

19-课堂笔记-0708-dij 优化+并查集+最小生成树

一、dijkstra 优化 (优先队列+邻接表)

priority_queue 优化 dijkstra 最短路
时间复杂度 $O((m+n)\log n)$

```
priority_queue< pair<int, int> > _pqque; //first为dis[]反数, second为节点编号

for(int i = 1; i <= _n; i++) {
    _minx = 1000;
    _k = 0; //中转点
    for(int j = 1; j <= _n; j++) {
        //蓝点中找到源点v的最小距离点
        if(!_b[j] && _dis[j] < _minx) {
            _minx = _dis[j];
            _k = j;
        }
    }
    //中转点蓝变白
    _b[_k] = true;
    //通过中转点更新其余蓝点
    for(int j = 1; j <= _n; j++) {
        if(_dis[_k] + w[_k][j] < _dis[j]) {
            _dis[j] = _dis[_k] + w[_k][j];
            _pre[j] = _k; //记录到j点的中转点
        }
    }
}
```

```
1 while(!_pqque.empty()) {
2     int x = _pqque.top().second; //取出最小值节点编号 |
3     _pqque.pop(); //最小值节点出队
4     if(!_b[x]) {
5         _b[x] = true;
6         //扫描所有出边, 进行更新
7         for(int i = _head[x]; i != 0; i = _e[i].next) {
8             int y = _e[i].to;
9             int z = _e[i].w;
10            if(_dis[x] + z < _dis[y]) {
11                _dis[y] = _dis[x] + z;
12                _pqque.push(make_pair(-_dis[y], y));
13            }
14        }
15    }
16 }
```

模板:

```
1 #include <iostream>
2 #include <queue>
3 #include <cstring>
4 using namespace std;
5 const int N = 100010;
6 const int M = 200010;
7 priority_queue< pair<int, int> > pqque;
8 int head[N];
9 int numEdge;
10 int dis[N];
11 bool b[N];
12 int n, m;
13 struct edge {
14     int next;
15     int to;
16     int w;
17 } e[2 * M];
18
19 void AddEdge(int from, int to, int w) {
20     numEdge++;
21     e[numEdge].next = head[from];
22     e[numEdge].to = to;
23     e[numEdge].w = w;
24     head[from] = numEdge;
25 }
26
27 void Dijkstra(int sta) {
28     memset(dis, 0x3f, sizeof(dis));
29     dis[sta] = 0;
30     pqque.push(make_pair(0, sta));
31     while(!pqque.empty()) {
32         int x = pqque.top().second; //取最小值的编号
33         pqque.pop();
34         if(!_b[x]) {
35             _b[x] = true;
36             for(int i = head[x]; i != 0; i = e[i].next) {
37                 int y = e[i].to;
38                 int z = e[i].w;
39                 if(dis[x] + z < dis[y]) {
40                     dis[y] = dis[x] + z;
41                     pqque.push(make_pair(-1 * dis[y], y)); //小顶堆
42                 }
43             }
44         }
45     }
46 }
47
48 int main() {
49     int u, v, w;
50     cin >> n >> m;
51     for(int i = 1; i <= m; i++) {
52         cin >> u >> v >> w;
53         AddEdge(u, v, w);
54         AddEdge(v, u, w);
55     }
56     int start = 1;
57     Dijkstra(start);
58     return 0;
59 }
```

练习 1: 1359

未优化做法:

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  int g[210][210];
5  int dis[210];
6  int n, minx, k;
7  bool b[210];
8  void Dijkstra(int sta) {
9      memset(dis, 0x3f, sizeof(dis));
10     dis[sta] = 0;
11     for(int i = 1; i <= n; i++) {
12         minx = 1e9;
13         k = 0; // 中转点
14         for(int j = 1; j <= n; j++) {
15             // 蓝点中找到源点v的最小距离点
16             if(!b[j] && dis[j] < minx) {
17                 minx = dis[j];
18                 k = j;
19             }
20         }
21         // 中转点k蓝变白
22         b[k] = true;
23         // 通过中转点更新其余蓝点
24         for(int j = 1; j <= n; j++) {
25             if(dis[k] + g[k][j] < dis[j]) {
26                 dis[j] = dis[k] + g[k][j];
27             }
28         }
29     }
30 }
31
32 int main(){
33     memset(g, 0x3f, sizeof(g));
34     cin >> n;
35     for(int i = 1; i <= n-1; i++){
36         for(int j = i + 1; j <= n; j++){
37             cin >> g[i][j];
38         }
39     }
40     int start = 1;
41     Dijkstra(start);
42     cout << dis[n];
43     return 0;
44 }

```

邻接表+优先队列优化：

