

7.10 课堂笔记 21

本次作业 (7.15 之前提交)

1. 洛谷 P5663 (最短路, bfs)
2. 洛谷 P2047 (最短路, floyed)
3. 洛谷 P1529 (最短路, 堆加邻接表优化的 dijkstra)
4. 洛谷 P1551 (并查集)
5. 洛谷 P3366 (最小生成树模板)
6. 洛谷 P3243 (拓扑排序)

课堂内容

1. Bellman-Ford

Bellman-ford 算法适用于单源最短路径, 图中边的权重可为负数, 但不可以出现负权环。

算法复杂度: $O(NM)$

贝尔曼-福特算法与迪科斯彻算法类似, 都以松弛操作为基础, 即估计的最短路径值渐渐地被更加准确的值替代, 直至得到最优解。

松弛函数

对边集合 E 中任意边, 以 $w(u, v)$ 表示顶点 u 出发到顶点 v 的边的权值, 以 $d[v]$ 表示当前从起点 s 到顶点 v 的路径权值

若存在边 $w(u, v)$, 使得:

$$d[v] > d[u] + w(u, v)$$

则更新 $d[v]$ 值:

$$d[v] = d[u] + w(u, v)$$

所以松弛函数的作用, 就是判断是否经过某个顶点, 或者说经过某条边, 可以缩短起点到终点的路径权值。

思路与 Dijkstra 最大的不同是每次都是从源点 s 重新出发进行"松弛"更新操作, 而 Dijkstra 则是从源点出发向外扩逐个处理相邻的节点, 不会去重复处理节点, 这边也可以看出 Dijkstra 效率相对更高点。

算法流程:

扫描所有边 (x, y, z) , 若 $\text{dist}[y] > \text{dist}[x] + z$, 则 $\text{dist}[y] = \text{dist}[x] + z$
重复上述所有操作, 直到没有更新操作发生。

```
17 for(int i = 1; i <= _m; i++) {
18     _dis[i] = 0x7fffffff / 3;
19     _f[i][1] = _f[i][2] = 0x7fffffff / 3;
20 }
21 for(int i = 1; i <= _m; i++) {
22     cin >> x >> y;
23     _f[i][1] = x; //起点
24     _f[i][2] = y; //终点
25     _w[i] = sqrt(pow(_map[x][1] - _map[y][1], 2) +
26                 pow(_map[x][2] - _map[y][2], 2));
27 }
28
29 cin >> _s >> _t;
30 _dis[_s] = 0;
31 for(int i = 1; i <= _n; i++) {
32     for(int j = 1; j <= _m; j++) {
33         if(_dis[_f[j][1]] + _w[j] < _dis[_f[j][2]]) {
34             _dis[_f[j][2]] = _dis[_f[j][1]] + _w[j];
35         }
36         if(_dis[_f[j][2]] + _w[j] < _dis[_f[j][1]]) {
37             _dis[_f[j][1]] = _dis[_f[j][2]] + _w[j];
38         }
39     }
40 }
```

2. SPFA: 队列优化的 BellmanFord:

用途: 求含负权边的单源最短路径, 以及判负权环

- SPFA 最坏情况下复杂度和朴素 Bellman-Ford 相同, 为 $O(MN)$ 。
- 随机图中时间复杂度为 $O(kM)$, k 为较小常数。

为了避免最坏情况的出现, 在正权图上应使用效率更高的 Dijkstra 算法

算法流程:

- 建立一个队列, 队列初始化时, 只含有起点 1。
- 取出队头结点 x , 扫描其出边 (x, y, z) , 若 $\text{dist}[y] > \text{dist}[x] + z$, 则 $\text{dist}[y] = \text{dist}[x] + z$
- 若 y 不在队列中, 则 y 入队。
- 重复至队列为空。

任意时刻, 该算法的队列都保存了待扩展的结点, 每次入队相当于完成一次 dist 数组的更新操作。

一个结点可能会多次入队，多次出队。

避免了 Bellman-Ford 算法中对不需要扩展结点的冗余扫描。

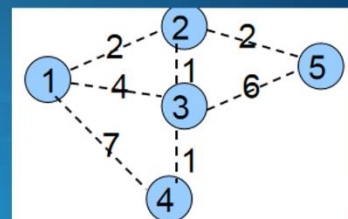
```
25 void SPFA() {
26     memset(_dis, 0x3f, sizeof(_dis));
27     _dis[1] = 0; //起点
28     _vis[1] = true;
29     _que.push(1);
30     while(!_que.empty()) {
31         int x = _que.front();
32         _que.pop();
33         _vis[x] = false;
34         for(int i = _head[x]; i != 0; i = _e[i].next) {
35             int y = _e[i].to;
36             int z = _e[i].w;
37             if(_dis[x] + z < _dis[y]) {
38                 _dis[y] = _dis[x] + z;
39                 if(!_vis[y]) {
40                     _que.push(y);
41                     _vis[y] = true;
42                 }
43             }
44         }
45     }
46 }
```

