

# 8.1 课堂笔记 28

## 初赛练习

### CSP-S 第十二套初赛模拟题解析

1.

1	<div><div>()</div><div>[]</div><div>-&gt;</div><div>.</div><div>::</div><div>++</div><div>--</div></div>	<div>调节优先级的括号操作符</div> <div>数组下标访问操作符</div> <div>通过指向对象的指针访问成员的操作符</div> <div>通过对象本身访问成员的操作符</div> <div>作用域操作符</div> <div>后置自增操作符</div> <div>后置自减操作符</div>	<div>(a + b) / 4;</div> <div>array[4] = 2;</div> <div>ptr-&gt;age = 34;</div> <div>obj.age = 34;</div> <div>Class::age = 2;</div> <div>for( i = 0; i &lt; 10; i++ ) ...</div> <div>for( i = 10; i &gt; 0; i-- ) ...</div>	从左到右
2	<div><div>!</div><div>~</div><div>++</div><div>--</div><div>-</div><div>+</div><div>*</div><div>&amp;</div><div>(type)</div><div>sizeof</div></div>	<div>逻辑取反操作符</div> <div>按位取反(按位取补)</div> <div>前置自增操作符</div> <div>前置自减操作符</div> <div>一元取负操作符</div> <div>一元取正操作符</div> <div>解引用操作符</div> <div>取地址操作符</div> <div>类型转换操作符</div> <div>返回对象占用的字节数操作符</div>	<div>if( !done ) ...</div> <div>flags = ~flags;</div> <div>for( i = 0; i &lt; 10; ++i ) ...</div> <div>for( i = 10; i &gt; 0; --i ) ...</div> <div>int i = -1;</div> <div>int i = +1;</div> <div>data = *ptr;</div> <div>address = &amp;obj;</div> <div>int i = (int) floatNum;</div> <div>int size = sizeof(floatNum);</div>	从右到左
3	<div><div>-&gt;*</div><div>.</div></div>	<div>在指针上通过指向成员的指针访问成员的操作符</div> <div>在对象上通过指向成员的指针访问成员的操作符</div>	<div>ptr-&gt;*var = 24;</div> <div>obj.*var = 24;</div>	从左到右
4	<div><div>*</div><div>/</div><div>%</div></div>	<div>乘法操作符</div> <div>除法操作符</div> <div>取余数操作符</div>	<div>int i = 2 * 4;</div> <div>float f = 10 / 3;</div> <div>int rem = 4 % 3;</div>	从左到右
5	<div><div>+</div><div>-</div></div>	<div>加法操作符</div> <div>减法操作符</div>	<div>int i = 2 + 3;</div> <div>int i = 5 - 1;</div>	从左到右
6	<div><div>&lt;&lt;</div><div>&gt;&gt;</div></div>	<div>按位左移操作符</div> <div>按位右移操作符</div>	<div>int flags = 33 &lt;&lt; 1;</div> <div>int flags = 33 &gt;&gt; 1;</div>	从左到右
7	<div><div>&lt;</div><div>&lt;=</div><div>&gt;</div><div>&gt;=</div></div>	<div>小于比较操作符</div> <div>小于或等于比较操作符</div> <div>大于比较操作符</div> <div>大于或等于比较操作符</div>	<div>if( i &lt; 42 ) ...</div> <div>if( i &lt;= 42 ) ...</div> <div>if( i &gt; 42 ) ...</div> <div>if( i &gt;= 42 ) ...</div>	从左到右
8	<div><div>==</div><div>!=</div></div>	<div>等于比较操作符</div> <div>不等于比较操作符</div>	<div>if( i == 42 ) ...</div> <div>if( i != 42 ) ...</div>	从左到右