```

1 #include <iostream>
2 #include <queue>
3 #include <cstring>
4 using namespace std;
5 const int N = 100010;
6 const int M = 200010;
7 priority_queue< pair<int, int> > pq;
8 int head[N];
9 int numEdge;
10 int dis[N];
11 bool b[N];
12 int n, m;
13 struct edge{
14     int next;
15     int to;
16     int w;
17 }e[2 * M];
18
19 void AddEdge(int from, int to, int w){
20     numEdge++;
21     e[numEdge].next = head[from];
22     e[numEdge].to = to;
23     e[numEdge].w = w;
24     head[from] = numEdge;
25 }
26
27
28 void Dijkstra(int sta) {
29     memset(dis, 0x3f, sizeof(dis));
30     dis[sta] = 0;
31     pq.push(make_pair(0, sta));
32     while(!pq.empty()){
33         int x = pq.top().second; //取最小值的编号
34         pq.pop();
35         if(!b[x]){
36             b[x] = true;
37             for(int i = head[x]; i != 0; i = e[i].next){
38                 int y = e[i].to;
39                 int z = e[i].w;
40                 if(dis[x] + z < dis[y]){
41                     dis[y] = dis[x] + z;
42                     pq.push(make_pair(-1 * dis[y], y)); //小顶堆
43                 }
44             }
45         }
46     }
47 }
48
49 int main(){
50     int w;
51     cin >> n;
52     for(int i = 1; i <= n-1; i++){
53         for(int j = i + 1; j <= n; j++){
54             cin >> w;
55             AddEdge(i, j, w);
56         }
57     }
58     int start = 1;
59     Dijkstra(start);
60     cout << dis[n];
61     return 0;
62 }

```

练习 2: 1629

思路点拨:

- 1、求点 1 到其余所有点的最短路
- 2、求 2-n 的每个点到点 1 的最短路，解决方案：存反向图再求一遍最短路即可
- 3、求和

```

1 #include <iostream>
2 #include <queue>
3 #include <cstring>
4 using namespace std;
5 #define ll long long
6 const int N = 1010;
7 const int M = 100010;
8 bool b[N];
9 ll dis[N];
10 ll head1[N];
11 ll head2[N];
12 ll numEdge;
13 ll numEdge1;
14 ll n, m;
15 ll sum;
16 ll sta;
17 struct edge{
18     int to;
19     int w;
20     int next;
21 }e[2 * M], e1[2 * M];
22 void AddEdge(int from, int to, int w) {
23     numEdge++;
24     e[numEdge].next = head1[from];
25     e[numEdge].w = w;
26     e[numEdge].to = to;
27     head1[from] = numEdge;
28 }
29 void AddEdge1(int from, int to, int w) //反向图
30 {
31     numEdge1++;
32     e1[numEdge1].next = head2[from];
33     e1[numEdge1].w = w;
34     e1[numEdge1].to = to;
35     head2[from] = numEdge1;
36 }
37
38 void Dijkstra(int start) {
39     priority_queue< pair<ll, ll> > pq; //first为dis[]反数, second为节点编号
40     memset(b, 0, sizeof(b));
41     memset(dis, 0x3f, sizeof(dis));
42     dis[start] = 0;
43     pq.push(make_pair(0, start));
44     while(!pq.empty()) {
45         int x = pq.top().second; //取出最小值节点编号
46         pq.pop(); //最小值节点出队
47         if(!b[x]) {
48             b[x] = true;
49             //扫描所有出边, 进行更新
50             for(int i = head1[x]; i != 0; i = e[i].next) {
51                 int y = e[i].to;
52                 int z = e[i].w;
53                 if(dis[x] + z < dis[y]) {
54                     dis[y] = dis[x] + z;
55                     pq.push(make_pair(-dis[y], y));
56                 }
57             }
58         }
59     }
60 }
61
62 void Dijkstra1(int start) {
63     priority_queue< pair<ll, ll> > pq; //first为dis[]反数, second为节点编号
64     memset(b, 0, sizeof(b));
65     memset(dis, 0x3f, sizeof(dis));
66     dis[start] = 0;
67     pq.push(make_pair(0, start));
68     while(!pq.empty()) {
69         int x = pq.top().second; //取出最小值节点编号
70         pq.pop(); //最小值节点出队
71         if(!b[x]) {
72             b[x] = true;
73             //扫描所有出边, 进行更新
74             for(int i = head2[x]; i != 0; i = e1[i].next) {
75                 int y = e1[i].to;
76                 int z = e1[i].w;
77                 if(dis[x] + z < dis[y]) {
78                     dis[y] = dis[x] + z;
79                     pq.push(make_pair(-dis[y], y));
80                 }
81             }
82         }
83     }
84 }
85
86 int main() {
87     cin >> n >> m;
88     for(int i = 1; i <= m; i++) {
89         int u, v, w;
90         cin >> u >> v >> w;
91         AddEdge(u, v, w);
92         AddEdge1(v, u, w);
93     }
94     Dijkstra(1);
95     Dijkstra1(1);
96     for(int i = 2; i <= n; i++) {
97         sum += dis[i];
98     }
99     cout << sum;
100     return 0;
101 }

```