3. prim (普里姆算法)

算法思路：以顶点为主导地位，从起始顶点出发，通过选择当前可用的最小权值边把顶点加入到生成树中：

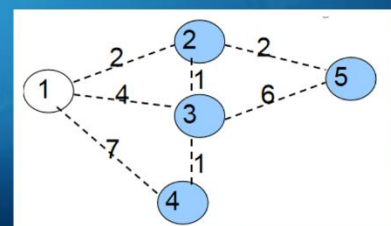
每次循环都将一个蓝点 u 变为白点，并且此蓝点 u 与白点相连的最小边权 $\min[u]$ 是当前所有蓝点中最小的。这样相当于向生成树中添加了 $n-1$ 次最小的边，最后得到的一定是最小生成树。

- 蓝点和虚线代表未进入最小生成树的点、边；
- 白点和实线代表已进入最小生成树的点、边。
- 初始时所有点都是蓝点
- $\min[1]=0$
- $\min[2, 3, 4, 5]=\infty$
- 权值之和 $MST=0$ 。



- 第一次循环找到 $\min[1]=0$ 最小的蓝点1。
- 将1变为白点，接着枚举与1相连的所有蓝点2、3、4，修改它们与白点相连的最小边权。

```
min[1]=0
min[2]=w[1][2]=2;
min[3]=w[1][3]=4;
min[4]=w[1][4]=7;
```

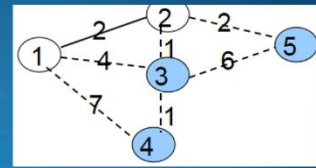


- 第二次循环是找到 $\min[2]$ 最小的蓝点2。
- 将2变为白点
- 枚举与2相连的所有蓝点3、5，修改它们与白点相连的最小边权。

$\min[1]=0$
 $\min[2]=w[1][2]=2;$
 $\min[3]=w[1][3]=4;$
 $\min[4]=w[1][4]=7;$

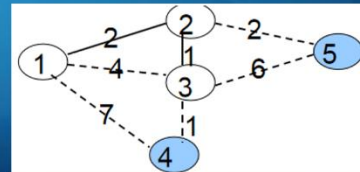


$\min[1]=0$
 $\min[2]=2$
 $\min[3]=w[2][3]=1;$
 $\min[4]=7$
 $\min[5]=w[2][5]=2;$



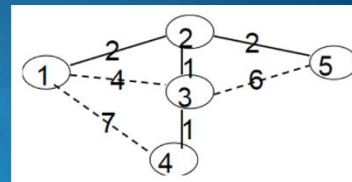
- 第三次循环是找到 $\min[3]$ 最小的蓝点3。
- 将3变为白点
- 枚举与3相连的所有蓝点4、5，修改它们与白点相连的最小边权。

$\min[4]=w[3][4]=1;$



最后两轮循环将点4、5以及边
 $w[2][5], w[3][4]$ 添加进最小生成树。

$\min[1]=0$
 $\min[2]=2$
 $\min[3]=1;$
 $\min[4]=1$
 $\min[5]=2;$



最后权值之和 $MST=6$ 。

这 n 次循环，每次循环我们都能让一个新的点加入生成树， n 次循环就能把所有点囊括到其中；

每次循环我们都取一条最小的边加入生成树， $n-1$ 次循环结束后，我们得到的就是一棵最小的生成树。这就是Prim采取贪心法生成一棵最小生成树的原理。算法时间复杂度： $O(n^2)$ 。

```

12 void Prim(int pos) {
13     _dis[pos] = 0;
14     //n个点遍历n次, 每次寻找一个权值最小的点, 记录其下标
15     for(int i = 1; i <= _n; i++) {
16         int k = 0;
17         int minx = MAXNUM;
18         for(int j = 1; j <= _n; j++) {
19             if(!_b[j] && _dis[j] < minx) {
20                 k = j;
21                 minx = _dis[j];
22             }
23         }
24         // 这里需要提前终止
25         if(_dis[k] >= MAXNUM) {
26             _flag = true;
27             return;
28         }
29         _sum += _dis[k];
30         _b[k] = true; // 蓝变白
31         for(int j = 1; j <= _n; j++) {
32             // 只更新还没有找到的最小权值
33             if(!_b[j]) {
34                 _dis[j] = min(_dis[j], _w[k][j]);
35             }
36         }
37     }
38 }