9	&	按位与操作符	flags = flags & 42;	从左到右
10	^	按位异或操作符	flags = flags ^ 42;	从左到右
11		按位或操作符	flags = flags   42;	从左到右
12	&&	逻辑与操作符	if( conditionA && conditionB ) ...	从左到右
13		逻辑或操作符	if( conditionA    conditionB ) ...	从左到右
14	? :	三元条件操作符	int i = (a > b) ? a : b;	从右到左
15	= += -= *= /= %= &= ^=  = <<= >>=	赋值操作符 复合赋值操作符(加法) 复合赋值操作符(减法) 复合赋值操作符(乘法) 复合赋值操作符(除法) 复合赋值操作符(取余) 复合赋值操作符(按位与) 复合赋值操作符(按位异或) 复合赋值操作符(按位或) 复合赋值操作符(按位左移) 复合赋值操作符(按位右移)	int a = b; a += 3; b -= 4; a *= 5; a /= 2; a %= 3; flags &= new_flags; flags ^= new_flags; flags  = new_flags; flags <<= 2; flags >>= 2;	从右到左
16	,	逗号操作符	for( i = 0, j = 0; i < 10; i++, j++ ) ...	从左到右

2.

计算机辅助制造 Computer aided manufacturing

计算机辅助设计 Computer Aided Design

计算机辅助教学 Computer Aided Instruction

计算机化适应性测试 computerized adaptive test

3.  $(10000 \times 1024 \times 1536 \times 32) / (300 \times 1024 \times 1024 \times 8)$

4.

$a \times (b + c) \times d$

$((a \times (b + c)) \times d)$

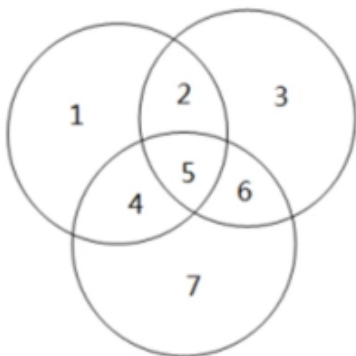
$((a \times (b + c) + d) \times$

$abc + d \times$

7.

两个集合的容斥关系公式： $A \cup B = |A \cup B| = |A| + |B| - |A \cap B|$  ( $\cap$ : 重合的部分)

三个集合的容斥关系公式： $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |C \cap A| + |A \cap B \cap C|$

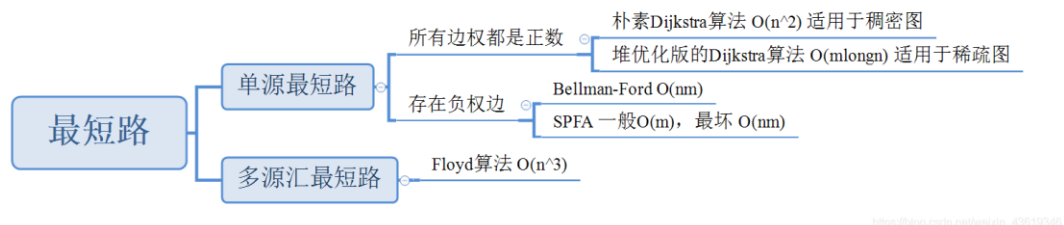


8.

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	Shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

注：基数排序的复杂度中， $r$ 代表关键字的基数， $d$ 代表长度， $n$ 代表关键字的个数。

10.



## 复赛练习

1. P1941

第一步：存数据和初始化

```

12 | cin>>n>>m>>k;
13 | for(int i=0;i<n;i++){
14 |     cin>>x[i]>>y[i]; //x数组存点击1次上升的高度，y数组存不点击下降的高度
15 | }
16 | for(int i=0;i<n;i++){ //初始化，每个管道的下边界为0，上边界为m+1
17 |     b[i][0]=0;
18 |     b[i][1]=m+1;
19 | }
20 | for(int i=1;i<=k;i++){
21 |     cin>>p>>l>>h;
22 |     b[p][0]=l,b[p][1]=h; //b[i][0]存下边界，b[i][1]存上边界
23 |     sum[p]=1;
24 | }
25 | for(int i=1;i<=n;i++){ //统计管道数量前缀和
26 |     sum[i]=sum[i-1];
27 | }
28 | memset(f,0x3f,sizeof f); //f[i][j]表示走到横坐标i，高度j需要点击的最少次数
29 | for(int i=0;i<=m;i++){ //起始高度可选，故不消耗点击次数
30 |     f[0][i]=0;
31 | }
  