```

49 int main() {
50     int u = 0, v = 0, w = 0;
51     cin >> n >> m;
52     for(int i = 1; i <= m; i++) {
53         cin >> u >> v >> w;
54         AddEdge(u, v, w);
55         AddEdge1(v, u, w); // 反图
56     }
57     sta = 1;
58     Dijkstra(sta);
59     for(int i = 2; i <= n; i++) {
60         sum += dis[i];
61     }
62     Dijkstra1(sta);
63     for(int i = 2; i <= n; i++) {
64         sum += dis[i];
65     }
66     cout << sum;
67
68     return 0;
69 }

```

二、并查集

假如已知有 n 个人和 m 对好友关系（存于数字 r ）。如果两个人是直接或者间接的好友（即就是好友的好友...），则认为他们属于一个朋友圈，请写出程序求出这 n 个人里一共有多少个朋友圈。

例如： $n = 5$, $m = 3$, $r = \{\{1,2\},\{2,3\},\{4,5\}\}$, 表示有 5 个人，1 和 2 是朋友，2 和 3 也是朋友，4 和 5 是朋友，则 1,2,3 属于一个朋友圈，4、5 属于另一个朋友圈，结果为 2 个朋友圈。

解法 1: dfs

```

1  #include <iostream>
2  #include <queue>
3  #include <cstring>
4  using namespace std;
5  const int N = 200;
6  const int M = 10010;
7  int head[N], numEdge;
8  int n, m;
9  bool vis[N];
10 struct edge{
11     int next;
12     int to;
13     int w;
14 }e[2 * M];
15 void AddEdge(int from, int to){
16     numEdge++;
17     e[numEdge].next = head[from];
18     e[numEdge].to = to;
19     head[from] = numEdge;
20 }
21
22 void dfs(int x){
23     vis[x] = true;
24     for(int i = head[x]; i != 0; i = e[i].next){
25         int y = e[i].to;
26         if(!vis[y]){
27             dfs(y);
28         }
29     }
30 }

```