```

课堂练习

1. P3371 【模板】单源最短路径（弱化版）

题目描述

如题，给出一个有向图，请输出从某一点出发到所有点的最短路径长度。

输入格式

第一行包含三个整数 n, m, s ，分别表示点的个数、有向边的个数、出发点的编号。

接下来 m 行每行包含三个整数 u, v, w ，表示一条 $u \rightarrow v$ 的，长度为 w 的边。

输出格式

输出一行 n 个整数，第 i 个表示 s 到第 i 个点的最短路径，若不能到达则输出 $2^{31} - 1$ 。

输入输出样例

输入 #1

复制

输出 #1

复制

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

```
0 2 4 3
```

说明/提示

【数据范围】

对于 20% 的数据： $1 \leq n \leq 5, 1 \leq m \leq 15$;

对于 40% 的数据： $1 \leq n \leq 100, 1 \leq m \leq 10^4$;

对于 70% 的数据： $1 \leq n \leq 1000, 1 \leq m \leq 10^5$;

对于 100% 的数据： $1 \leq n \leq 10^4, 1 \leq m \leq 5 \times 10^5, 1 \leq u, v \leq n, w \geq 0, \sum w < 2^{31}$ ，保证数据随机。

代码：


```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int _n, _m, _k, c, d, a, b, _cnt;
4  struct p{
5      int nt, to, w;
6  } _num[500005];
7  int _head[500500];
8  int _b[500005], _dis[500000];
9  queue<int> _mq;
10 void putin(int a, int b, int c) {
11     _cnt++;
12     _num[_cnt].to=b;
13     _num[_cnt].nt=_head[a];
14     _num[_cnt].w=c;
15     _head[a]=_cnt;
16 }
17 void spfa(int st) {
18     memset(_dis, 0x3f, sizeof(_dis));
19     _dis[st]=0;
20     _b[st]=1;
21     _mq.push(st);
22     int bj=0;
23     while(!_mq.empty()) {
24         bj=_mq.front();
25         cout<<bj<<endl;
26         _mq.pop();
27         _b[bj]=0;
28         for(int i = _head[bj]; i!=0; i=_num[i].nt) {
29             int y=_num[i].to;
30             int z=_num[i].w;
31             if(_dis[bj]+z<_dis[y]) {
32                 _dis[y]=_dis[bj]+z;
33                 if(!_b[y]) {
34                     _mq.push(y);
35                     _b[y]=1;
36                 }
37             }
38         }
39     }
40 }
41
42
43 int main() {
44     cin >> _n >> _m >> _k;
45     for(int i = 1; i <= _m; i++) {
46         cin >> a >> b >> c;
47         putin(a, b, c);
48     }
49     spfa(_k);
50     for(int i = 1; i <= _n; i++) {
51         if(_dis[i] != 0x3f3f3f3f) cout << _dis[i] << " ";
52         else cout << INT_MAX << " ";
53     }
54     return 0;
55 }
56

```

2. P1744 采购特价商品

距离+SPFA 模板

代码:

```

#include<bits/stdc++.h>

using namespace std;

int _n, _m, _k, c, d, a, b, _cnt, n1[100000], n2[100000];
struct p{
    double nt, to, w;
} _num[1000005];
int _head[5000500];
double _b[1000005], _dis[1000000];
queue<int> _mq;

double js(int a, int b, int c, int d){
    return sqrt(pow(a-b, 2)+pow(c-d, 2));
}

void putin(int a, int b, double c){
    _cnt++;
    _num[_cnt].to=b;
    _num[_cnt].nt=_head[a];
    _num[_cnt].w=c;
    _head[a]=_cnt;
}

void spfa(int st){
    for(int i = 1; i <= _n; i++) _dis[i]=114514.00;
    _dis[st]=0;
    _b[st]=1;
    _mq.push(st);
    int bj=0;

    while(!_mq.empty()){
        bj=_mq.front();
        _mq.pop();
        _b[bj]=0;
        for(int i = _head[bj]; i!=0; i=_num[i].nt){
            int y=_num[i].to;
            double z=_num[i].w;
            if(_dis[bj]+z<_dis[y]){
                _dis[y]=_dis[bj]+z;
                if(!_b[y]){
                    _mq.push(y);
                    _b[y]=1;
                }
            }
        }
    }

int main(){
    cin >> _n;
    for(int i = 1; i <= _n; i++){
        cin >> n1[i] >> n2[i];
    }
    cin >> _m;
    for(int i = 1; i <= _m; i++){
        cin >> a >> b;
        putin(a, b, js(n1[a], n1[b], n2[a], n2[b]));
        putin(b, a, js(n1[a], n1[b], n2[a], n2[b]));
    }
    cin >> _k;
    spfa(_k);
    cin >> _k;
    cout << fixed << setprecision(2) << _dis[_k];
    return 0;
}

