

ACM Template

Rien

August 11, 2021



rien_zhu@163.com

Contents

1	字符串	1
1.1	KMP	1
2	数据结构	2
2.1	并查集	2
2.1.1	优化并查集	2
2.2	01 字典树	3
2.3	树状数组	4
2.3.1	一维树状数组	4
2.3.2	求逆序对	6
3	数学	7
3.1	快速幂	7
3.1.1	数字快速幂	7
3.2	组合数学	7
3.3	莫比乌斯反演	9
3.3.1	整除分块	9
3.3.2	莫比乌斯函数	10
3.4	BSGS	10
3.5	中国剩余定理	11
4	图论	13
4.1	BFS	13
4.1.1	伪代码	13
4.1.2	BFS 记录距离和路径	13
4.2	最短路	14
4.2.1	Dijkstra 算法单源最短路	14
4.2.2	bellman_ford 算法单源最短路	15
4.2.3	Floyd 任意两点最短路径	17
4.3	最小生成树	18
4.3.1	Prim 算法	18
4.3.2	Kruskal 算法	19
5	动态规划	21
5.1	悬线法	21
5.2	LIS 最长上升子序列	22
5.3	背包	23
5.3.1	基础背包	23
6	杂项	25
6.1	输入输出	25
6.1.1	快读	25
6.1.2	关闭同步	25

6.1.3	快读快写	25
6.1.4	7.19 所用板子	26
6.2	高精度	27
6.2.1	int128	27
6.2.2	简单高精度	27
6.2.3	压位高精度	30
6.3	离散化	32
6.4	常用公式	33
7	经典例题	34
7.1	某天是星期几	34
8	注意事项	35

1 字符串

1.1 KMP

```
1  /*
2  Next[] 的含义:  $x[i-Next[i] \dots i-1] = x[0 \dots Next[i]-1]$ 
3  Next[i] 为满足:  $x[i-z \dots i-1] = x[0 \dots z-1]$  的最大值
4  */
5  void kmp_pre(char x[], int m, int next[]) {
6      int i, j;
7      j = next[0] = -1;
8      i = 0;
9      while(i < m) {
10         while(-1 != j && x[i] != x[j])
11             j = next[j];
12         next[++i] = ++j;
13     }
14 }
15 const int N = 1000005;
16 int Next[N];
17 /*
18 返回 x 在 y 中出现的次数, 可以重叠
19 y 是主串, x 是模式串
20 */
21 int KMP_Count(char x[], int m, char y[], int n) {
22     int i, j;
23     int ans = 0;
24     kmp_pre(x, m, Next);
25     i = j = 0;
26     while(i < n) {
27         while(-1 != j && y[i] != x[j])
28             j = Next[j];
29         i++; j++;
30         if(j >= m) {
31             ans++;
32             j = Next[j];
33         }
34     }
35     return ans;
36 }
```

2 数据结构

2.1 并查集

2.1.1 优化并查集

```
1  /*
2  路径压缩并查集,
3  */
4  const int N=1e5+5;
5  int fa[N];
6  void init(){
7      for(int i=1;i<=N;++i)
8          fa[i]=i;
9  }
10 int find(int x){
11     int temp=x;
12     while(temp!=fa[temp])
13         temp=fa[temp];
14     while(x!=fa[x]){
15         x=fa[x];
16         fa[x]=temp;
17     }
18     return temp;
19 }
20 void union(int x,int y){
21     fa[find(x)]=find(y);
22 }
23 /*
24 按秩合并并查集, 不破坏树形结构
25 连通块数量为 block, 大小为 size
26 */
27 int f[N],size[N],block;
28 void Init(){
29     for(int i=1;i<=N;++i)
30         f[i]=0;
31     block=n;
32 }
33 int Find(int x){
34     if(!f[x]) f[x]=x,size[x]=1;
35     if(f[x]==x) return x;
36     return f[x]=Find(f[x]);
37 }
38 void Union(int x,int y){
39     x=Find(x);
```

```

40     y=Find(y);
41     if(x=y) return ;
42     if(size[x]>size[y]) swap(x,y);
43     f[x]=y;
44     size[y]+=size[x];
45     block--;
46 }

```

2.2 01 字典树

```

1  /*
2  01trie 树
3  在一组数中找跟某个数异或结果最大的数
4  */
5  const int MAXN=100005;
6  //MAXN 右移需根据题目判断, 如数据范围为 int 则 *32 即右移 5 位
7  int val[MAXN<<5];          //val[i]=j 表示编号为 i 的节点的值为 j
8  int cnt;                   //节点数量
9  int tree[MAXN<<5][2];      //tree[i][0]=j 表示编号为 i 的节点的 0
    ↳ 子节点的编号为 j
10 void init(){
11     cnt=1;
12     tree[0][0]=tree[0][1]=0;
13 }
14 void insert(int x){
15     int v,u=0;
16     for(int i=31;i>=0;--i){
17         v=(x>>i)&1;
18         if(!tree[u][v]){
19             tree[cnt][0]=tree[cnt][1]=0; //初始化新节点的子节点
20             val[cnt]=0;                  //节点值为 0, 表示到此
                ↳ 不是一个数
21             tree[u][v]=cnt++;           //指向该节点
22         }
23         u=tree[u][v]; //到下一节点
24     }
25     val[u]=x;
26 }
27 int query(int x){
28     int v,u=0;
29     for(int i=31;i>=0;i--){
30         v=(x>>i)&1;
31         if(tree[u][v^1]) //利用贪心策略, 优先寻找和当前位不同的数
32             u=tree[u][v^1];

```

```
33         else
34             u=tree[u][v];
35     }
36     return val[u];
37 }
38 int main(){
39     int n,m;scanf("%d%d",&n,&m);
40     init();
41     int tmp;
42     for(int i=0;i<n;++i){
43         scanf("%d",&tmp);
44         insert(tmp);
45     }
46     for(int i=0;i<m;++i){
47         scanf("%d",&tmp);
48         printf("%d\n",query(tmp));
49     }
50     return 0;
51 }
```