```

第二步：上升的过程是一个完全背包

```
32 白 for(int i=1;i<=n;i++){
33      //上升的过程是一个完全背包，因为单位时间可以点击多次
34 白     for(int j=1;j<=m;j++){
35 白         if(j!=m){
36 白             if(j>x[i-1]){
37 白                 //f[i-1][j-x[i-1]]+1表示从前一个横坐标点击一次而来
38 白                 //f[i][j-x[i-1]]+1表示从前一个横坐标点击多次而来
39 白                 f[i][j]=min(f[i][j],min(f[i-1][j-x[i-1]]+1,f[i][j-x[i-1]]+1));
40 白             }
41 白         }
42 白     }
43 白     else{//j=m，处理边界情况
44 白         for(int k=m-x[i-1];k<=m;k++){
45 白             f[i][j]=min(f[i][j],min(f[i-1][k]+1,f[i][k]+1));
46 白         }
47 白     }
```

第三步：下降的过程是一个 01 背包

```
48      //下降的过程是一个01背包，即不点击下降，点击不下降
49 白     for(int j=b[i][0]+1;j<b[i][1];j++){
50 白         if(j+y[i-1]<=m){
51 白             f[i][j]=min(f[i][j],f[i-1][j+y[i-1]]);
52 白         }
53 白     //不属于管道范围内，重新赋值为INF
54 白     for(int j=0;j<=b[i][0];j++) f[i][j]=inf;
55 白     for(int j=b[i][1];j<=m;j++) f[i][j]=inf;
56 白 }
```

第四步：找最少点击次数 或 能通过的最大管道数量

```
57 白 int ans=inf;
58 白 for(int i=0;i<=m;i++){//寻找f[n][i]的最小值即为最少点击次数
59 白     ans=min(ans,f[n][i]);
60 白 }
61 白 if(ans!=inf){//可以到达终点，输出最少点击次数
62 白     cout<<1<<endl<<ans;
63 白 }
64 白 else{
65 白     cout<<0<<endl;
66 白     for(int i=n;i>=0;i--){ //到不到终点，逆序找最近能到达的管道
67 白         for(int j=0;j<=m;j++){
68 白             if(f[i][j]!=inf){
69 白                 cout<<sum[i];
70 白                 return 0;
71 白             }
72 白         }
73 白     }
74 白 }
75 白 return 0;
```

代码：

```

11 int main(){
12     cin>>n>>m>>k;
13     for(int i=0;i<n;i++){
14         cin>>x[i]>>y[i]; //x数组存点击1次上升的高度, y数组存不点击下降的高度
15     }
16     for(int i=0;i<n;i++){ //初始化, 每个管道的下边界为0, 上边界为m+1
17         b[i][0]=0;
18         b[i][1]=m+1;
19     }
20     for(int i=1;i<=k;i++){
21         cin>>p>>l>>h;
22         b[p][0]=l,b[p][1]=h; //b[i][0]存下边界, b[i][1]存上边界
23         sum[p]=1;
24     }
25     for(int i=1;i<=n;i++){ //统计管道数量前缀和
26         sum[i]+=sum[i-1];
27     }
28     memset(f,0x3f,sizeof f); //f[i][j]表示走到横坐标i, 高度j需要点击的最少次数
29     for(int i=0;i<=m;i++){ //起始高度可选, 故不消耗点击次数
30         f[0][i]=0;
31     }
32     for(int i=1;i<=n;i++){
33         //上升的过程是一个完全背包, 因为单位时间可以点击多次
34         for(int j=1;j<=m;j++){
35             if(j!=m){
36                 if(j>x[i-1]){
37                     //f[i-1][j-x[i-1]]+1表示从前一个横坐标点击一次而来
38                     //f[i][j-x[i-1]]+1表示从前一个横坐标点击多次而来
39                     f[i][j]=min(f[i][j],min(f[i-1][j-x[i-1]]+1,f[i][j-x[i-1]]+1));
40                 }
41             }