```

1  int main(){
2      int u, v = 0, cnt = 0;
3      cin >> n >> m;
4      for(int i = 1; i <= m; i++){
5          cin >> u >> v;
6          AddEdge(u, v);
7          AddEdge(v, u);
8      }
9
10     for(int i = 1; i <= n; i++){
11         if(!vis[i]){
12             dfs(i);
13             cnt++;
14         }
15     }
16     cout << cnt;
17     return 0;
18 }

```

解法二：并查集

并查集(union-find set)：一种用于分离集合操作的抽象数据类型

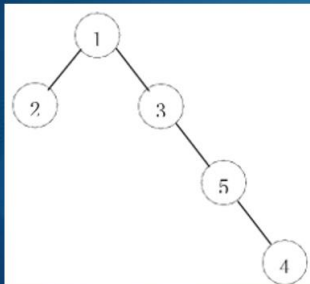
并查集作用：动态维护和处理元素之间的关系

- 找到一个元素的所属集合
- 将两个元素各自所属的集合进行合并（当给出两元素的无序对(a,b)时，能够快速的合并a和b所在的集合）
- 反复查找某个元素在哪个集合中

并查集的实现：

- 数组
- 链表
- 树

并查集支持的操作：



通过数组来控制，初始化如下

(下标)	1	2	3	4	5
FA	1	2	3	4	5

合并 1 和 2

(下标)	1	2	3	4	5
FA	1	1	3	4	5

合并 5 和 4

(下标)	1	2	3	4	5
FA	1	1	1	5	5

合并 5 和 3

(下标)	1	2	3	4	5
FA	1	1	1	5	3

FA 数组随着查询而更新

判断某个点时候有祖先，判断 $FA[i] = i$

查询过程举例：

① 若想查询 5 的最后祖先，将通过下标有几次查询过程，过程如下图

(下标)	1	2	3	4	5
FA	1	1	1	5	3

同时通过递归将最终祖先传回，更新数组如下：

(下标)	1	2	3	4	5
FA	1	1	1	5	1

共查询了两次

② 此时再查询 4 的最终祖先，将通过下标有几次查询过程，过程如下图：

(下标)	1	2	3	4	5
FA	1	1	1	5	1

同时通过递归将最终祖先传回，更新数组如下：

(下标)	1	2	3	4	5
FA	1	1	1	1	1

并查集支持的操作:

1、Make(x):建立一个新的集合，其成员只有x

2、Find(x):返回一个指向包含x集合的代表

3、Unionn(x,y):将包含x的集合和包含y的集合合并为一个新的集合

4、Judge(x, y):判断两个元素是否属于同一个集合

```
for(int i = 1; i <= _n; i++) {
    _father[i] = i;
}
```

```
int Find(int x){//实现了找祖先的过程
    if(x == fa[x]){//说明自己就是祖先
        return x;
    }
    else {
        return Find(fa[x]);
    }
}
```

```
void Unionn(int x, int y) {
    x = Find(x);
    y = Find(y);
    _father[y] = x;
}
```

```
bool Judge(int x, int y) {
    x = Find(x);
    y = Find(y);
    if(x == y) {
        return true;
    }
    return false;
}
```

模板:

```
1 #include <iostream>
2 using namespace std;
3 int fa[110];
4 int n, m, q;
5 void print(){
6     for(int i = 1; i <= n; i++){
7         cout << fa[i] << " ";
8     }
9     cout << endl;
10 }
11 int Find(int x){//实现了找祖先的过程
12     if(x == fa[x]){//说明自己就是祖先
13         return x;
14     }
15     else {
16         return Find(fa[x]);
17     }
18 }
19
20 int Find1(int x){//实现了找祖先的过程
21     if(x != fa[x]){
22         fa[x] = Find(fa[x]);
23     }
24     return fa[x];
25 }
26
```

```
1
2 int main(){
3     int x, y;
4     cin >> n >> m >> q;
5     for(int i = 1; i <= n; i++){
6         fa[i] = i;
7     }
8
9     for(int i = 1; i <= m; i++){
10         cin >> x >> y;
11         x = Find(x);
12         y = Find(y);
13         fa[y] = x;
14     }
15
16     for(int i = 1; i <= q; i++){
17         cin >> x >> y;
18         x = Find(x);
19         y = Find(y);
20         if(x == y){
21             cout << "yes" << endl;
22         }
23         else {
24             cout << "no" << endl;
25         }
26     }
27
28     return 0;
29 }
```

练习 1: 1536

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4 int fa[1010];
5 int tong[1010];
6 int n, m, cnt;
7 //查找 (2)
8 int Find(int x) {
9     if(fa[x] != x){
10         fa[x] = Find(fa[x]);
11     }
12     return fa[x];
13 }
14 void Init() { //多组样例必涉及初始化
15     // memset(fa, 0, sizeof(fa));
16     memset(tong, 0, sizeof(tong));
17     cnt = 0;
18     //初始化并查集 (1)
19     for(int i = 1; i <= n; i++) {
20         fa[i] = i;
21     }
22 }