2.3 树状数组

2.3.1 一维树状数组

```
1  /*
2  BIT 一维树状数组
3  */
4  const int N=1000005;
5  ll tree[N],a[N],b[N];
6  int n,m,op,l,r,v;
7  //单点修改区间查询
8  void add(int p,int x){
9      for(int i=p;i<=n;i+=i&-i)
10         tree[i]+=x;
11 }
12 ll query(int p){
13     ll ret=0;
14     for(int i=p;i;i-=i&-i)
15         ret+=tree[i];
16     return ret;
17 }
18 int main(){
19     //读入数据
20     for(int i=1;i<=n;++i){
```

```
21         cin>>v;
22         add(i,v);
23     }
24     add(l,v); //将第 l 个数加上 v
25     cout<<query(r)-query(l-1)<<endl; //询问 [l-r] 区间的和
26     return 0;
27 }
28
29 //区间修改单点查询
30 void add(int p,int x){
31     for(int i=p;i<=n;i+=i&-i)
32         tree[i]+=x;
33 }
34 ll query(int p){
35     ll ret=0;
36     for(int i=p;i;i-=i&-i)
37         ret+=tree[i];
38     return ret;
39 }
40 int main(){
41     //读入数据, 维护差分数组
42     for(int i=1;i<=n;++i){
43         cin>>a[i];
44         add(i,a[i]-a[i-1]);
45     }
46     add(l,v),add(r+1,-v); //将 [l-r] 加上 v
47     cout<<query(l)<<endl; //询问第 l 个数的值
48     return 0;
49 }
50
51 //区间修改区间查询
52 void add(int p,int x){
53     for(int i=p;i<=n;i+=i&-i){
54         b[i]+=(ll)x;
55         tree[i]+=(ll)p*x;
56     }
57 }
58 ll query(int p){
59     ll ret=0;
60     for(int i=p;i;i-=i&-i)
61         ret+=1ll*(p+1)*b[i]-tree[i];
62     return ret;
63 }
64 int main(){
```



```
65     //读入数据, 维护差分数组
66     for(int i=1;i<=n;++i){
67         cin>>a[i];
68         add(i,a[i]-a[i-1]);
69     }
70     add(1,v),add(r+1,-v);//将 [l-r] 加上 v
71     cout<<query(r)-query(1-1)<<endl;//询问 [l-r] 区间的和
72     return 0;
73 }
```

2.3.2 求逆序对

```
1  //树状数组求逆序对
2  void modify(int x, int d){
3      for(int i = x; i <= n; i += i&-i)c[i] += d;
4  }
5  int getsum(int x){
6      int sum = 0;
7      for(int i = x; i >= 1; i -= i&-i)sum += c[i];
8      return sum;
9  }
10 int main(){
11     scanf("%d", &n);
12     for(int i = 1; i <= n; i ++){
13         scanf("%d", &x);
14         ans += getsum(n) - getsum(x);
15         modify(x, 1);
16     }
17 }
```

3 数学

3.1 快速幂

3.1.1 数字快速幂

```

1 //a 的 b 次方对 p 取余
2 //可能 a,b 都比较大, 所以需要自己另写乘法
3 ll Mul(ll x, ll y, ll P){
4     ll tmp=(x*y-(ll)((long double)x/P*y+1.0e-8)*P);
5     return (tmp+P)%P;
6 }
7 ll ksm(ll a, ll b, ll p){
8     ll ret=1;
9     while(b){
10         if(b&1) ret=Mul(ret,a,p);
11         a=Mul(a,a,p);
12         b>>=1;
13     }
14     return ret;
15 }
```

3.2 组合数学

```

1 /*
2 组合数学求法:
3 1. 杨辉三角打表
4 所需空间:  $O(nm)$ , 时间: 预处理  $O(nm)$ , 查询  $O(1)$ 
5 */
6 const int mod = 1e9+7;
7 int c[maxn][maxn];
8 void init(int n){
9     c[0][0]=1;
10     rep(i,1,n){
11         c[i][0]=1;
12         rep(j,1,n+1) c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
13     }
14 }
15
16 /*
17 2. 直接用公式
18  $Cnm = n! / (m!(n-m)!)$ 
19 不能取模, 时间: 预处理  $O(1)$ , 查询  $O(n)$ 
20 */
21 ll C(ll n, ll m)
```

```

22 {
23     if (n < m) return 0;
24     ll ret = 1;
25     rep(i, n - m + 1, n)
26         ret *= i;
27     rep(i, 2, m)
28         ret /= i;
29     return ret;
30 }
31
32 /*
33 3. 乘法逆元
34 可以取模, 适用于  $P$  比较大的题目
35 所需空间  $O(n)$ , 时间: 预处理  $O(n)$ , 查询  $O(1)$ 
36 */
37 //  $N$  的范围需要根据题目确定
38 int fac[N+5], inv[N+5];
39 ll qpow(ll bsc, ll y){
40     ll ret = 1;
41     while(y){
42         if(y&1) ret = ret*bsc%mod;
43         bsc = bsc*bsc%mod;
44         y >>= 1;
45     }
46     return ret;
47 }
48 void init(){
49     fac[0] = 1;
50     for(int i=1; i<=N; i++)
51         fac[i] = (ll)fac[i-1]*i%mod;
52     inv[N] = qpow(fac[N], mod-2);
53     for(int i=N-1; i>=0; i--)
54         inv[i] = (ll)inv[i+1]*(i+1)%mod;
55 }
56 int C(int n, int m){
57     if(m>n) return 0;
58     return (ll)fac[n]*inv[m]%mod*inv[n-m]%mod;
59 }
60
61 /*
62 4. Lucas 定理
63 模数必须是质数,  $n$  比较大,  $p$  比较小, 不能通过预处理阶乘和逆元来计算
64 lucas( $n, m, p$ ) 返回  $C(n, m)$  对  $p$  取模的结果