```

```

41         }
42         else{//j=m, 处理边界情况
43             for(int k=m-x[i-1];k<=m;k++){
44                 f[i][j]=min(f[i][j],min(f[i-1][k]+1,f[i][k]+1));
45             }
46         }
47     }
48     //下降的过程是一个01背包, 即不点击下降, 点击不下降
49     for(int j=b[i][0]+1;j<b[i][1];j++){
50         if(j+y[i-1]<=m){
51             f[i][j]=min(f[i][j],f[i-1][j+y[i-1]]);
52         }
53     }
54     //不属于管道范围内, 重新赋值为INF
55     for(int j=0;j<=b[i][0];j++) f[i][j]=inf;
56     for(int j=b[i][1];j<=m;j++) f[i][j]=inf;
57
58     int ans=inf;
59     for(int i=0;i<=m;i++){ //寻找f[n][i]的最小值即为最少点击次数
60         ans=min(ans,f[n][i]);
61     }
62     if(ans!=inf){ //可以到达终点, 输出最少点击次数
63         cout<<1<<endl<<ans;
64     }
65     else{
66         cout<<0<<endl;
67         for(int i=n;i>=0;i--){ //到不到终点, 逆序找最近能到达的管道
68             for(int j=0;j<=m;j++){
69                 if(f[i][j]!=inf){
70                     cout<<sum[i];
71                     return 0;
72                 }

```

## 2. P2296

解题思路：

第一步：正向反向存边，通过反图将终点能够到达的点标记

```

25 void bfs1(){ //标记：从终点能走到的其他点
26     queue<int> s;
27     s.push(endd);
28     flag[endd]=1;
29     while(!s.empty()){
30         int u=s.front();
31         s.pop();
32         for(int i=head2[u]; i!=0; i=G[i].next)
33         {
34             int to = G[i].to;
35             if(!flag[to])
36             {
37                 s.push(to);
38                 flag[to]=1;
39             }
40         }
41     }
42 }

```

第二步：对终点能够到达的点进行正向广搜，如果相连的点有不能到达终点的，此点删去（即不满足条件1）

```

43 void judge(){ //将上一步bfs1标记的点中“出边不能到达终点”的点删去，所以是正图
44     int fl;
45     for(int i=1; i<=n; i++)
46     {
47         fl=0;
48         if(!flag[i]) continue; //如果此点都不能到达终点，那么直接跳过
49         for(int j=head1[i]; j!=0; j=e[j].next)
50         {
51             int to=e[j].to;
52             if(!flag[to])
53             {
54                 fl=1; //出边的点不能到达终点，说明此点要被删去
55                 break;
56             }
57         }
58         if(fl) continue;
59         able[i]=1; //把最终能走的点标记下来
60     }
61 }

```



第三步：对保留下来的点进行广搜，找最短路

```
62 void bfs2(){ //bfs求最短路
63     queue <int> q;
64     q.push(startt);
65     dis[startt]=1;
66     while(!q.empty()){
67         int u=q.front();
68         q.pop();
69         if(u==endd){
70             cout << dis[u]-1;
71             return;
72         }
73         for(int i=head1[u]; i!=0; i=e[i].next)
74         {
75             int to=e[i].to;
76             if(!dis[to] && able[to])
77             {
78                 dis[to]=dis[u]+1;
79                 q.push(to);
80             }
81         }
82     }
```

