

## 7.17 课堂笔记 23

本次作业 (7.23 之前提交)

1. 洛谷 P7913
2. 洛谷 P7914
3. 洛谷 P7915
4. 洛谷 P7916

### 课堂内容

1. 最小生成树

T1: P2820

第一问：为什么要用最小生成树？

- 权值和最值问题

第二问：怎么求被删掉的边权值和最大？

- 使得被保留的边权值和最小即可
- 答案 = 所有边权值和 - 最小生成树权值和

代码实现：

第一步：存边，排序，求所有边权值和

```
int main () {
    cin >> n >> k;
    for(int i = 1; i <= k; i++){
        cin >> e[i].u >> e[i].v >> e[i].w;
        sum += e[i].w;
    }
    sort(e + 1, e + 1 + k, cmp);
```

第二步：利用并查集连边，求最小生成树

```
int Find(int x){
    if(fa[x] != x){
        fa[x] = Find(fa[x]);
    }
    return fa[x];
}
```

```

for(int i = 1; i <= n; i ++){
    fa[i] = i;
}
for(int i = 1; i <= k; i ++){
    int x = Find(e[i].u);
    int y = Find(e[i].v);
    if(x != y){
        ans += e[i].w;
        fa[y] = x;
    }
}
}

```

完整代码：

```

1  #include<iostream>
2  #include<algorithm>
3  using namespace std;
4  struct mst{
5      int u;
6      int v;
7      int w;
8  };
9  int n = 0;
10 int k = 0;
11 mst e[6010];
12 int fa[110];
13 long long sum;
14 long long ans;
15 bool cmp(mst one, mst two){
16     return one.w < two.w;
17 }
18 int Find(int x){
19     if(fa[x] != x){
20         fa[x] = Find(fa[x]);
21     }
22     return fa[x];
23 }

```

```

24 int main () {
25     cin >> n >> k;
26     for(int i = 1; i <= k; i++){
27         cin >> e[i].u >> e[i].v >> e[i].w;
28         sum += e[i].w;
29     }
30     sort(e + 1, e + 1 + k, cmp);
31     for(int i = 1; i <= n; i++){
32         fa[i] = i;
33     }
34     for(int i = 1; i <= k; i++){
35         int x = Find(e[i].u);
36         int y = Find(e[i].v);
37         if(x != y){
38             ans += e[i].w;
39             fa[y] = x;
40         }
41     }
42     cout << sum - ans;
43     return 0;
44 }

```

## 2. 最短路——dijkstra

T2: P3905

第一问：为什么要用 dijkstra？

- 有权值，没有负权
- 单源最短路问题，某点到某点

第二问：怎么计算需要修复的最短路？

- 将没有被毁掉的道路的权值再赋为 0 即可

代码实现：

第一步：初始化，存边

```

int main () {
    cin >> _n >> _m;
    memset(_w, 0x3f, sizeof(_w));
    memset(_dis, 0x3f, sizeof(_dis));
    int u = 0, v = 0, w = 0;
    for(int i = 1; i <= _m; i++) {
        cin >> u >> v >> w;
        _w[u][v] = w;
        _w[v][u] = w;
    }
}

```

第二步：标记被毁掉的边

```
cin >> _d;
for(int i = 1; i <= _d; i++){
    cin >> u >> v;
    _b[u][v] = true;
    _b[v][u] = true;
}
```

第三步：没有被毁掉的边的权值全部改为 0

```
cin >> A >> B;
for(int i = 1; i <= _n; i++){
    for(int j = 1; j <= _n; j++){
        if(_w[i][j] != 1061109567 && _b[i][j] == false){
            _w[i][j] = 0;
            _w[j][i] = 0;
        }
    }
}
Dijkstra(A);

return 0;
```

第四步：dijkstra 求 B 到 A 的最短距离

```
void Dijkstra(int sta) {
    _dis[sta] = 0;
    for(int i = 1; i <= _n; i++) {
        _minx = 2e9;
        _k = 0;
        for(int j = 1; j <= _n; j++) {
            if(!_blue[j] && _dis[j] < _minx) {
                _minx = _dis[j];
                _k = j;
            }
        }
        _blue[_k] = true;
        for(int j = 1; j <= _n; j++) {
            if(_dis[_k] + _w[_k][j] < _dis[j]) {
                _dis[j] = _dis[_k] + _w[_k][j];
            }
        }
    }
    cout << _dis[B] << " ";
}
```

