ma trận kề của đồ thị
    vector<EDGE> _dsCanh; // Danh sách cạnh
public:

    //...

    // Tạo danh sách cạnh từ ma trận kề
    void TaoDanhSachCanh();
    // Sắp xếp danh sách cạnh
    void SapXepDanhSachCanh();
    // Kiểm tra cạnh thứ i trong danh sách cạnh
    // có tạo chu trình với các cạnh trong T không
    bool KiemTraChuTrinh(int i);
```

```
// Thuật toán Kruskal để tìm cây khung nhỏ nhất
// SpanningTree là class cho cây khung (xem bài HDTH trước)
SpanningTree ChayKruskal();
};
```

Ngoài việc biểu diễn bằng **ma trận kề**, một đồ thị còn có thể biểu diễn bằng **danh sách cạnh**. Lưu ý rằng, cây khung là một dạng đồ thị đặc biệt. Nên ta có thể biểu diễn cây khung dưới dạng một đồ thị với danh sách cạnh.

Hàm **KiemTraChuTrinh(int i)**:

- Để kiểm tra chu trình, sinh viên cần khởi tạo trước danh sách nhãn labels[], cho từng đỉnh. Gán nhãn của các đỉnh bằng chỉ mục của đỉnh.
- Việc kiểm tra tiến hành như sau:
Xem giá trị nhãn của 2 đỉnh của cạnh cần kiểm tra.
 - Nếu có cùng nhãn → tạo thành chu trình.
 - Ngược lại → Không tạo thành chu trình. Khi đó, tìm nhãn nhỏ hơn (a) và nhãn nhỏ hơn (b). Cập nhật cho toàn bộ các đỉnh có nhãn là b bằng a.

Hàm tìm cây khung là một phương thức của lớp Graph.

```
SpanningTree ChayKruskal()
{
    SpanningTree sT // cây khung cần tìm
    // Khởi tạo danh sách cạnh.
    TaoDanhSachCanh()

    // Sắp xếp các cạnh theo thứ tự tăng dần.
    SapXepDanhSachCanh()

    // Khởi tạo nhãn cho từng đỉnh
    int labels[100];
    for (int i = 0....)
        labels[i] = i;

    while (số cạnh của cây khung sT chưa đủ)
    {
        // chọn cạnh e bé nhất chưa xét
        if (eMinIndex < nEdgeCount )
        {
            //Kiểm tra xem cạnh này có tạo thành chu trình khi thêm vào không
            if (KiemTraChuTrinh(eMinIndex) == false)
            {
                // thêm cạnh có chỉ mục eMinIndex vào cây khung
                sT.AddEdge();
                // Cập nhật labels[]
                labels[...] = ...
            }
            eMinIndex++;
        }
    }
}
```

```
    }  
    else // Số cạnh của cây khung chưa đủ nhưng đã xét hết các cạnh của đồ thị  
    {  
        //Dừng thuật toán  
    }  
}  
}
```