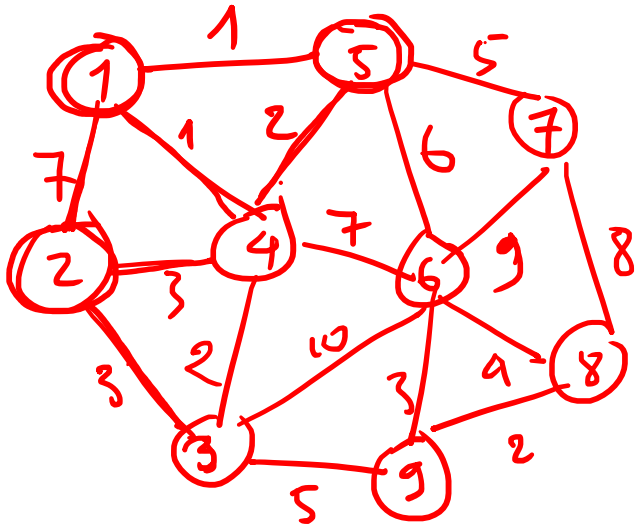
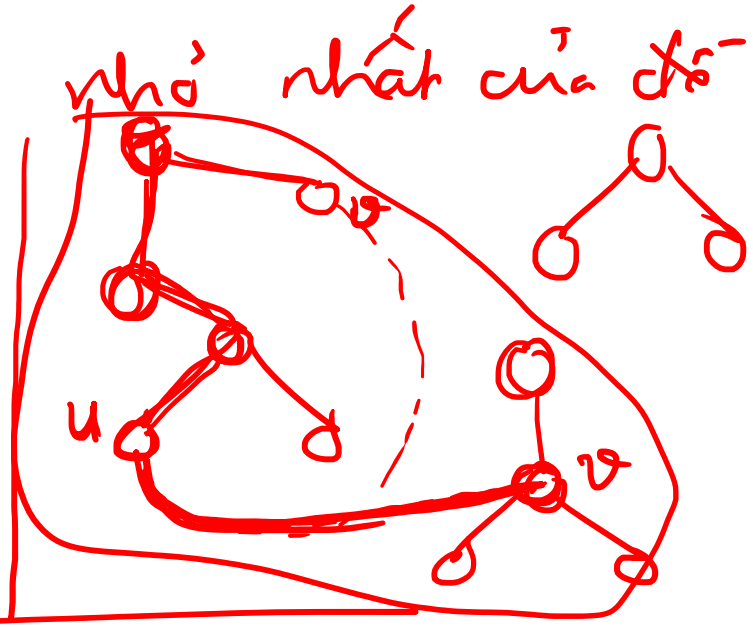
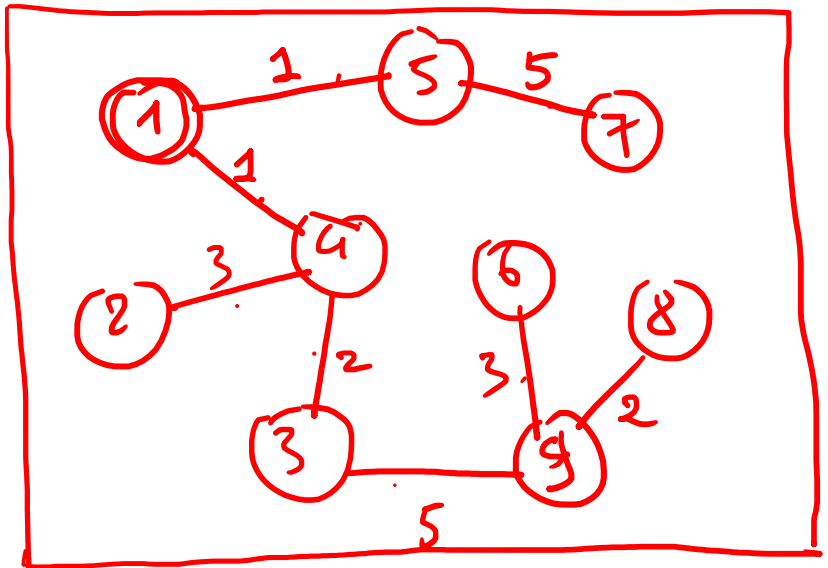
Thuật toán Kruskal giải BT cây khung nhỏ nhất của đồ thị (Minimum Spanning Tree - MST)



Chọn ra tập con các cạnh sao cho kết nối liên thông các đỉnh với tổng trọng số nhỏ nhất






```
Kruskal( )
[ L = sắp xếp các cạnh của G (theo chiều giảm trọng số)
  T = {};
  for (u,v) ∈ L do
    if [T ∪ {(u,v)} không tạo chu trình] then
      T = T ∪ {(u,v)};
      if |T| = |V| - 1 then Break;
  return T;
```