代码：

```
25 void bfs1(){ //标记：从终点能走到的其他点
26     queue <int> s;
27     s.push(endd);
28     flag[endd]=1;
29     while(!s.empty()){
30         int u=s.front();
31         s.pop();
32         for(int i=head2[u]; i!=0; i=g[i].next)
33         {
34             int to = g[i].to;
35             if(!flag[to])
36             {
37                 s.push(to);
38                 flag[to]=1;
39             }
40         }
41     }
42 }
43 void judge(){ //将上一步bfs标记的点中“出边不能到达终点”的点删去，所以是正图
44     int fl;
45     for(int i=1;i<=n;i++)
46     {
47         fl=0;
48         if(!flag[i]) continue; //如果此点都不能到达终点，那么直接跳过
49         for(int j=head1[i]; j!=0; j=e[j].next)
50         {
51             int to=e[j].to;
52             if(!flag[to])
53             {
54                 fl=1; //出边的点不能到达终点，说明此点要被删去
55                 break;
56             }
57         }
58     }
```

```

56     }
57     }
58     if(f1) continue;
59     able[i]=1; //把最终能走的点标记下来
60 }
61 }
62 void bfs2(){ //bfs求最短路
63     queue<int> q;
64     q.push(startt);
65     dis[startt]=1;
66     while(!q.empty()){
67         int u=q.front();
68         q.pop();
69         if(u==endd){
70             cout << dis[u]-1;
71             return;
72         }
73         for(int i=head1[u]; i!=0; i=e[i].next)
74         {
75             int to=e[i].to;
76             if(!dis[to] && able[to])
77             {
78                 dis[to]=dis[u]+1;
79                 q.push(to);
80             }
81         }
82     }
83 }
84 int main(){
85     cin >> n >> m;
86     for(int i=1;i<=m;i++){
87         cin >> x >> y;
88         add1(x,y);
89         add2(y,x);
90     }
91     cin >> startt >> endd;
92     bfs1(); //通过反图标记能到达终点的点
93     judge(); //将出边指向的点不能到达终点的点删去
94     if(!able[startt]) //如果起点都没被标记, 那么直接输出-1
95     {
96         cout << -1;
97         return 0;
98     }
99     bfs2(); //求最短路
100    return 0;

```

3. P2312

$$= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

$$= (a_n x^{n-1} + a_{n-1} x^{n-2} + \cdots + a_2 x + a_1) x + a_0$$

$$= ((a_n x^{n-2} + a_{n-1} x^{n-3} + \cdots + a_3 x + a_2) x + a_1) x + a_0$$

⋮

$$= (\cdots ((a_n x + a_{n-1}) x + a_{n-2}) x + \cdots + a_1) x + a_0$$



求多项式的值时，首先计算最内层括号内一次多项式的值，即

$$v_1 = a_n * x + a_{n-1}$$

然后由内向外逐层计算一次多项式的值，即

$$v_2 = v_1 x + a_{n-2}$$

$$v_3 = v_2 x + a_{n-3}$$

$\vdots$

$$v_n = v_{n-1} x + a_0$$

这样，求n次多项式f(x)的值就转化为求n个一次多项式的值。

第一步：存高精度正数负数

```
16 int main(){
17     cin >> n >> m;
18     for(int i=0;i<=n;i++){
19         cin >> str;
20         int len=str.length();
21         if(str[0]=='-'){ //存高精度负数
22             for(int j=1;j<len;j++) a[i]=(a[i]*10+(str[j]-'0'))%mod;
23             a[i]=-a[i];
24         }
25         else{ //存高精度正数
26             for(int j=0;j<len;j++) a[i]=(a[i]*10+(str[j]-'0'))%mod;
27         }
28     }
```

第二步：遍历区间的每一个值，判断是不是多项式的解

```
7 //mod=998244353;数据比较大，对一个较大的质数取余
8 bool judge(long long x){ //判断x是不是多项式的解
9     long long v=(a[n]*x+a[n-1])%mod;
10    for(int i=n-2;i>=0;i--){
11        v=(v*x+a[i])%mod;
12    }
13    if(v==0) return 1;
14    return 0;
15 }

29 for(int i=1;i<=m;i++){ //遍历区间的每一个值
30     if(judge(i)){
31         cnt++;
32         ans[cnt]=i;
33     }
34 }
35 cout << cnt << endl;
36 for(int i=1;i<=cnt;i++){
37     cout << ans[i] << endl;
38 }
39 return 0;
40 }
```