```

```

65  */
66  ll f[N], rf[N];
67  ll mul(ll x, ll y, ll p){
68      return (x*y-(ll)(x/(long double)p*y+1e-3)*p+p)%p;
69  }
70  ll pow(ll a, ll b, ll p){
71      ll t=1;
72      for(; b>=1; a=mul(a,a,p))
73          if(b & 1) t=mul(t,a,p);
74      return t;
75  }
76  //使用时为 init(p)
77  void init(int p){
78      f[0]=1;
79      for(int i=1; i<p; ++i) f[i]=f[i-1]*i%p;
80      rf[p-1]=pow(f[p-1], p-2, p);
81      for(int i=p-1; i; i--) rf[i-1]=rf[i]*i%p;
82  }
83  ll C(int n, int m, int mod){
84      if(m>n || m<0 || n<0) return 0;
85      return f[n]*rf[m]%mod*rf[n-m]%mod;
86  }
87  ll lucas(ll n, ll m, ll p){
88      if(n<m) return 0;
89      if(!m || n==m) return 1;
90      return C(n%p, m%p, p) *lucas(n/p, m/p, p)%p;
91  }

```

3.3 莫比乌斯反演

3.3.1 整除分块

可以用到整除分块的形式，大致是这样的：

$$\sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor$$

用整除分块可以在 $O(\sqrt{n})$ 求出，代码如下：

```

1  for(int l=1, r; l<=n; l=r+1){
2      r=n/(n/l);
3      ans+=(r-l+1)*(n/l);
4  }

```

3.3.2 莫比乌斯函数

```

1  const int MAXN=1000000;
2  bool check[MAXN+10];
3  int prime[MAXN+10];
4  int mu[MAXN+10];
5  void Moblus(){
6      memset(check,false,sizeof(check));
7      mu[1]=1;
8      int tot=0;
9      for(int i=2;i<=MAXN;i++){
10         if(!check[i]){
11             prime[tot++]=i;
12             mu[i]=-1;
13         }
14         for(int j=0;j<tot;j++){
15             if(i*prime[j]>MAXN) break;
16             check[i*prime[j]]=true;
17             if(i%prime[j]==0){
18                 mu[i*prime[j]]=0;
19                 break;
20             }
21             else mu[i*prime[j]]=-mu[i];
22         }
23     }
24 }

```

3.4 BSGS

```

1  //a^x = b (mod n) n 是素数和不是素数都行
2  //求解上式 0 <= x < n
3  //调用函数的时候记得 b=b%n
4  #define MOD 76543
5  int hs[MOD],head[MOD],Next[MOD],id[MOD],top;
6  void insert(int x,int y){
7      int k=x%MOD;
8      hs[top]=x,id[top]=y,Next[top]=head[k],head[k]=top++;
9  }
10 int find(int x){
11     int k=x%MOD;
12     for(int i=head[k];i!=-1;i=Next[i])
13         if(hs[i]==x) return id[i];
14     return -1;
15 }

```

```

16 int BSGS(int a,int b,int n){
17     memset(head,-1,sizeof(head));
18     top=1;
19     if(b==1) return 0;
20     int m=sqrt(n*1.0),j;
21     long long x=1,p=1;
22     for(int i=0;i<m;++i,p=p*a%n) insert(p*b%n,i);
23     for(long long i=m;;i+=m){
24         if((j=find(x=x*p%n))!=-1) return i-j;
25         if(i>n) break;
26     }
27     return -1;
28 }

```

3.5 中国剩余定理

中国剩余定理 (Chinese Remainder Theorem, CRT) 可求解如下形式的一元线性同余方程组 (其中 n_1, n_2, \dots, n_k 两两互质):

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_k \pmod{n_k} \end{cases}$$

算法流程

1. 计算所有模数的积 n ;
2. 对于第 i 个方程:
 - (a) 计算 $m_i = \frac{n}{n_i}$;
 - (b) 计算 m_i 在模 n_i 意义下的 [逆元](./inverse.md) m_i^{-1} ;
 - (c) 计算 $c_i = m_i m_i^{-1}$ (不要对 n_i 取模)。
3. 方程组的唯一解为: $x = \sum_{i=1}^k a_i c_i \pmod{n}$ 。

代码如下:

```

1 LL CRT(int k, LL* a, LL* r) {
2     LL n = 1, ans = 0;
3     for (int i = 1; i <= k; i++) n = n * r[i];
4     for (int i = 1; i <= k; i++) {
5         LL m = n / r[i], b, y;
6         exgcd(m, r[i], b, y); // b * m mod r[i] = 1
7         ans = (ans + a[i] * m * b % mod) % mod;

```

```
8 | }  
9 | return (ans % mod + mod) % mod;  
10 | }
```

4 图论

4.1 BFS

4.1.1 伪代码

```

1  /*
2  宽度优先，就是每次都尝试访问同一层的节点。如果同一层都访问完了，
   ↳ 再访问下一层。
3  这样做的结果是，BFS 算法找到的路径是从起点开始的最短合法路径。换
   ↳ 言之，这条路所包含的边数最小。
4  在 BFS 结束时，每个节点都是通过从起点到该点的最短路径访问的。
5  */
6  bfs(s) {
7      q = new queue()
8      q.push(s), visited[s] = true
9      while (!q.empty()) {
10         u = q.pop()
11         for each edge(u, v) {
12             if (!visited[v]) {
13                 q.push(v)
14                 visited[v] = true
15             }
16         }
17     }
18 }
```