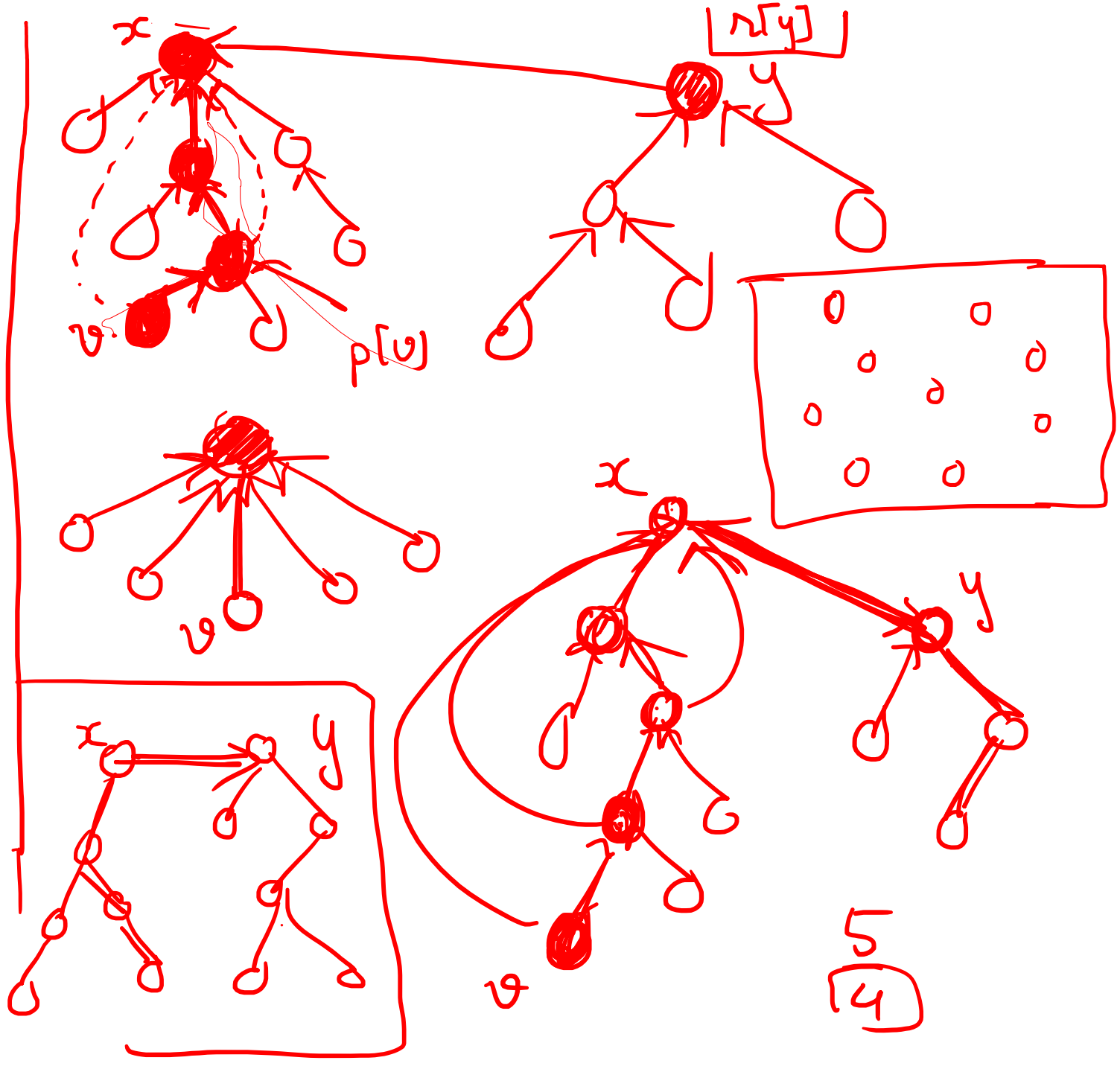
Disjoint Set:

Cấu trúc lưu trữ các tập không giao nhau

- Find(v): tìm gốc của cây chứa v
- Unify(x, y): hợp nhất cây gốc x và y làm

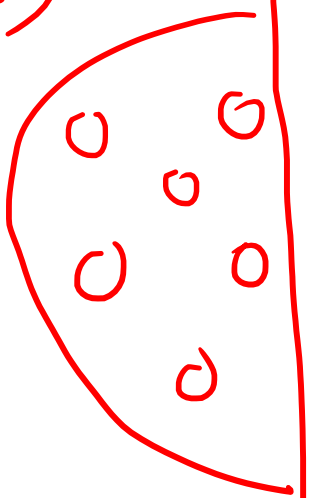
CTDL:

