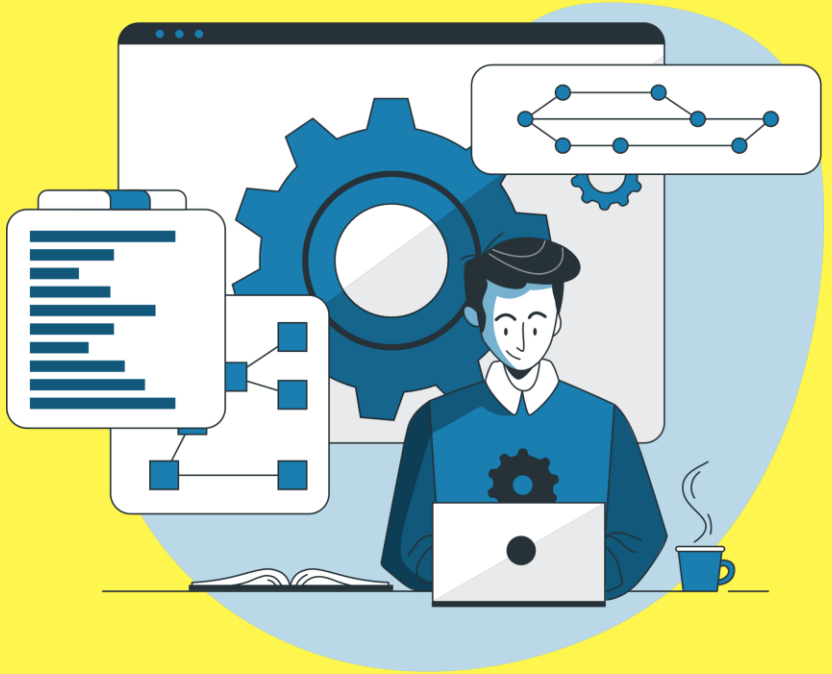
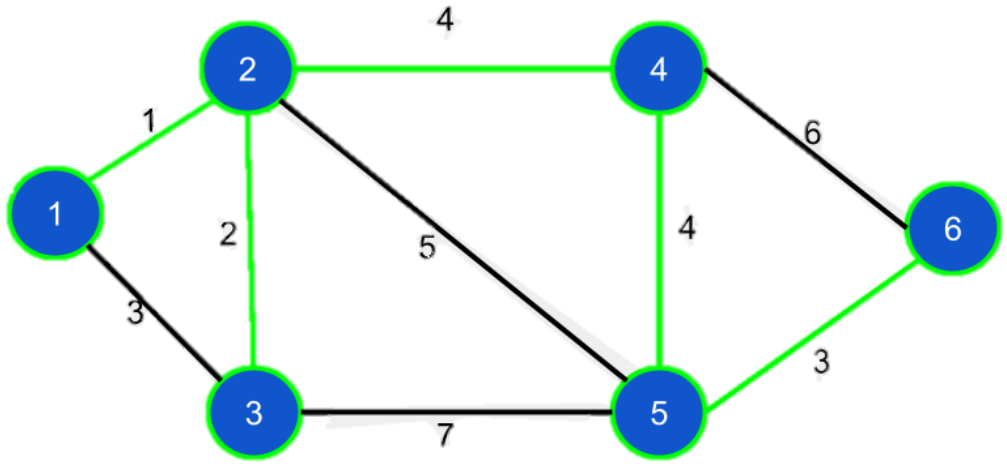
CÂY KHUNG CỰC TIỂU



Giới thiệu:

Bài toán cây khung cực tiểu (Minimum spanning tree) trên đồ thị vô hướng liên thông yêu cầu bạn tìm cây khung hay cây bao trùm có tổng trọng số các cạnh là nhỏ nhất.

Cây khung sẽ kết nối toàn bộ các đỉnh trên đồ thị và không tồn tại chu trình trên cây khung.



Cây khung cực tiểu: (1,2), (2,3), (2,4), (5,6), (4,5)

1. Thuật toán Kruskal:

Tư tưởng của thuật toán Kruskal đó là ở mỗi bước bạn sẽ đưa thêm 1 cạnh có trọng số nhỏ nhất (chưa thuộc cây khung) vào cây khung nếu nó không tạo chu trình. Để code được thuật toán Kruskal các bạn cần biết cấu trúc dữ liệu DSU.

Thuật toán sẽ kết thúc nếu tìm đủ N - 1 cạnh hoặc không còn cạnh nào chưa nằm trong cây khung.

1. Thuật toán Kruskal:

Mã giả:

```
Kruskal(){
   //Khởi tạo
   MST = ∅; // cây khung ban đầu rỗng
   d = 0; // độ dài cây khung
   //Sắp xếp
   <Sắp xếp các cạnh theo trọng số tăng dần>
   //Lặp
   while(|MST| < n - 1 && E != ∅){
      e = <Cạnh có trọng số nhỏ nhất>
      E = E\{e}; // Loại e khỏi tập cạnh
      if(MST ∪ {e} không tạo chu trình){
         MST = MST ∪ {e}; //Thêm e vào cây khung
         d = d + d(e); // Cập nhật độ dài cây khung
      }
   }
   //Kết quả
   if(|MST| < n - 1)
      <Đồ thị không liên thông>;
   else
      return{MST, d};
}
```