完整代码：

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  int _w[110][110];
5  int _dis[110];
6  int _pre[110];
7  bool _blue[110];
8  bool _b[110][110];
9  int _n, _m, _d, A, B;
10 int _k, _minx;
11 void Dijkstra(int sta) {
12     _dis[sta] = 0;
13     for(int i = 1; i <= _n; i++) {
14         _minx = 2e9;
15         _k = 0;
16         for(int j = 1; j <= _n; j++) {
17             if(!_blue[j] && _dis[j] < _minx) {
18                 _minx = _dis[j];
19                 _k = j;
20             }
21         }
22         _blue[_k] = true;
23         for(int j = 1; j <= _n; j++) {
24             if(_dis[_k] + _w[_k][j] < _dis[j]) {
25                 _dis[j] = _dis[_k] + _w[_k][j];
26             }
27         }
28     }
29     cout << _dis[B] << " ";
30 }
31 int main () {
32     cin >> _n >> _m;
33     memset(_w, 0x3f, sizeof(_w));
34     memset(_dis, 0x3f, sizeof(_dis));
35     int u = 0, v = 0, w = 0;
36     for(int i = 1; i <= _m; i++) {
37         cin >> u >> v >> w;
38         _w[u][v] = w;
39         _w[v][u] = w;
40     }
41     cin >> _d;
42     for(int i = 1; i <= _d; i++){
43         cin >> u >> v;
44         _b[u][v] = true;
45         _b[v][u] = true;
46     }
47     cin >> A >> B;
48     for(int i = 1; i <= _n; i++){
49         for(int j = 1; j <= _n; j++){
50             if(_w[i][j] != 1061109567 && _b[i][j] == false){
51                 _w[i][j] = 0;
52                 _w[j][i] = 0;
53             }
54         }
55     }
56     Dijkstra(A);
57
58     return 0;
59 }

```

### 3. 拓扑排序

T3: P1960

第一问：为什么是拓扑排序？

- 典型的点排序问题

第二问：如何判断拓扑排序是不是唯一的？

- 拓扑排序之前，判断入度为 0 的点是否大于等于 2
- 拓扑排序之中，判断入度为 0 的 y 的个数是否大于等于 2

第三问：如果要你求具体有几种拓扑排序怎么办？

代码实现：

第一步：存边，求每个点的入度

```
int main(){
    cin >> n >> m;
    for(int i = 1; i <= m; i++){
        cin >> u >> v;
        rudu[v] ++;
        AddEdge(u, v);
    }
    TuoPu();
    return 0;
}
```

第二步：拓扑排序之前，判断入度为 0 的个数是不是 大于等于 2

```
void TuoPu(){
    for(int i = 1; i <= n; i++){
        if(rudu[i] == 0){
            que.push(i);
            cnt++;
        }
    }
    if(cnt > 1){
        ans = 1;
    }
    cnt = 0;
```

第三步：拓扑排序之中，判断入度为 0 的个数是不是 大于等于 2

```
while(!que.empty()){
    int x = que.front();
    que.pop();
    cout << x << endl;
    for(int i = head[x]; i != 0; i = e[i].next){
        int y = e[i].to;
        rudu[y] --;
        if(rudu[y] == 0){
            que.push(y);
            cnt++;
        }
    }
    if(cnt > 1){
        ans = 1;
    }
    cnt = 0;
}
cout << ans;
```



完整代码：