4.1.2 BFS 记录距离和路径

```

1  /*
2  队列 Q 记录要处理的节点，vis 数组来标记某个节点是否已经访问过。
3  d 数组记录某个点到起点的距离，可以得到起点到一个点的距离。
4  p 数组是记录从起点到这个点的最短路上的上一个点，可以方便地还原出
   ↳ 起点到一个点的最短路径。
5  restore(x) 输出的是从起点到 x 这个点所经过的点。
6
7  开始的时候，我们把起点 s 以外的节点的 vis 值设为 0，意思是没有访
   ↳ 问过。然后把起点 s 放入队列 Q 中。
8  之后，我们每次从队列 Q 中取出队首的点 u，把 u 相邻的所有点 v 标记
   ↳ 为已经访问过了并放入队列 Q。
9  直到某一时刻，队列 Q 为空，这时 BFS 结束。
10
11  时间复杂度  $O(n + m)$ 
12  空间复杂度  $O(n)$  (vis 数组和队列)
13  */
```



```

14 void bfs(int u) {
15     while (!Q.empty()) Q.pop();
16     Q.push(u);
17     vis[u] = 1;
18     d[u] = 0;
19     p[u] = -1;
20     while (!Q.empty()) {
21         u = Q.front();
22         Q.pop();
23         for (int i = head[u]; i; i = e[i].x) {
24             if (!vis[e[i].t]) {
25                 Q.push(e[i].t);
26                 vis[e[i].t] = 1;
27                 d[e[i].t] = d[u] + 1;
28                 p[e[i].t] = u;
29             }
30         }
31     }
32 }
33 void restore(int x) {
34     vector<int> res;
35     for (int v = x; v != -1; v = p[v]) {
36         res.push_back(v);
37     }
38     std::reverse(res.begin(), res.end());
39     for (int i = 0; i < res.size(); ++i) printf("%d", res[i]);
40     puts("");
41 }

```

4.2 最短路

4.2.1 Dijkstra 算法单源最短路

```

1  /*
2  Dijkstra 算法 + 堆优化, 复杂度为  $O(E\log E)$ 
3  注意初始化
4  权值必须是非负
5  */
6  const int N=100010,M=1000010;
7  int head[N],ver[M],edge[M],Next[M],d[N];
8  bool v[N];
9  int n,m,tot;
10 //大根堆 (优先队列),pair 的第二维为节点编号
11 //pair 的第一维为 dist 的相反数 (利用相反数变成小根堆)

```

```

12 priority_queue<pair<int,int> > q;
13 void add(int x,int y,int z){
14     ver[++tot]=y,edge[tot]=z,Next[tot]=head[x],head[x]=tot;
15 }
16 void dijkstra(){
17     memset(d,0x3f,sizeof(d));//dist 数组
18     memset(v,0,sizeof(v));//节点标记
19     //起始点为 1, 若更改要改两处
20     d[1]=0;
21     q.push(make_pair(0,1));
22     while(q.size()){
23         int x=q.top().second;q.pop();
24         if(v[x]) continue;
25         v[x]=1;
26         for(int i=head[x];i;i=Next[i]){
27             int y=ver[i],z=edge[i];
28             if(d[y]>d[x]+z){
29                 d[y]=d[x]+z;
30                 q.push(make_pair(-d[y],y));
31             }
32         }
33     }
34 }
35 int main()
36 {
37     cin>>n>>m;
38     for(int i=1;i<=m;++i){
39         int x,y,z;
40         scanf("%d%d%d",&x,&y,&z);
41         //如果是无向图还要 add(y,x,z)
42         add(x,y,z);
43     }
44     dijkstra();
45     //求单源最短路
46     for(int i=1;i<=n;++i)
47         printf("%d\n",d[i]);
48     return 0;
49 }

```

4.2.2 bellman_ford 算法单源最短路

```

1  /*
2  单源最短路 bellman_ford 算法, 复杂度 O(VE)
3  可以处理负边权图

```

```

4  可以判断是否存在负环回路。
5  返回 true: 当且仅当图中不包含从源点可达的负权回路
6  vector<Edge> E; 先 E.clear() 初始化, 然后加入所有边
7  点的编号从 1 开始
8  */
9  const int INF=0x3f3f3f3f;
10 const int N=2550;
11 int dist[N];
12 struct Edge
13 {
14     int u,v;
15     int cost;
16     Edge(int _u=0,int _v=0,int
        ↪ _cost=0):u(_u),v(_v),cost(_cost){}
17 };
18 vector<Edge>E;
19 //点的编号从 1 开始
20 void addedge(int u,int v,int w){
21     E.push_back(Edge(u,v,w));
22 }
23 bool bellman_ford(int start,int n){
24     for(int i=1;i<=n;++i)    dist[i]=INF;
25     dist[start]=0;
26     for(int i=1;i<n;++i){
27         bool flag=false;
28         for(int j=0;j<E.size();j++){
29             int u=E[j].u;
30             int v=E[j].v;
31             int cost=E[j].cost;
32             if(dist[v]>dist[u]+cost){
33                 dist[v]=dist[u]+cost;
34                 flag=true;
35             }
36         }
37         if(!flag) return true;//没有负环回路
38     }
39     for(int j=0;j<E.size();j++)
40         if(dist[E[j].v]>dist[E[j].u]+E[j].cost)
41             return false;//有负环回路
42     return true;//没有负环回路
43 }
44 int main()
45 {

```

```

46     int n,m;cin>>n>>m;
47     E.clear();
48     for(int i=1;i<=m;++i){
49         int x,y,z;
50         scanf("%d%d%d",&x,&y,&z);
51         addedge(x,y,z);
52         addedge(y,x,z);
53     }
54     bellman_ford(s,n);
55     for(int i=1;i<=n;++i)
56         printf("%d\n",dist[i]);
57     return 0;
58 }