1. Thuật toán Kruskal:

Cài đặt thuật toán

```
struct edge{
    int x, y, w;
};

int maxn = 1005;
int n, m; // đỉnh, cạnh
vector<edge> E; // tập cạnh
int parent[maxn], sz[maxn];

void nhap(){
    cin >> n >> m;
    for(int i = 0; i < m; i++){
        int x, y, w; cin >> x >> y >> w;
        edge e{x, y, w};
        E.push_back(e);
    }
}
```

```
void init(){
    for(int i = 1; i <= n; i++){
        parent[i] = i;
        sz[i] = 1;
    }
}
int Find(int u){
    if(u == parent[u]) return u;
    return parent[u] = Find(parent[u]);
}
bool Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return false;
    if(sz[u] < sz[v]) swap(u, v);
    parent[v] = u;
    sz[u] += sz[v];
    return true;
}
```

1. Thuật toán Kruskal:

Cài đặt thuật toán

```
void Kruskal(){
    vector<edge> MST;
    int d = 0;
    sort(E.begin(), E.end(), [](edge x, edge y)->bool{
        return x.w < y.w;
    });
    for(edge e : E){
        if(MST.size() == n - 1) break;
        if(Union(e.x, e.y)){
            MST.push_back(e);
            d += e.w;
        }
    }
    if(MST.size() < n - 1){
        cout << "Do thi khong lien thong !\n";
    }
    else{
        cout << d << endl;
        for(edge e : MST){
            cout << e.x << ' ' << e.y << ' ' << e.w << endl;
        }
    }
}
```

```
int main(){
    nhap();
    init();
    Kruskal();
}
```

INPUT

```
6 9
1 2 1
2 3 2
1 3 3
2 4 4
2 5 5
3 5 7
4 6 6
5 6 3
4 5 4
```

OUTPUT

```
14
1 2 1
2 3 2
5 6 3
2 4 4
4 5 4
```

2. Thuật toán Prim:

- Tư tưởng của thuật toán Prim đó là duy trì 2 tập đỉnh V : tập đỉnh ban đầu và MST là tập đỉnh cây khung. Thuật toán Prim sẽ bắt đầu với một đỉnh bất kỳ của đồ thị. Ban đầu MST = {s}, s là đỉnh bắt đầu của thuật toán, V = V\{s}
- Mỗi bước chọn ra 1 cạnh có trọng số nhỏ nhất mà 1 đỉnh của cạnh này thuộc tập V và đỉnh còn lại thuộc tập MST sau đó đưa đỉnh này vào cây khung. Cập nhật V và MST. Thuật toán kết thúc khi cây khung đủ n - 1 cạnh hoặc tập V rỗng.

2. Thuật toán Prim:

Mã giả:

```
Prim(s){
    //Khởi tạo
    V = V\{s}; //Loại s khỏi tập V
    MST = {s}; // Khởi tạo tập MST là đỉnh s
    d = 0; // chiều dài cây khung
    T = {}; //Cây khung
    //Lặp
    while(V != ∅){
        e = (u, v); // cạnh có độ dài nhỏ nhất mà u
thuộc V, v thuộc MST
        MST = MST ∪ {u}; // Thêm u vào tập MST
        V = V\{u}; // Loại u khỏi tập V
        T = T ∪ {e}; // Thêm cạnh e vào cây khung
        d += d(e); // cập nhật độ dài cây khung
    }
    //Kết quả
    if(|T| < n - 1)
        <Đồ thị không liên thông>;
    else
        return {T, d};
}
```

2. Thuật toán Prim:

Để có thể nhanh chóng tìm ra cạnh có độ dài ngắn nhất để nạp vào cây khung, ta sử dụng hàng đợi ưu tiên. Hàng đợi ưu tiên này lưu pair, trong đó first lưu trọng số và second lưu đỉnh.

Để kiểm tra đỉnh thuộc tập V hay MST ta dùng mảng đánh dấu taken[], trong đó taken[u] = true nếu đỉnh u thuộc tập MST, ngược lại thuộc tập V

2. Thuật toán Prim:

Cài đặt thuật toán

```
typedef pair<int, int> ii;
const int maxn = 1005;
int n, m;
vector<ii> adj[maxn];
bool taken[maxn];

void nhap(){
    cin >> n >> m;
    for(int i = 0; i < m; i++){
        int x, y, w; cin >> x >> y >> w;
        adj[x].push_back({y, w});
        adj[y].push_back({x, w});
    }
    memset(taken, false, sizeof(taken));
}
```

2. Thuật toán Prim:

Cài đặt thuật toán

```
void Prim(int s){
    priority_queue<ii, vector<ii>, greater<ii>> Q;
    taken[s] = true;
    int d = 0;
    for(ii x : adj[s]){
        if(!taken[x.first]){
            Q.push({x.second, x.first});
        }
    }
    while(!Q.empty()){
        ii top = Q.top(); Q.pop();
        int u = top.second, w = top.first;
        if(!taken[u]){
            d += w;
            taken[u] = true;
            for(ii x : adj[u]){
                if(!taken[x.first]){
                    Q.push({x.second, x.first});
                }
            }
        }
    }
    cout << d << endl;
}
```

```
int main(){
    nhap();
    Prim(1);
}
```

INPUT

```
6 9
1 2 1
2 3 2
1 3 3
2 4 4
2 5 5
3 5 7
4 6 6
5 6 3
4 5 4
```

OUTPUT

```
14
```