```
1  #include <iostream>
2  #include <queue>
3  using namespace std;
4  int numEdge = 0;
5  int n = 0;
6  int m = 0;
7  int u = 0;
8  int v = 0;
9  int cnt = 0;
10 int ans = 0;
11 int head[10010];
12 int rudu[10010];
13 queue<int> que;
14 struct edge{
15     int next;
16     int to;
17 }e[1000010];
18 void AddEdge(int from, int to){
19     numEdge++;
20     e[numEdge].next = head[from];
21     e[numEdge].to = to;
22     head[from] = numEdge;
23 }
24 void TuoPu(){
25     for(int i = 1; i <= n; i++){
26         if(rudu[i] == 0){
27             que.push(i);
28             cnt++;
29         }
30     }
31     if(cnt > 1){
32         ans = 1;
33     }
34     cnt = 0;
35
36     while(!que.empty()){
37         int x = que.front();
38         que.pop();
39         cout << x << endl;
40         for(int i = head[x]; i != 0; i = e[i].next){
41             int y = e[i].to;
42             rudu[y]--;
43             if(rudu[y] == 0){
44                 que.push(y);
45                 cnt++;
46             }
47         }
48         if(cnt > 1){
49             ans = 1;
50         }
51         cnt = 0;
52     }
53     cout << ans;
54
55 int main(){
56     cin >> n >> m;
57     for(int i = 1; i <= m; i++){
58         cin >> u >> v;
59         rudu[v]++;
60         AddEdge(u, v);
61     }
62     TuoPu();
63     return 0;
64 }
```

#### 4. 最短路—floyd

T4: P1119

第一问：为什么是 floyed?

- 多源最短路问题

第二问：怎么处理修好的时间限制?

- 询问时的时间要大于等于某个城市的修好时间，该城市才能作为中转点

完整代码：

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int g[210][210]; // 两城市之间的最短距离
4  int day[210]; // 某城市修建好的时间
5  int n, m;
6  int x, y, w;
7  int q, _time, k; // k表示中转点城市
8
9  void floyed(){
10     for(int i=0; i<n; i++){
11         for(int j=0; j<n; j++){
12             if(i!=j && i!=k && j!=k){
13                 g[i][j] = min(g[i][j], g[i][k]+g[k][j]);
14             }
15         }
16     }
17 }
18
19 int main(){
20     cin >> n >> m;
21     for(int i=0; i<n; i++){
22         cin >> day[i]; // 每个城市修好需要的时间
23     }
24     memset(g, 0x3f, sizeof(g));
25     for(int i=0; i<m; i++){
26         cin >> x >> y >> w;
27         g[x][y] = w; // 存边
28         g[y][x] = w;
29     }
30     cin >> q;
31     for(int i=0; i<q; i++){
32         cin >> x >> y >> _time;
33         while(_time >= day[k] && k<n){ // 修好时间小于_time的城市才能作为中转点
34             floyed();
35             k++;
36         }
37
38         // x、y城市修好时间小于_time 或者 路走不通，都输出-1
39         if(_time < day[x] || _time < day[y] || g[x][y] == 1061109567){
40             cout << -1 << endl;
41         }
42         else{
43             cout << g[x][y] << endl;
44         }
45     }
46     return 0;
47 }
```

5. 初赛答案：



# 提高级 CSP-S 第 11 套初赛模拟试题答案及解析

## 一、单项选择题

题号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
答案	B	B	D	C	C	B	A	D	D	A	C	A	C	D	B

### 【解析】

1.  $(2333)_{10} = 2333$ ,  $(999)_{10} = 2457$ ,  $(100011111111)_2 = 2303$ ,  $2^{11} = 2048$ 。
2. 考查计算机基础知识。寄存器的存取速度最快。
3.  $128 \text{ KB} = 128 \times 1024 \text{ B} = 128 \times 1024 \times 8 \text{ bit}$ 。
4. 考查队列基本知识。共有三次出队操作,所以队首就是第四次入队的元素。
5. 考查位运算基本知识。
6. 一张两侧点数分别为  $x$  和  $y$  的二分图至多有  $xy$  条边,这里取  $x=4, y=5$ 。
7. 答案为  $5 + 4 + 3 + 2 + 1 = 15$ 。
8. 考查排序基本知识。
9. 根据数字模 3 的余数讨论。一个数模 3 所得的余数即为这个数各数位上数字之和模 3 的余数。所以可以把三位数分成 3 位考虑,即找到 3 个数字模 3 结果之和为 3 的倍数。枚举可知:对 3 取模的结果中可行组合有 3 个“0”或者 3 个“1”或者 3 个“2”或者“012”。
10. 考查快速排序相关知识。
11. 考查基础知识。
12. 由于是邻接矩阵,对于每个点都必须  $O(n)$  找边。
13. 找到最大的数  $n$  满足  $n(n+1)/2 \leq 1414$ 。
14. 考查计算机语言基础知识。
15. 纸不能带入考场。

## 二、阅读程序

1.

