Cấu trúc dữ liệu và giải thuật

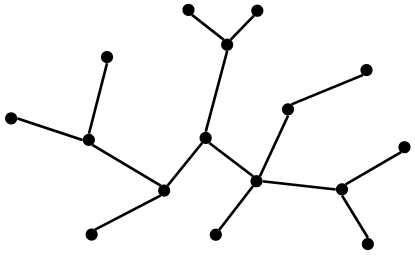
TS. Phạm Tuấn Minh
Khoa Công nghệ Thông tin, Đại học Phenikaa
minh.phamtuan@phenikaa-uni.edu.vn
https://sites.google.com/site/phamtuanminh/

Chương 5: Đồ thị

- Cây bao trùm tối thiểu

Cây

Đồ thị vô hướng là cây nếu có chính xác một đường đi giữa bất kì cặp đỉnh nào.

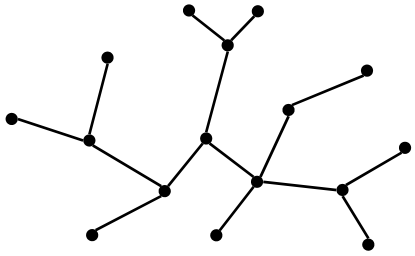


Tính chất của cây

- #E = #V – 1
- Liên thông
- Không có chu trình

- Cây có #V = 1, #E = 0

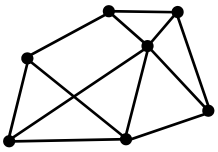
Cây có #V = 3, #E = 2



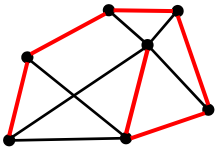
Cây bao trùm

Cây bao trùm (spanning tree) của đồ thị vô hướng liên thông (V,E) là một đồ thị con (V,E') và là một cây

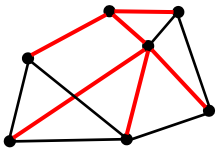
- Cùng tập đỉnh V
- E' ⊆ E
- (V, E') là cây



- Cùng tập đỉnh V
- Tập cạnh lớn nhất không tạo thành chu trình

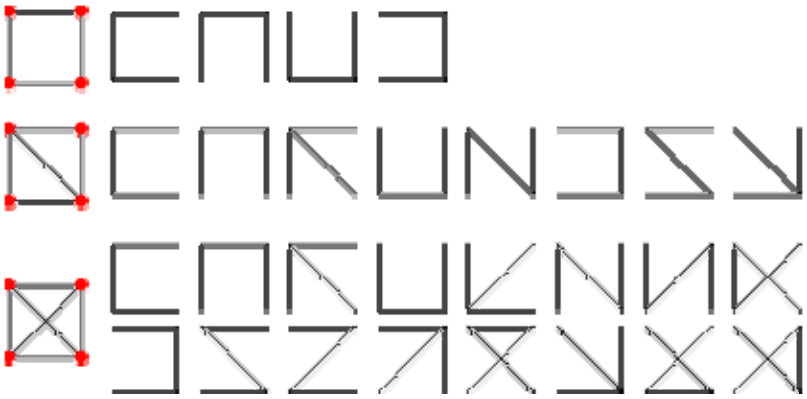


- Cùng tập đỉnh V
- Tập cạnh nhỏ nhất kết nối tất cả các đỉnh



Ba định nghĩa tương đương

Ví dụ cây bao trùm

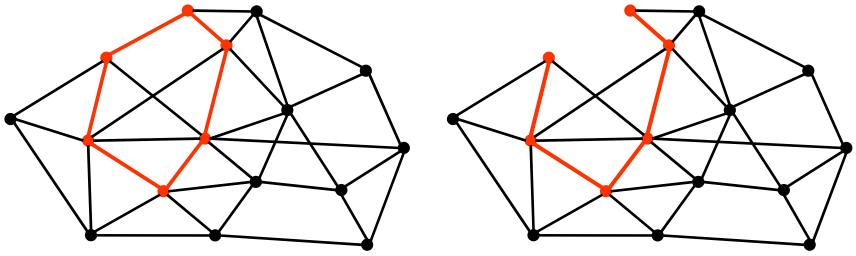


6

Tìm cây bao trùm: Phương pháp bớt

- Bắt đầu với toàn bộ đồ thị (liên thông)
- Trong khi có chu trình:
  Chọn một cạnh của chu trình và bỏ đi

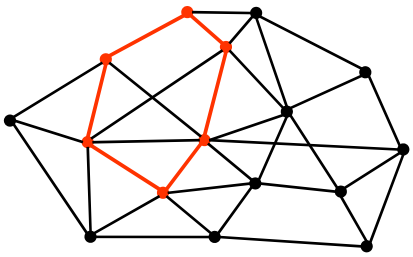
Tập cạnh lớn nhất không chứa chu trình



Kiểm tra đồ thị vô hướng có chu trình không

DFS: Thăm mọi nút tới được theo các đường chưa thăm từ u. u là đỉnh chưa thăm.