```

4.2.3 Floyd 任意两点最短路径

```

1  /*
2  Floyd 算法 计算任意两点间的距离 复杂度为  $O(N^3)$ 
3  */
4  #define inf 0x3f3f3f3f
5  int dist[1000][1000];
6  int main()
7  {
8      int k,i,j,n,m;///n 表示顶点个数, m 表示边的条数
9      scanf("%d %d",&n,&m);
10     for(i=1; i<=n; i++){//初始化
11         for(j=1; j<=n; j++){
12             if(i==j) dist[i][j]=0;
13             else dist[i][j]=inf;
14         }
15     }
16     int a,b,c;
17     for(i=1; i<=m; i++){//有向图
18         scanf("%d %d %d",&a,&b,&c);
19         dist[a][b]=min(dist[a][b],c);
20     }
21     for(k=1; k<=n; k++){    //Floyd-Warshall 算法核心语句
22         for(i=1; i<=n; i++){
23             for(j=1; j<=n; j++){
24
25                 ⇨ dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
26             }
27         }
28     }
29     for(i=1; i<=n; i++){//输出最终的结果, 最终二维数组中存的即
30         ⇨ 使两点之间的最短距离
31     }

```

```

27         printf("%10d",dist[i][j]);
28     printf("\n");
29 }
30 return 0;
31 }

```

4.3 最小生成树

无向图中，其某个子图中任意两个顶点都互相连通并且是一棵树，称之为生成树；若每个顶点有权值，则其权值和最小的生成树为最小生成树。

4.3.1 Prim 算法

```

1  /*
2  Prim 求 MST
3  耗费矩阵 cost[][MAXN], 标号从 0 开始, 0~n-1 注意注意!!!
4  返回最小生成树的权值, 返回-1 则表示原图不连通
5  */
6  const int INF=0x3f3f3f3f;
7  const int MAXN=110;
8  bool vis[MAXN];
9  int lowc[MAXN];
10 //点从 0 到 n-1
11 int Prim(int cost[][MAXN],int n){
12     int ans=0;
13     memset(vis,0,sizeof vis);
14     vis[0]=true;
15     for(int i=1;i<n;++i) lowc[i]=cost[0][i];
16     for(int i=1;i<n;++i){
17         int minc=INF;
18         int p=-1;
19         for(int j=0;j<n;++j){
20             if(!vis[j]&&minc>lowc[j]){
21                 minc=lowc[j];
22                 p=j;
23             }
24         }
25         if(minc==INF) return -1;//原图不连通
26         ans+=minc;
27         vis[p]=true;
28         for(int j=0;j<n;++j){
29             if(!vis[j] && lowc[j]>cost[p][j])
30                 lowc[j]=cost[p][j];
31         }

```

```

32     }
33     return ans;
34 }
35 int main()
36 {
37     int cost[MAXN][MAXN];
38     //注意对耗费矩阵赋初值,memset 只能读 1 个字符,这样和都赋给
    ↪ INF 结果一样
39     memset(cost,0x3f,sizeof(cost));
40     while (k--){
41         int u,v,w;
42         cin>>u>>v>>w;
43         //读入数据,注意题目给出的节点是否从 0 开始且无向边要两
    ↪ 次赋值
44         u--;
45         v--;
46         cost[u][v]=w;
47         cost[v][u]=w;
48     }
49     cout<<Prim(cost,n)<<endl;
50     return 0;
51 }

```

4.3.2 Kruskal 算法

```

1  /*
2  Kruskal 算法求 MST
3  */
4  //根据题目调试最大点数和边数
5  const int MAXN=1100; //最大点数
6  const int MAXM=200005; //最大边数
7  int F[MAXN]; //并查集
8  struct Edge {
9      int u,v,w;
10 }edge[MAXM]; //储存边的信息: 起点, 终点, 权值
11 int tol=0; //边数, 记得赋值为 0
12 void addedge(int u,int v,int w){ //加边
13     edge[tol].u=u;
14     edge[tol].v=v;
15     edge[tol++].w=w;
16 }
17 bool cmp(Edge a,Edge b){ //排序
18     return a.w<b.w;
19 }

```

```
20 int find(int x){
21     return F[x]==x?x:F[x]=find(F[x]);
22 }
23 //传入点数，返回最小生成树权值，如果不连通返回-1
24 int Kruskal(int n){
25     //根据点的编号（从 0 或从 1 开始）初始并查集
26     for(int i=1;i<=n;++i) F[i]=i;
27     sort(edge,edge+tol,cmp);
28     int cnt=0;//计算加入边数
29     int ans=0;
30     for(int i=0;i<tol;++i){
31         int u=edge[i].u;
32         int v=edge[i].v;
33         int w=edge[i].w;
34         int t1=find(u);
35         int t2=find(v);
36         if(t1!=t2){
37             ans+=w;
38             F[t1]=t2;
39             cnt++;
40         }
41         if(cnt==n-1) break;
42     }
43     if(cnt<n-1) return -1;//不连通
44     else return ans;
45 }
46 int main()
47 {
48     int n,m;cin>>n>>m;//点数和边数
49     for(int i=0;i<m;++i){
50         int u,v,w;cin>>u>>v>>w;
51         addedge(u,v,w);//只用读一次就行
52     }
53     cout<<Kruskal(n);//返回最小生成树权值
54     return 0;
55 }
```