```

3. P1744 采购特价商品

距离+prim 模板

代码:

```
#include<bits/stdc++.h>

using namespace std;

int _n, _m, _k, c, d, a, b, _cnt, _max, n1[100000], n2[100000];
struct p{
    double nt, to, w;
} _num[100086];

vector<p> ald;

int _head[5000500], _map[10005][10005];
int _b[1000005], _dis[1000000], _sum;

void prim() {
    _dis[1]=0;
    for(int i = 1; i <= _n; i++) {
        int k=0;
        int _max=INT_MAX;
        for(int j = 1; j <= _n; j++) {
            if(!_b[j]&&_dis[j]<_max) {
                k=j;
                _max=_dis[j];
            }
        }
        if(_dis[k]>= INT_MAX) {
            //
            fg=1;
            return;
        }
        _sum+=_dis[k];
        _b[k]=1;
        for(int j = 1; j <= _n; j++) {
            if(!_b[j]) {
                _dis[j]=min(_dis[j], _map[k][j]);
            }
        }
    }
}

int main() {
    cin >> _n ;
    memset(_dis, 0x3f, sizeof(_dis));
    for(int i = 1; i <= _n; i++) {
        for(int j = 1; j <= _n; j++) {
            cin >> _map[i][j];
        }
    }
    prim();
    cout << _sum;
    return 0;
}
```

4. P1991 无线通讯网

模板里有 $n-1$ 条边退出循环, 这里考虑电台数量, s 个卫星电话就不用连接最大的 $s-1$ 条边。

代码:

```

#include<bits/stdc++.h>

using namespace std;

int _n, _m, _k, c, d, a, b, _cnt, _max, n1[100000], n2[100000];
struct p{
    double nt, to, w;
} _num[100086];

vector<p> ald;

double _head[5000500], _map[10005][10005];
double _b[1000005], _dis[1000000], _sum; int _s;

double js(int a, int b, int c, int d){
    return sqrt(pow(a-b, 2)+pow(c-d, 2));
}

void prim(){
    for(int i = 1; i <= _n; i++) _dis[i]=10000;
    _dis[1]=0;
    for(int i = 1; i <= _n; i++){
        int k=0;
        double _max=1000000;
        for(int j = 1; j <= _n; j++){
            if(!_b[j]&&_dis[j]<_max){
                k=j;
                _max=_dis[j];
            }
        }
        if(_dis[k]>= 1000000){
            return;
        }
        _sum+=_dis[k];
        _b[k]=1;
        for(int j = 1; j <= _n; j++){
            if(!_b[j]){
                _dis[j]=min(_dis[j], _map[k][j]);
            }
        }
    }
}

int main(){
    cin >> _s >> _n;
    for(int i = 1; i <= _n; i++){
        for(int j = 1; j <= _n; j++) _map[i][j]=10000;
    }
    for(int i = 1; i <= _n; i++){
        cin >> n1[i] >> n2[i];
    }
    for(int i = 1; i <= _n; i++){
        for(int j = 1; j <= _n; j++){
            _map[i][j]=js(n1[i], n1[j], n2[i], n2[j]);
        }
    }
    prim();
    sort(_dis+1, _dis+1+_n);
    cout << fixed << setprecision(2) << _dis[_n-_s+1];
    return 0;
}

```

5. P1396 营救

将边从小到大排序，然后最小生成树连边，当 S 和 T 第一次联通的时候当前权值就是答案

代码：

```

2
3 #include<bits/stdc++.h>
4
5 using namespace std;
6
7 int _n,_m,_k,c,d,a,b,_cnt,_ans,_max,n1[100000],n2[100000],n3[100000];
8 int _f[100005],_sum;int _s,_e;
9
10
11 int f(int a){
12     if(_f[a]!=a)return f(_f[a]);
13     return _f[a];
14 }
15
16 bool f2(int a,int b){
17     if(a==b)return 1;
18     else if(_f[a]==a)return 0;
19     return f2(_f[a],b);
20 }
21
22 int _min=1000000;
23
24 bool ck(int a){
25     for(int i =1;i <= _n;i++)_f[i]=i;
26     for(int i = 1;i <= _m;i++){
27         if(n3[i]<=a){
28             int fn1=f(n1[i]),fn2=f(n2[i]);
29             if(fn1!=fn2){
30                 _f[fn1]=fn2;
31             }
32         }
33     }
34     if(f(_s)==f(_e))return 1;
35     return 0;
36 }

```

```

void fen() {
    int l =_min, r= _max,mid=0;
    while(l<=r){
        mid=(l+r)/2;
        if(ck(mid)){
            _ans=mid;
            r=mid-1;
        }else{
            l=mid+1;
        }
    }
    cout << _ans;
}

```

```

int main() {
    cin >> _n >> _m >> _s >> _e;
    for(int i = 1;i <= _m;i++){
        cin >> n1[i] >> n2[i] >> n3[i];
        _max=max(_max,n3[i]);
        _min=min(_min,n3[i]);
    }
    fen();
    return 0;
}

```