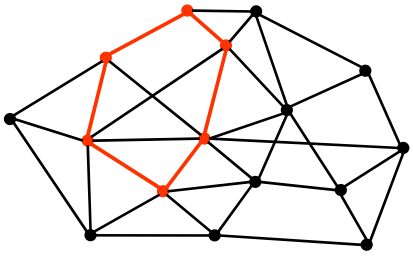
```
void DFS(int u) {
    Stack s= (u);
    while (s không rỗng) {
        u= s.pop();
        if (u chưa thăm) {
            Thăm u;
            for each edge (u, v) leaving u:
                s.push(v);
        }
    }
}
```



Kiểm tra đồ thị vô hướng có chu trình không

Trả về 1 nếu nút tới từ u có chu trình

```
int hasCycle(int u) {
    Stack s= (u);
    while (s is not empty) {
        u= s.pop();
        if (u đã thăm) return 1;
        Thăm u;
        for each edge (u, v) leaving u {
            s.push(v);
        }
    }
   return 0;
}
```

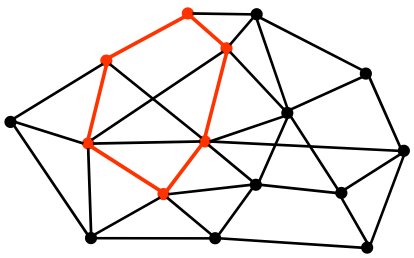


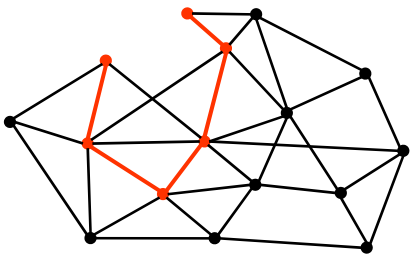
Tìm cây bao trùm: Phương pháp bớt

- Bắt đầu với toàn bộ đồ thị (liên thông)
- Trong khi có chu trình:
  Chọn một cạnh của chu trình và bỏ đi

Tập cạnh lớn nhất không chứa chu trình

Bỏ e – (n-1) cạnh
Đồ thị dày: Bỏ O(n*n) cạnh =>
Không hiệu quả.



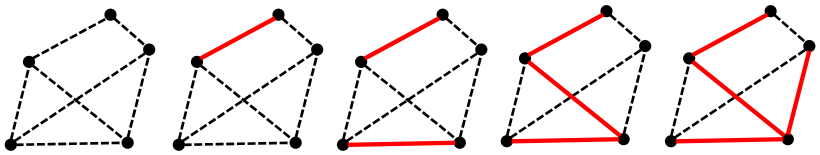


Tìm cây bao trùm: Phương pháp thêm

- Khởi tạo với tập cạnh rỗng
- Trong khi đồ thị chưa liên thông:
  Chọn một cạnh nối 2 thành phần liên thông và thêm vào tập cạnh

Tập cạnh nhỏ nhất kết nối mọi đỉnh

Thêm n-1 cạnh



Cây bao trùm tối thiểu

- Giả sử cạnh có trọng số (> 0)
- Tìm cây bao trùm có chi phí (tổng trọng số các cạnh của cây) tối thiểu => cây bao trùm tối thiểu (minimum spanning tree)

- Đồ thị có thể có một hoặc nhiều cây bao trùm tối thiểu
- Ứng dụng: định tuyến trong mạng, video streaming,...

Tìm cây bao trùm tối thiểu

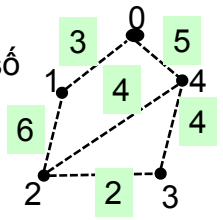
Dựa trên phương pháp thêm:

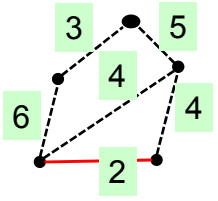
- Kruskal: Thêm một cạnh với trọng số nhỏ nhất mà không tạo thành chu trình (có thể tạo thành rừng).
- Prim: Thêm một cạnh có trọng số nhỏ nhất để cạnh thêm vào (và nút thuộc cạnh đó) tạo thành một cây.

Giải thuật Kruskal

Thêm một cạnh (không tạo thành chu trình) có trọng số nhỏ nhất.