5 动态规划

5.1 悬线法

悬线法的用途：针对求给定矩阵中满足某条件的极大矩阵，比如“面积最大的长方形、正方形”“周长最长的矩形等等”。

悬线法的基本思路：维护三个二维数组，Left, Right, Up 数组。

Left 数组存储从 map[i][j] 这个点出发，满足条件能到达的最左边地方。

Right 数组存储从 map[i][j] 这个点出发，满足条件能到达的最右边地方。

Up 数组存储从这点以上满足条件的能到达的最大长度。

递推公式：

Up: $Up[i][j] = Up[i-1][j] + 1$

Right: $\min(Right[i][j], Right[i-1][j])$

Left: $\max(Left[i][j], Left[i-1][j])$

```

1  /*
2  悬线法求最大全 1 子矩阵
3  */
4  const int MAXN=2005;
5  char x[MAXN][MAXN];
6  int y[MAXN][MAXN], l[MAXN][MAXN], r[MAXN][MAXN], up[MAXN][MAXN];
7  int main(){
8      int n,m; cin>>n>>m;
9      for(int i=1;i<=n;++i){
10         for(int j=1;j<=m;++j){
11             cin>>x[i][j];
12             //将输入处理成 01 矩阵
13             if(x[i][j]=='F'){
14                 //此处赋值需注意，只有 1 处才赋值，否则全 0 矩
15                 //阵也会输出面积为 1
16                 y[i][j]=1;
17                 l[i][j]=j;
18                 r[i][j]=j;
19                 up[i][j]=1;
20             }
21             else
22                 y[i][j]=0;
23         }
24     }
25     //按行处理
26     for(int i=1;i<=n;++i){
27         for(int j=2;j<=m;++j){
28             if(y[i][j-1] && y[i][j])
29                 l[i][j]=l[i][j-1];

```



```

30         for(int j=m-1;j>=1;--j){
31             if(y[i][j+1] && y[i][j])
32                 r[i][j]=r[i][j+1];
33         }
34     }
35     int len,ans=0;
36     for(int i=1;i<=n;++i){
37         for(int j=1;j<=m;++j){
38             //i>1 很关键, 用于处理只有 1 行的情况
39             if(i>1 && y[i-1][j] && y[i][j]){
40                 up[i][j]=up[i-1][j]+1;
41                 l[i][j]=max(l[i][j],l[i-1][j]);
42                 r[i][j]=min(r[i][j],r[i-1][j]);
43             }
44             len=r[i][j]-l[i][j]+1;
45             ans=max(ans,len*up[i][j]);
46         }
47     }
48     cout<<ans<<endl;
49     return 0;
50 }

```

5.2 LIS 最长上升子序列

```

1  /*
2  动态规划求最长上升子序列, 复杂度为  $O(n \log n)$ 
3  */
4  int DP(int n){
5      int i,len,pos;
6      b[1]=a[1];
7      len=1;
8      for(i=2;i<=n;++i){
9          //a[i] 大则可直接插到后面
10         if(a[i]>=b[len]){
11             len=len+1;
12             b[len]=a[i];
13         }
14         //二分法在 b 数组中找出第一个比 a[i] 大的位置并让 a[i]
15         ↪ 代替
16         else {
17             pos=lower_bound(b+1,b+1+len,a[i])-b;
18             b[pos]=a[i];
19         }
20     }
21 }

```

```

20     return len;
21 }

```

5.3 背包

5.3.1 基础背包

```

1  int nValue,nKind;
2  const int N=100005;
3  int dp[N],cost[N],value[N],num[N];
4  //0-1 背包, 代价为 cost, 价值为 value
5  void ZeroOnePack(int cost,int value){
6      for(int i=nValue;i>=cost;i--)
7          dp[i]=max(dp[i],dp[i-cost]+value);
8  }
9  //完全背包, 代价为 cost, 价值为 value
10 void CompletePack(int cost,int value){
11     for(int i=cost;i<=nValue;++i)
12         dp[i]=max(dp[i],dp[i-cost]+value);
13 }
14 //多重背包, 代价为 cost, 价值为 value, 数量为 amount
15 void MultiplePack(int cost,int value,int amount){
16     if(cost*amount>=nValue)
17         CompletePack(cost,value);
18     else {
19         int k=1;
20         while(k<amount){
21             ZeroOnePack(k*cost,k*value);
22             amount-=k;
23             k<<=1;
24         }
25         ZeroOnePack(amount*cost,amount*value);
26     }
27 }
28 int main(){
29     //记得清空数组
30     memset(dp,0,sizeof dp);
31     //0-1 背包
32     rep(i,1,nKind)
33         ZeroOnePack(cost[i],value[i]);
34     printf("%d\n",dp[nValue]);
35     //完全背包
36     rep(i,1,nKind)
37         CompletePack(cost[i],value[i]);

```

```
38     printf("%d\n",dp[nValue]);
39     //多重背包
40     rep(i,1,nKind)
41         MultiplePack(cost[i],value[i],num[i]);
42     printf("%d\n",dp[nValue]);
43     //分组背包
44     rep(k,1,ts)//循环每一组
45         for(int i=nValue;i>=0;--i)//循环背包容量
46             rep(j,1,cnt[k])//循环该组的每一个物品
47                 if (i >= cost[t[k][j]])
48                     ↪ dp[i]=max(dp[i],dp[i-cost[t[k][j]]]+value[t[k][j]]);
49                     //像 0-1 背包一样状态转移
50     return 0;
51 }
```

6 杂项

6.1 输入输出

6.1.1 快读

```

1 //快读
2 template <typename T> T &read(T &r) {
3     r = 0; bool w = 0; char ch = getchar();
4     while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch =
        ↪ getchar();
5     while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch
        ↪ ^ 48), ch = getchar();
6     return r = w ? -r : r;
7 }
8 //用法:
9 read(n);

```

6.1.2 关闭同步

```

1 //关闭同步
2 ios::sync_with_stdio(0);
3 cin.tie(0);

```

6.1.3 快读快写

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //快读
4 inline int read() {
5     int num=0, w=0;
6     char ch=0;
7     while (!isdigit(ch)) {
8         w|=ch=='-';
9         ch = getchar();
10    }
11    while (isdigit(ch)) {
12        num = (num<<3) + (num<<1) + (ch^48);
13        ch = getchar();
14    }
15    return w? -num: num;
16 }
17 //快写
18 inline void write(int x)
19 {

```