```

```

1 int main() {
2     while(cin >> n) {
3         Init();
4         if(n == 0) {
5             return 0;
6         }
7         cin >> m;
8
9         int x = 0, y = 0;
10        for(int i = 1; i <= m; i++) {
11            cin >> x >> y;
12            //(3) 合并
13            x = Find(x);
14            y = Find(y);
15            fa[y] = x;
16        }
17
18        for(int i = 1; i <= n; i++) {
19            if(fa[i] == i) {
20                cnt++;
21            }
22        }
23
24        cnt--;
25        cout << cnt << endl;
26    }
27    return 0;
28 }

```

练习 2: 2814

解题思路:

通过 map 解决重复出现的值的问题, 自动筛选, 即使用 map 替换并查集中的 Fa 数组;

初始化: 若没有出现父节点前, 将其初始化为空, 即通过如下代码进行判断是否有父节点:
`ma.find("str") == ma.end();`

```

1 #include <iostream>
2 #include <map>
3 using namespace std;
4 int n, m, cnt;
5 map<string, string> ma;
6 string Find(string x) {
7     if(ma.find(x) == ma.end()){
8         return x;
9     }
10    ma[x] = Find(ma[x]);
11    return ma[x];
12 }
13 int main() {
14     char curChar;
15     string name, father;
16     while(cin >> curChar){
17         if(curChar == '$'){
18             return 0;
19         }
20         cin >> name;
21         if(curChar == '#'){
22             father = name;
23         }
24         else if(curChar == '+'){
25             ma[name] = father;
26         }
27         else if(curChar == '?'){
28             cout << name << " " << Find(name) << endl;
29         }
30     }
31     return 0;
32 }

```

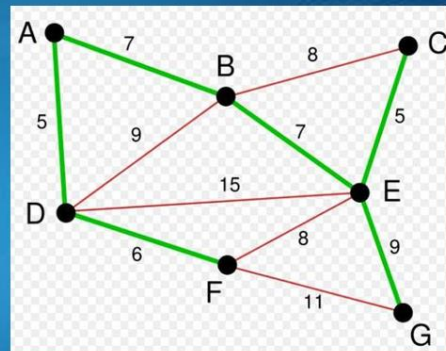
三、最小生成树

假设：我们要在 n 个城市中建立一个通信网络，则连通这 n 个城市需要布置 $n-1$ 条通信线路，这个时候我们需要考虑如何在成本最低的情况下建立这个通信网？

数据结构建立：

- n 个城市就是图上的 n 个顶点，
 - 边表示两个城市的通信线路
 - 每条边上的权重就是我们搭建这条线路所需要的成本
- n 个顶点的连通网可以建立不同的生成树，每一颗生成树都可以作为一个通信网，当我们构造这个连通网所花的成本最小时，搭建该连通网的生成树，就称为**最小生成树**。

定义：所有边的权值之和最小的生成树



kruskal (克鲁斯卡尔算法)

思想：将连通网图中的所有边按照权值大小进行升序排序，从小到大依次选择。

方法：

建立并查集，每个点各自构成一个集合

边升序排列，依次扫边 (x,y,z)

若 x,y 属于同一集合，则忽略此边，继续扫

否则，合并 x,y ，累加 z 到 MST 中

所有边扫完，结束算法

模板：

```
20 #include <iostream>
21 #include <algorithm>
22 using namespace std;
23 struct edge{
24     int x;
25     int y;
26     int z;
27 } e[500010];
28 int fa[100010];
29 int n, m, ans;
30 int Find(int x){
31     if(fa[x] != x){
32         fa[x] = Find(fa[x]);
33     }
34     return fa[x];
35 }
36 bool Cmp(edge a, edge b){
37     return a.z < b.z;
38 }
```

```
1 int main(){
2     cin >> n >> m;
3     for(int i = 1; i <= m; i++){
4         cin >> e[i].x >> e[i].y >> e[i].z;
5     }
6     sort(e + 1, e + m + 1, Cmp);
7     for(int i = 1; i <= n; i++){
8         fa[i] = i;
9     }
10    for(int i = 1; i <= m; i++){
11        int x = Find(e[i].x);
12        int y = Find(e[i].y);
13        if(x != y){
14            fa[y] = x;
15            ans += e[i].z;
16        }
17    }
18    cout << ans;
19 }
```