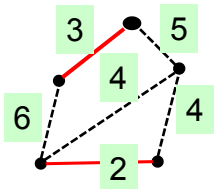
cạnh trọng số 2



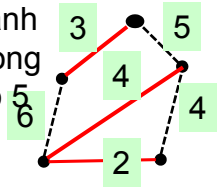


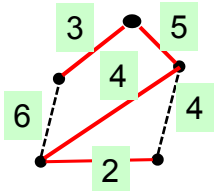
cạnh trọng số 3

một cạnh trọng số 4



cạnh trọng số 5



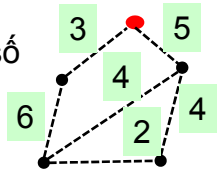


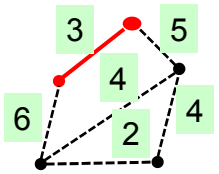
Các cạnh thêm vào không tạo thành cây (cho tới khi kết thúc

Giải thuật Prim

Thêm một cạnh (không tạo thành chu trình) với trọng số nhỏ nhất và cạnh này nối với nút đã thêm.

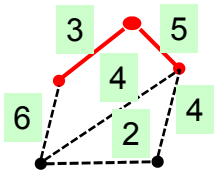
cạnh trọng số 3



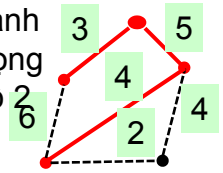


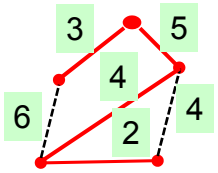
cạnh trọng số 5

một cạnh trọng số 4



cạnh trọng số 2

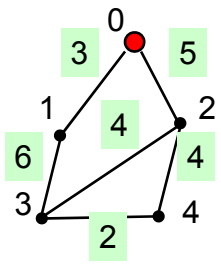




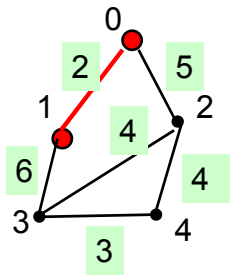
So sánh Kruscal và Prim

Prim yêu cầu các nút đã thêm luôn liên thông, trong khi Kruskal không yêu cầu.
Nếu trọng số của tất cả các cạnh là khác nhau, giải thuật Kruskal và Prim cho ra cùng kết quả.

Prim chọn (0, 1)
Kruskal chọn (3, 4)



Prim chọn (0, 2)
Kruskal chọn (3, 4)



Cài đặt giải thuật Kruskal

- 1) Sắp xếp các cạnh tăng dần theo trọng số cạnh
- 2) Duyệt các cạnh theo dãy cạnh đã sắp xếp. Nếu cạnh không tạo thành chu trình thì thêm cạnh đó vào cây bao trùm.
- 3) Lặp tới khi duyệt hết các cạnh hoặc số cạnh trên cây bao trùm = số đỉnh - 1.

1-17

Cài đặt giải thuật Kruskal

```
typedef struct _edge {
        int u, v, w;
} Edge;
typedef struct _edge_list {
        Edge data[MAX];
        int n;
} EdgeList;

EdgeList e;
EdgeList st;

int findCluster(int cluster[], int v) {
    return cluster[v];
}

void applyUnion(int cluster[], int c1, int c2) {
    int i;
    for (i = 0; i < n; i++)
        if (cluster[i] == c2) cluster[i] = c1;
}
```

```
void kruskal() {
        int cluster[MAX], i, j, c1, c2;
        bubbleSort(e);
        for (i = 0; i < n; i++)
            cluster[i] = i;

        st.n = 0;
        for (i = 0; i < e.n; i++) {
            c1 = findCluster(cluster, e.data[i].u);
            c2 = findCluster(cluster, e.data[i].v);
            if (c1 != c2) {
                st.data[st.n] = e.data[i];
                st.n++;
                applyUnion(cluster, c1, c2);
            }
        }
}
```

1-18

Cài đặt giải thuật Prim

- 1) Khởi tạo cây là 1 đỉnh bất kì
- 2) Tìm tất cả các cạnh nối cây với các đỉnh còn lại, thêm cạnh có trọng số nhỏ nhất vào cây
- 3) Lặp bước 2 tới khi thêm tất cả các đỉnh vào cây

1-19

Cài đặt giải thuật Prim

```
void prim(int G[MAX][MAX], int n) {
        int ne = 0, i, cost = 0, s[MAX], i1, j1;
        for (i = 0; i < n; i++)  s[i] = 0;
        s[0] = 1;
        while (ne < n - 1) {
            int min = 9999;
            for (int i = 0; i < n; i++)
                if (s[i] == 1)
                    for (int j = 0; j < n; j++)
                        if (s[j] == 0 && G[i][j] > 0)
                            if (min > G[i][j]) {
                                min = G[i][j];
                                i1 = i;  j1 = j;
                            }
            printf("%d - %d : %d\n", i1, j1, G[i1][j1]);
            s[j1] = 1;
            ne++;
            cost = cost + G[i1][j1];
        }
}
```

1-20

Cấu trúc dữ liệu và giải thuật

- Nội dung bài giảng được biên soạn bởi TS. Phạm Tuấn Minh.

1-21