```

20     if(x<0) {
21         putchar('-');
22         x = -x;
23     }
24     if(x>9) write(x / 10);
25     putchar(x % 10 + '0');
26 }
27
28
29 int main(){
30     int a;
31     a = read();    //读入到 t 中
32     write(t);      //输出 t
33     putchar('\n');
34 }

```

6.1.4 8.11 所用板子

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define ll long long
4  #define ull unsigned long long
5  #define ld long double
6  #define rep(i, s, t) for (int i = (int)(s); i <= (int)(t); ++
   ↪ i)
7  #define debug(x) cerr<<#x<<" = "<<(x)<<endl
8  template <typename T> T &read(T &r) {
9      r = 0; bool w = 0; char ch = getchar();
10     while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch =
   ↪ getchar();
11     while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch
   ↪ ^ 48), ch = getchar();
12     return r = w ? -r : r;
13 }
14 ll mod=1e9+7;
15 ll INF=1e15;
16 ll Inf=0x3f3f3f3f;
17 double pi=acos(-1.0);
18
19
20 int main()
21 {
22     ios::sync_with_stdio(0);
23     cin.tie(0);

```

```
24 |
25 |     return 0;
26 | }
```

6.2 高精度

6.2.1 int128

```
1  //int128 是 ll 的两倍，需要自己写输入输出
2  #include <bits/stdc++.h>
3  using namespace std;
4  inline __int128 read(){
5      __int128 x=0,f=1;
6      char ch=getchar();
7      while(ch<'0' || ch>'9'){
8          if(ch=='-')
9              f=-1;
10         ch=getchar();
11     }
12     while(ch>='0' && ch<='9'){
13         x=x*10+ch-'0';
14         ch=getchar();
15     }
16     return x*f;
17 }
18 inline void print(__int128 x){
19     if(x<0){
20         putchar('-');
21         x=-x;
22     }
23     if(x>9)
24         print(x/10);
25     putchar(x%10+'0');
26 }
27 int main(){
28     __int128 a = read();
29     __int128 b = read();
30     print(a + b);
31     cout<<endl;
32     return 0;
33 }
```

6.2.2 简单高精度

```
1 static const int LEN = 1004;
2 int a[LEN], b[LEN], c[LEN], d[LEN];
3
4 void clear(int a[]) {
5     for (int i = 0; i < LEN; ++i) a[i] = 0;
6 }
7
8 void read(int a[]) {
9     static char s[LEN + 1];
10    scanf("%s", s);
11    clear(a);
12    int len = strlen(s);
13    for (int i = 0; i < len; ++i) a[len - i - 1] = s[i] - '0';
14 }
15
16 void print(int a[]) {
17     int i;
18     for (i = LEN - 1; i >= 1; --i)
19         if (a[i] != 0) break;
20     for (; i >= 0; --i) putchar(a[i] + '0');
21     putchar('\n');
22 }
23
24 void add(int a[], int b[], int c[]) {
25     clear(c);
26     for (int i = 0; i < LEN - 1; ++i) {
27         c[i] += a[i] + b[i];
28         if (c[i] >= 10) {
29             c[i + 1] += 1;
30             c[i] -= 10;
31         }
32     }
33 }
34
35 void sub(int a[], int b[], int c[]) {
36     clear(c);
37     for (int i = 0; i < LEN - 1; ++i) {
38         c[i] += a[i] - b[i];
39         if (c[i] < 0) {
40             c[i + 1] -= 1;
41             c[i] += 10;
42         }
43     }
```

```
43     }
44 }
45
46 void mul(int a[], int b[], int c[]) {
47     clear(c);
48     for (int i = 0; i < LEN - 1; ++i) {
49         for (int j = 0; j <= i; ++j) c[i] += a[j] * b[i - j];
50         if (c[i] >= 10) {
51             c[i + 1] += c[i] / 10;
52             c[i] %= 10;
53         }
54     }
55 }
56
57 inline bool greater_eq(int a[], int b[], int last_dg, int len)
58 ↪ {
59     if (a[last_dg + len] != 0) return true;
60     for (int i = len - 1; i >= 0; --i) {
61         if (a[last_dg + i] > b[i]) return true;
62         if (a[last_dg + i] < b[i]) return false;
63     }
64     return true;
65 }
66
67 void div(int a[], int b[], int c[], int d[]) {
68     clear(c); clear(d);
69     int la, lb;
70     for (la = LEN - 1; la > 0; --la)
71         if (a[la - 1] != 0) break;
72     for (lb = LEN - 1; lb > 0; --lb)
73         if (b[lb - 1] != 0) break;
74     if (lb == 0) {
75         puts("> <");
76         return;
77     }
78     for (int i = 0; i < la; ++i) d[i] = a[i];
79     for (int i = la - lb; i >= 0; --i) {
80         while (greater_eq(d, b, i, lb)) {
81             for (int j = 0; j < lb; ++j) {
82                 d[i + j] -= b[j];
83                 if (d[i + j] < 0) {
84                     d[i + j + 1] -= 1;
85                     d[i + j] += 10;
86                 }
87             }
88         }
89     }
90 }
```



```
86         }
87         c[i] += 1;
88     }
89 }
90 }
91
92 int main() {
93     read(a);
94     char op[4], scanf("%s", op);
95     read(b);
96     switch (op[0]) {
97         case '+':
98             add(a, b, c); print(c);
99             break;
100        case '-':
101            sub(a, b, c); print(c);
102            break;
103        case '*':
104            mul(a, b, c); print(c);
105            break;
106        case '/':
107            div(a, b, c, d);
108            print(c); print(d);
109            break;
110    }
111    return 0;
112 }
```