题号	(1)	(2)	(3)	(4)	(5)	(6)
答案	x	x	x	√	B	C

### 【解析】

程序统计了  $a[i] \text{ xor } a[j]$  中出现次数最多的数(如有多个则取最小的)。

- (1)  $n$  可以等于 5000。
- (2) 输出的可能为 0。
- (3)  $a[i]$  很大时,它们的异或值不一定大。如当  $n=2, a_1=a_2=65535$  时,不会数组越界。
- (4) 因为答案初始化为 0,可以不从 0 开始枚举。
- (5) 只要  $n>1$ ,所有  $a[i]=i$  时,答案一定为 1。
- (6) 0~15 中的所有结果都有可能。设输出为  $t$ ,则令  $n=2, a_1=0, a_2=t$ ,则可以输出 0~15 的任意结果。由于  $\max\{a\} \leq 15 = 2^4 - 1$ ,所以  $t \leq 15$ 。

2.

题号	(1)	(2)	(3)	(4)	(5)	(6)
答案	√	x	x	√	C	C

【解析】

- (1) 每次只会在满足条件的情况下调用该函数。
- (2) 准确地说,应该满足  $i = \max(t_1 + t_2 - n, 0)$ 。
- (3) 因为处理出  $t_1$  后才输入  $s_2$ ,所以答案会变。
- (4) 由于  $s_1$  在统计出  $t_1$  之后并不会再使用,可以重复利用这个数组。
- (5) 当  $s_1$  和  $s_2$  分别为 0101010 和 1101101 时,由于  $t_1 = 3, t_2 = 5$ ,所以答案为  $C_7^6 + C_7^4 + C_7^2 = 63$ 。
- (6) 对于  $1 \sim N$  中的每一个数,都需要做一次快速幂,复杂度为  $O(N \log N)$ 。

3.

题号	(1)	(2)	(3)	(4)	(5)	(6)
答案	×	×	×	×	C	A

【解析】

- (1) 如果不使用数组进行记忆化组合数的值,算法每次求组合数将花费  $O(nm)$  的时间,所以是错误的。
- (2) 预处理 pw 数组时,  $i$  枚举到了 1005,超出了数组定义的范围。超出数组定义的方位是未定义行为,在开启 O2 优化的情况下会运行错误。
- (3) 显然会改变,将  $j=n$  代入验证即可。
- (4) 显然会改变,将  $j=n$  代入验证即可。
- (5) 直接暴力枚举即可。
- (6) 考虑范围比较大,可以直接画表格递推。

三、完善程序

1.

题号	(1)	(2)	(3)	(4)	(5)
答案	C	D	B	C	A

【解析】

看似数据很大:  $n, m, k (1 \leq n, m \leq 1000, 1 \leq k \leq 10)$ , 但是,  $k < n+m-1$  时,可以直接输出 0。因为无法走完一条路径(一条路径长度为  $n+m-1$ ,只能向下、向右走),所以实际数据范围很小,大概是  $n+m-1 \leq 10$  左右吧。这么小的范围很容易就可想到 dfs。

这里有两个优化,一个是如果搜到一半,发现剩下的颜色不够用了就直接 return。还有一个就是利用颜色 A 与颜色 B 的先后次序问题,路径 AB 与路径 BA 并不是同一种方案,所以搜索时如果搜到是第一次时,就可以直接乘 num 就可以省去很多 dfs。

- (1) 记录到达状态。
- (2) 小剪枝。
- (3) 判断是否访问过。
- (4) 记忆化。
- (5) 特判无解。

2.

题号	(1)	(2)	(3)	(4)	(5)
答案	C	D	B	A	D

【解析】

题目要我们求深度最浅的带权重心。对于一个点  $v$ ,如果它是这棵树的带权重心,把它作为根,那么 root 所在的子树大小  $\leq$  权值和的一半,也就是说剩下的子树大小(以 root 为根时子树  $v$  的大小)  $\geq$  权值和的一半,也就是说中位数一定包含在  $v$  所包含的子树里。直

接上轻重链剖分,可以线段树上二分求出中位数,每次倍增向上跳,判一下是否符合条件即可。

(1)维护出每个点最大子树。

(2)线段树基本操作。

(3)注意下文包含了  $bk$ 。

(4)特判。

(5)倍增向上跳,判断是否符合条件。