最小生成树

最小生成树是针对的无向图。给定一个无向图，若它的某个子图中的任意两个顶点都互相连通并且是一棵树，那么这棵树就叫生成树，若图的边带有权值那么使得权值最小的生成树叫最小生成树。

Prim算法

Prim算法和Dijkstra算法十分相似，不同之处在于Dijkstra算法处理的是没有圈的有向图且要指定源点，以及不是求一点到另一点的最短距离，体现在跟新最小权值处(Dijkstra用d[],Prim使用mincost[])。相同之处在于，两者都是通过选择最小权值的边，将其顶点加入已选择的集合中。

```
#include <iostream>
#include <algorithm>
#define maxsize 501
#define INF 999999

using namespace std;
int cost[maxsize][maxsize], mincost[maxsize], V, E;
bool used[maxsize];

void init() {
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            if (i != j)
                cost[i][j] = INF;
}

int Prim() {
    fill(mincost, mincost + V, INF);
    mincost[0] = 0;
    int res = 0;
    while (true) {
        int v = -1;
        for (int i = 0; i < V; i++)
            if (!used[i] && (v == -1 || mincost[i] < mincost[v])) v = i;
        if (v == -1) break;
        used[v] = true;
        res += mincost[v];
        for (int i = 0; i < V; i++)
            mincost[i] = min(mincost[i],  cost[v][i]);
    }
    return res;
}

int main() {
    cin >> V >> E;
    init();
    for (int i = 0; i < E; i++) {
        int f, t, c;
        cin >> f >> t >> c;
        cost[f - 1][t - 1] = c;
        cost[t - 1][f - 1] = c;
    }
    cout << Prim() << endl;
```

```
    return 0;
}
```

测试数据：

> input:
>
> 7 9
> 1 3 1
> 2 3 2
> 3 4 3
> 2 5 10
> 5 6 5
> 3 6 7
> 4 6 1
> 4 7 5
> 6 7 8
>
> output:
>
> 17

队列优化：

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#define maxsize 501
#define INF 999999

using namespace std;
typedef pair<int, int> P;//first:权值;second:顶点
struct edge { int to, cost; public: edge(int t, int c) { to = t, cost = c; } };
int mincost[maxsize], V, E;
bool used[maxsize];
vector<edge> es[maxsize * 2];

int Prim() {
    fill(mincost, mincost + V, INF);
    mincost[0] = 0;
    int res = 0;
    priority_queue<P, vector<P>, greater<P>> queue;
    queue.push(P(0, 0));
    while (!queue.empty()) {
        P p = queue.top(); queue.pop();
        if (used[p.second]) continue;
        res += p.first;
        used[p.second] = true;
        for (int i = 0; i < es[p.second].size(); i++) {
            edge e = es[p.second][i];
            if (!used[e.to] && mincost[e.to] > e.cost) {
                mincost[e.to] = e.cost;
                queue.push(P(e.cost, e.to));
            }
        }
    }
```

```
    return res;
}

int main() {
    cin >> V >> E;
    for (int i = 0; i < E; i++) {
        int f, t, c;
        cin >> f >> t >> c;
        es[f - 1].push_back(edge(t - 1, c));
        es[t - 1].push_back(edge(f - 1, c));
    }
    cout << Prim() << endl;
    return 0;
}
```

Kruskal算法

将边的权值从小到大进行排序，如果两点没有联通，则将两点进行联通，直到生成树。借助并查集高效地判断两点是否属于同一连通分量。

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#define maxsize 501
#define INF 999999

using namespace std;
typedef pair<int, int> P;//first:权值;second:顶点
struct edge { int from, to, cost; };
int V, E;
edge es[maxsize];
//UnionFind
int par[maxsize], prank[maxsize];
void init(int n) {
    for (int i = 0; i < n; i++) {
        par[i] = i;
        prank[i] = 0;
    }
}
int find(int x) {
    if (par[x] == x)
        return x;
    return par[x] = find(par[x]);
}
void unite(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y)return;
    if (prank[x] < prank[y])
        par[x] = y;
    else {
        par[x] = y;
        if (prank[x] == prank[y])
            prank[x]++;
    }
}
```

```
bool same(int x, int y) {
    return find(x) == find(y);
}
bool cmp(edge a, edge b) {
    return a.cost < b.cost;
}

int Kruskal() {
    int res = 0;
    sort(es, es + V, cmp);
    init(V);
    for (int i = 0; i < E; i++) {
        if (!same(es[i].from, es[i].to)) {
            unite(es[i].from, es[i].to);
            res += es[i].cost;
        }
    }
    return res;
}

int main() {
    cin >> V >> E;
    for (int i = 0; i < E; i++) {
        int f, t, c;
        cin >> f >> t >> c;
        es[i].from = f - 1, es[i].to = t - 1, es[i].cost = c;
    }
    cout << Kruskal() << endl;
    return 0;
}
```