6.2.3 压位高精度

```
1 //高精度，支持乘法和加法但只支持正数
2 struct BigInt{
3     const static int mod = 10000;
4     const static int DLEN = 4;
5     //根据题目要求可对 a 数组大小进行修改
6     int a[6000], len;
7     BigInt(){
8         memset(a, 0, sizeof(a));
9         len=1;
10    }
11    BigInt(int v){
12        memset(a, 0, sizeof(a));
13        len=0;
14        do{
```

```

15         a[len++] = v % mod;
16         v /= mod;
17     } while (v);
18 }
19 BigInt(const char s[]) {
20     memset(a, 0, sizeof(a));
21     int L = strlen(s);
22     len = L / DLEN;
23     if (L % DLEN) len++;
24     int index = 0;
25     for (int i = L - 1; i >= 0; i -= DLEN) {
26         int t = 0;
27         int k = i - DLEN + 1;
28         if (k < 0) k = 0;
29         for (int j = k; j <= i; ++j)
30             t = t * 10 + s[j] - '0';
31         a[index++] = t;
32     }
33 }
34 BigInt operator +(const BigInt &b) const {
35     BigInt res;
36     res.len = max(len, b.len);
37     for (int i = 0; i <= res.len; ++i)
38         res.a[i] = 0;
39     for (int i = 0; i < res.len; ++i) {
40         res.a[i] += ((i < len) ? a[i] : 0) + ((i < b.len) ? b.a[i] : 0);
41         res.a[i + 1] += res.a[i] / mod;
42         res.a[i] %= mod;
43     }
44     if (res.a[res.len] > 0) res.len++;
45     return res;
46 }
47 BigInt operator *(const BigInt &b) const {
48     BigInt res;
49     for (int i = 0; i < len; ++i) {
50         int up = 0;
51         for (int j = 0; j < b.len; ++j) {
52             int temp = a[i] * b.a[j] + res.a[i + j] + up;
53             res.a[i + j] = temp % mod;
54             up = temp / mod;
55         }
56         if (up != 0)
57             res.a[i + b.len] = up;
58     }

```

```

59         res.len=len+b.len;
60         while(res.a[res.len-1]==0 && res.len>1) res.len--;
61         return res;
62     }
63     void output(){
64         printf("%d",a[len-1]);
65         for(int i=len-2;i>=0;--i)
66             printf("%04d",a[i]);
67         printf("\n");
68     }
69 };
70 int main()
71 {
72     //字符串读入
73     char a[2005],b[2005];
74     cin>>a>>b;
75     BigInt A,B;
76     A=BigInt(a),B=BigInt(b);
77     //可以直接用 cout 输出 char 数组内容
78     cout<<a<<" "<<b<<endl;
79     (A+B).output();//加法
80     (A*B).output();//乘法
81     return 0;
82 }

```

6.3 离散化

```

1  /*
2  对于包含重复元素，并且相同元素离散化后也要相同
3  例：
4  a:7 4 7 3 4 10
5  b:3 2 3 1 2 4
6  m=4
7  */
8  const int maxn=1e5+10;
9  int a[maxn], t[maxn], b[maxn];
10 int main(){
11     int n;scanf("%d",&n);
12     for(int i=1; i<=n; i++)
13         scanf("%d",&a[i]),t[i]=a[i];
14     sort(t+1,t+n+1);
15     int m=unique(t+1,t+n+1)-t-1;//求出的 m 为不重复的元素的
    ↪ 个数
16     for(int i=1; i<=n; i++)

```

```

17         b[i]=lower_bound(t+1,t+1+m,a[i])-t;
18         return 0;
19     }
20
21     /*
22     较上一种复杂度低
23     对于:
24     1. 包含重复元素, 并且相同元素离散化后不相同
25     2. 不包含重复元素, 并且不同元素离散化后不同
26     例:
27     a:7 4 7 3 4 10
28     b:4 2 5 1 3 6
29     */
30     struct A
31     {
32         int x, idx;
33         bool operator < (const A &rhs) const
34         {
35             return x < rhs.x;
36         }
37     };
38     A a[N];
39     int b[N];
40     int main()
41     {
42         int n;
43         scanf("%d",&n);
44         for(int i = 1; i <= n; ++i){
45             scanf("%d", &a[i].x);
46             a[i].idx = i;
47         }
48         sort(a + 1, a + n + 1);
49         for(int i = 1; i <= n; ++i)
50             b[a[i].idx] = i;
51         return 0;
52     }

```

6.4 常用公式

- $\sum_{i=1}^n i = \frac{n*(n+1)}{2}$
- $\sum_{i=1}^n i^2 = \frac{n*(n+1)*(2n+1)}{6}$
- $\sum_{i=1}^n i^3 = \left[\frac{n*(n+1)}{2} \right]^2$

7 经典例题

7.1 某天是星期几

```
/*
求某天是星期几
*/
char *name[] = { "monday", "tuesday", "wednesday",
"thursday", "friday", "saturday", "sunday" };
int main(void)
{
    int d, m, y, a;
    printf("Day: "); scanf("%d",&d);
    printf("Month: "); scanf("%d",&m);
    printf("Year: "); scanf("%d",&y);
    // 1 月 2 月当作前一年的 13,14 月
    if (m == 1 || m == 2) { m += 12; y--; }
    // 判断是否在 1752 年 9 月 3 日之前
    if ((y < 1752) || (y == 1752 && m < 9) || (y == 1752 && m
↵ == 9 && d < 3))
        a = (d + 2*m + 3*(m+1)/5 + y + y/4 +5) % 7;
    else
        a = (d + 2*m + 3*(m+1)/5 + y + y/4 - y/100 + y/400)%7;
    printf("it's a %s\n", name[a]);
    return 0;
}
```

8 注意事项

- 注意范围!!! 爆 int 多少回了? 有时候 ans 定义为 ll 也是不够的, 注意改为 ll 后 scanf 和 printf, 不行就 signed main, #define int ll
- 注意初始点的设置, 初始点是否有效是否得到正确的更新
- 注意边界条件如 $<$ 和 \leq , 注意特判如 $n = 1$