- p[v]: cha của v
- r[v]: độ cao của v



```
Find(v) {
    if p[v] ≠ v then
        p[v] = Find(p[v])
    return p[v];
}
```

```
Unify(x, y) {
    if r[x] > r[y] then
        p[y] = x
    else [ p[x] = y;
           if r[x] = r[y] then
               r[y] = r[y] + 1
}
```



O(1)

```
KruskalDisjointSet( G=(V,E) )
[ L = Sort(E); // Sắp xếp cạnh
  (Init)
  for v ∈ V do [ p[v] = v;
                 r[v] = 1;
  T = {};
  for (u,v) ∈ L do
      [ ru = Find(u); rv = Find(v);
        if ru ≠ rv then
            [ T = T ∪ {(u,v)};
              Unify(ru, rv);
              if |T| = |V| - 1 then Break;
  return T;
```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define N 100001
struct Edge{
  int u,v,w;
};

Edge E[N];
int n,m;
// data structures for Disjoint Set
int p[N], r[N];
```

```cpp
int Find(int v){
  if(v != p[v]) p[v] = Find(p[v]);
  return p[v];
}
void Unify(int x, int y){// x and y are the root
  if(r[x] > r[y]){
    p[y] = x;
  }else{
    p[x] = y;
    if(r[x] == r[y]) r[y]++;
  }
}
bool cmp(Edge& e1, Edge& e2){
  return e1.w < e2.w;
}
```

```
int Kruskal(){
  sort(E,E+m,cmp);
  for(int v = 1; v <= n; v++){     p[v] = v; r[v] = 1; }
  vector<Edge> T;
  int W = 0;
  for(int i = 0; i < m; i++){
    int u = E[i].u; int v = E[i].v; int w = E[i].w;
    int ru = Find(u); int rv = Find(v);
    if(ru != rv){// edge (u,v) can be added to T
      T.push_back(E[i]);     W += w;    Unify(ru,rv);
      if(T.size() == n-1) break;
    }
  }
  return W;
}
```

```
void input(){
  scanf("%d%d",&n,&m);
  for(int k = 0; k < m; k++){
    int u,v,w;
    scanf("%d%d%d",&u,&v,&w);
    E[k].u = u; E[k].v = v; E[k].w = w;
  }
}
int main(){
  input();
  int r void input()();
  cout << res;
  return 0;
}
```

Thuật toán Dijkstra tìm đường đi ngắn nhất từ s đến t trên đồ thị trọng số KHÔNG âm

Duy trì đường đi cận trên của ĐĐNN từ s đến v { d[v]: độ dài cực đại của đường đi cận trên
p[v]: đỉnh trước v trên đường đi cận trên

$w(u,v) = +\infty$, if $(u,v) \notin E$

- Nếu G là đồ thị thưa: F → priority queue
  Time: $O(|E| \log |V|)$
- Nếu G là đồ thị đầy (đầy đủ)
  → F set: $O(|V|^2)$

```
Dijkstra(G=(V,E))
//Init
for v ∈ V do { d[v]=w(s,v);
               p[v]=s;
F = V; // tập các đỉnh mà chưa tìm được ĐĐNN
while F not empty do
  u = select(F) có d[u] → min;
  if u=t then return;
  F = F \ {u};
  // update d[v], v∈F.
  for v ∈ A[u] do
    if v ∈ F then
      if d[v] > d[u]+w(u,v) then
        d[v] = d[u]+w(u,v);
        p[v]=u;
```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define N 100001
#define pii pair<int, int>
struct Arc{
  int node;
  int w;
  Arc(int _node, int _w){
    node = _node; w = _w;
  }
};
const int INF = 1e9;
vector<Arc> A[N];
int n,m,s,t;
int d[N],p[N];
```

```
int dijkstra(){
  for(int v = 1; v <= n; v++){ d[v] = INF; p[v] = -1; }
  priority_queue<pii, vector<pii>, greater<pii> > F;
  d[s] = 0; F.push(make_pair(0,s));
  while(!F.empty()){
    pii e = F.top(); F.pop(); int u = e.second;
    if(u == t) return d[t];
    for(int i = 0; i < A[u].size(); i++){
      int v = A[u][i].node; int w = A[u][i].w;
      if(d[v] > d[u] + w){  d[v] = d[u] + w; p[v] = u;
        F.push(make_pair(d[v],v));
      }
    }
  }
  return INF;// not found
```

```
void input(){
  scanf("%d%d",&n,&m);
  for(int k = 1; k <= m; k++){
    int u, v, w; scanf("%d%d%d",&u,&v,&w);
    A[u]int input::urc(v,w));
  }
  scanf("%d%d",&s,&t);
}
int main(){
  input();
  int res = dijkstra();
  cout << res;
  return 0;
}
```