代码：

```

7 //mod=998244353; 数据比较大, 对一个较大的质数取余
8 bool judge(long long x){ //判断x是不是多项式的解
9     long long v=(a[n]*x+a[n-1])%mod;
10    for(int i=n-2;i>=0;i--){
11        v=(v*x+a[i])%mod;
12    }
13    if(v==0) return 1;
14    return 0;
15 }

17 int main(){
18     cin >> n >> m;
19     for(int i=0;i<=n;i++){
20         cin >> str;
21         int len=str.length();
22         if(str[0]=='-'){ //存高精度负数
23             for(int j=1;j<len;j++) a[i]=(a[i]*10+(str[j]-'0'))%mod;
24             a[i]=-a[i];
25         }
26         else{ //存高精度正数
27             for(int j=0;j<len;j++) a[i]=(a[i]*10+(str[j]-'0'))%mod;
28         }
29     }
30     for(int i=1;i<=m;i++){ //遍历区间的每一个值
31         if(judge(i)){
32             cnt++;
33             ans[cnt]=i;
34         }
35     }
36     cout << cnt << endl;
37     for(int i=1;i<=cnt;i++){
38         cout << ans[i] << endl;
39     }
40     return 0;
}

```

#### 4. P2661

解题思路:

1、第一次有人从别人口中听得自己的信息?

求图最小环的长度

2、如何判断图中出现了环?

建立并查集, 连边的时候 (即合并祖先的时候) 发现两个点已经有公共祖先了, 就说明此时出现了环, 更新最短环长

第一步：建立并查集，连边（合并）

```
28 int main(){
29     cin >> n;
30     for(int i=1;i<=n;i++){ //建立并查集
31         f[i]=i;
32     }
33     for(int i=1;i<=n;i++){
34         cin >> t;
35         merge(i,t); //连边，即合并公共祖先
36     }
37     cout << minn;
38     return 0;
39 }
```

第二步：更新每个点的祖先，同时求出每个点到祖先的距离  
如果找到环，就更新最短环长

```
7 int find(int x){
8     if(f[x]!=x){
9         int temp = f[x];
10        f[x] = find(f[x]);
11        d[x] += d[temp]; //找祖先的同时更新 此点到祖先的距离
12    }
13    return f[x]; //找到祖先
14 }
15 void merge(int a, int b){
16     int x=find(a);
17     int y=find(b);
18     if (x!=y){
19         f[x]=y; //更新a的祖先
20         d[a]=d[b]+1; //更新a到祖先的距离
21     }
22     else{ //如果两个点有公共祖先，那么说明有环，更新环的最短边数
23         minn=min(minn,d[a]+d[b]+1);
24     }
25     return;
26 }
```

代码：

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int f[200010],d[200010];
4  int n, minn=INT_MAX;
5  int t;
6
7  int find(int x){
8      if(f[x]!=x){
9          int temp = f[x];
10         f[x] = find(f[x]);
11         d[x] += d[temp]; //找祖先的同时更新 此点到祖先的距离
12     }
13     return f[x]; //找到祖先
14 }
15 void merge(int a, int b){
16     int x=find(a);
17     int y=find(b);
18     if (x!=y){
19         f[x]=y; //更新a的祖先
20         d[a]=d[b]+1; //更新a到祖先的距离
21     }
22     else{ //如果两个点有公共祖先, 那么说明有环, 更新环的最短边数
23         minn=min(minn,d[a]+d[b]+1);
24     }
25     return;
26 }
27
28 int main(){
29     cin >> n;
30     for(int i=1;i<=n;i++){ //建立并查集
31         f[i]=i;
32     }
33     for(int i=1;i<=n;i++){
34         cin >> t;
35         merge(i,t); //连边, 即合并公共祖先
36     }
37     cout << minn;
38     return 0;
39 }

```