练习 1: 1195

n 个点的所有边中连 k 条边可以生成 n-k 棵最小生成树

转化：连 n-k 条边，构造 k 个最小生成树就可以了。

使用最小生成树模板，在连边的时候判断已经连了几条边 (cc)，如果 $cc \geq n-k$ 的话，即代表已经构造了 K 个棉花糖，break 输出代价和即可。如果正常构建 MST 完毕后，cc 不够 n-k 的话，输出 No Answer

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int n , m , k , ans;
4  int fa[100010];
5  int cc;
6  struct edge{
7      int x , y , z;
8  } e[500010];
9
10 bool Cmp(edge a , edge b){
11     return a.z < b.z;
12 }
13
14 int find (int x){
15     if(x != fa[x]){
16         fa[x] = find(fa[x]);
17     }
18     return fa[x];
19 }
20
21 int main () {
22     cin >> n >> m >> k;
23     for (int i = 1; i <= n; i++){
24         fa[i] = i;
25     }
26     for (int i = 1; i <= m; i++){
27         cin >> e[i].x >> e[i].y >> e[i].z;
28     }
29     sort(e + 1 , e + m + 1 , Cmp);
30     for (int i = 1; i <= m; i++){
31         int x1 = find(e[i].x);
32         int y1 = find(e[i].y);
33         if (x1 != y1){
34             cc++;
35             fa[y1] = x1;
36             ans += e[i].z;
37         }
38         if (cc >= n - k){
39             break;
40         }
41     }
42     if(cc >= n - k){
43         cout << ans;
44     }
45     else{
46         cout << "No Answer";
47     }
48     return 0;
49 }

```

练习 2: 2872

此题的难点是在构建两点之间的边，同样是用的两点之间的距离：

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

这样就可以根据读入的点的信息，将每个点之间的距离算出来作为边的权值

知道了每个点的相连接需要的代价后就可以构建最小生成树了。

首先先把已知相连的点读入，他们的代价就是 0.

然后再将所有边升序排序，从最小的边开始连，直到连接的次数到了 N-1，就构建完毕，break 输出即可。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int n , m;
4  double ans;
5  int fa[1010];
6  int cc;
7  int cnt;
8
9  struct edge1{
10     double x , y;
11 } e1[1010];
12
13 struct edge2{
14     int x , y;
15     double z;
16 } e2[501010];
17
18 double solve(int x, int y) {
19     return (sqrt((e1[x].x - e1[y].x) * (e1[x].x - e1[y].x)
20     + (e1[x].y - e1[y].y) * (e1[x].y - e1[y].y)));
21 }
22
23 bool Cmp(edge2 a , edge2 b){
24     return a.z < b.z;
25 }
26
27 int find (int x){
28     if(x != fa[x]){
29         fa[x] = find(fa[x]);
30     }
31     return fa[x];
32 }
33

```

```

1
2  int main () {
3      cin >> n >> m;
4      for (int i = 1; i <= n; i++){
5          fa[i] = i;
6      }
7      for (int i = 1; i <= n; i++){
8          cin >> e1[i].x >> e1[i].y;
9      }
10     for (int i = 1; i <= n; i++){
11         for (int j = i + 1; j <= n; j++){
12             cnt++;
13             e2[cnt].x = i;
14             e2[cnt].y = j;
15             e2[cnt].z = solve(i, j);
16         }
17     }
18     for (int i = cnt + 1; i <= cnt + m; i++){
19         cin >> e2[i].x >> e2[i].y;
20         e2[i].z = 0.0;
21     }
22     sort(e2 + 1 , e2 + cnt + m + 1 , Cmp);
23     cc = 0;
24     for (int i = 1; i <= cnt + m; i++){
25         int x1 = find(e2[i].x);
26         int y1 = find(e2[i].y);
27         if (x1 != y1){
28             fa[y1] = x1;
29             cc++;
30             ans += e2[i].z;
31         }
32         if(cc == n - 1){
33             break;
34         }
35     }
36     cout << fixed << setprecision(2) << ans;
37     return 0;
38 }

```