ACM Template

Rien

August 11, 2021

rien_zhu@163.com

Contents

1 字符串

1.1 KMP

```
/*
Next[] 的含义:x[i-Next[i]...i-1]=x[0...Next[i]-1]
Next[i] 为满足:x[i-z...i-1]=x[0...z-1] 的最大值
*/
void kmp_pre(char x[],int m,int next[]){
    int i,j;
    j=next[0]=-1;
    i=0;
    while(i<m){
        while(-1!=j && x[i]!=x[j])
            j=next[j];
        next[++i]=++j;
    }
}
const int N=1000005;
int Next[N];
/*
  返回 x 在 y 中出现的次数, 可以重叠
  y 是主串,x 是模式串
*/
int KMP_Count(char x[],int m,char y[], int n){
    int i,j;
    int ans=0;
    kmp_pre(x,m,Next);
    i=j=0;
    while(i<n){
        while(-1!=j && y[i]!=x[j])
            j=Next[j];
        i++;j++;
        if(j>=m){
            ans++;
            j=Next[j];
        }
    }
    return ans;
}
```

2 数据结构

2.1 并查集

2.1.1 优化并查集

```
/*
路径压缩并查集,
*/
const int N=1e5+5;
int fa[N];
void init(){
    for(int i=1;i<=N;++i)
        fa[i]=i;
}
int find(int x){
    int temp=x;
    while(temp!=fa[temp])
        temp=fa[temp];
    while(x!=fa[x]){
        x=fa[x];
        fa[x]=temp;
    }
    return temp;
}
void union(int x,int y){
    fa[find(x)]=find(y);
}
/*
按秩合并并查集, 不破坏树形结构
连通块数量为 block, 大小为 size
*/
int f[N],size[N],block;
void Init(){
    for(int i=1;i<=N;++i)
        f[i]=0;
    block=n;
}
int Find(int x){
    if(!f[x]) f[x]=x,size[x]=1;
    if(f[x]==x) return x;
    return f[x]=Find(f[x]);
}
void Union(int x,int y){
    x=Find(x);
```

```
    y=Find(y);
    if(x==y) return ;
    if(size[x]>size[y]) swap(x,y);
    f[x]=y;
    size[y]+=size[x];
    block--;
}
```

2.2 01 字典树

```
/*
01trie 树
在一组数中找跟某个数异或结果最大的数
*/
const int MAXN=100005;
//MAXN 右移需根据题目判断，如数据范围为 int 则 *32 即右移 5 位
int val[MAXN<<5];       //val[i]=j 表示编号为 i 的节点的值为 j
int cnt;                //节点数量
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
void init(){
    cnt=1;
    tree[0][0]=tree[0][1]=0;
}
void insert(int x){
    int v,u=0;
    for(int i=31;i>=0;--i){
        v=(x>>i)&1;
        if(!tree[u][v]){
            tree[cnt][0]=tree[cnt][1]=0;//初始化新节点的子节点
            val[cnt]=0;                 //节点值为 0，表示到此不是一个数
            tree[u][v]=cnt++;           //指向该节点
        }
        u=tree[u][v];//到下一节点
    }
    val[u]=x;
}
int query(int x){
    int v,u=0;
    for(int i=31;i>=0;i--){
        v=(x>>i)&1;
        if(tree[u][v^1])//利用贪心策略，优先寻找和当前位不同的数
            u=tree[u][v^1];
```

```
        else
            u=tree[u][v];
    }
    return val[u];
}
int  main(){
    int n,m;scanf("%d%d",&n,&m);
    init();
    int tmp;
    for(int i=0;i<n;++i){
        scanf("%d",&tmp);
        insert(tmp);
    }
    for(int i=0;i<m;++i){
        scanf("%d",&tmp);
        printf("%d\n",query(tmp));
    }
    return 0;
}
```

2.3 树状数组

2.3.1 一维树状数组

```
/*
BIT 一维树状数组
*/
const int N=1000005;
ll tree[N],a[N],b[N];
int n,m,op,l,r,v;
//单点修改区间查询
void add(int p,int x){
    for(int i=p;i<=n;i+=i&-i)
        tree[i]+=x;
}
ll query(int p){
    ll ret=0;
    for(int i=p;i;i-=i&-i)
        ret+=tree[i];
    return ret;
}
int main(){
    //读入数据
    for(int i=1;i<=n;++i){
```

```
        cin>>v;
        add(i,v);
    }
    add(l,v);//将第 l 个数加上 v
    cout<<query(r)-query(l-1)<<endl;//询问 [l-r] 区间的和
    return 0;
}

//区间修改单点查询
void add(int p,int x){
    for(int i=p;i<=n;i+=i&-i)
        tree[i]+=x;
}
ll query(int p){
    ll ret=0;
    for(int i=p;i;i-=i&-i)
        ret+=tree[i];
    return ret;
}
int main(){
    //读入数据，维护差分数组
    for(int i=1;i<=n;++i){
        cin>>a[i];
        add(i,a[i]-a[i-1]);
    }
    add(l,v),add(r+1,-v);//将 [l-r] 加上 v
    cout<<query(l)<<endl//询问第 l 个数的值
    return 0;
}

//区间修改区间查询
void add(int p,int x){
    for(int i=p;i<=n;i+=i&-i){
        b[i]+=(ll)x;
        tree[i]+=(ll)p*x;
    }
}
ll query(int p){
    ll ret=0;
    for(int i=p;i;i-=i&-i)
        ret+=1ll*(p+1)*b[i]-tree[i];
    return ret;
}
int main(){
```

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

2.3.2 求逆序对

```
//树状数组求逆序对
void modify(int x, int d){
        for(int i = x; i <= n; i += i&-i)c[i] += d;
}
int getsum(int x){
        int sum = 0;
        for(int i = x; i >= 1; i -= i&-i)sum += c[i];
        return sum;
}
int main(){
        scanf("%d", &n);
        for(int i = 1; i <= n; i ++){
                scanf("%d", &x);
                ans += getsum(n) - getsum(x);
                modify(x, 1);
        }
}
```

3 数学

3.1 快速幂

3.1.1 数字快速幂

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

3.2 组合数学

```
/*
组合数学求法：
1. 杨辉三角打表
所需空间:O(nm)，时间：预处理 O(nm)，查询 O(1)
*/
const int mod = 1e9+7;
int c[maxn][maxn];
void init(int n){
        c[0][0]=1;
        rep(i,1,n){
                c[i][0]=1;
                rep(j,1,n+1)c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
        }
}

/*
2. 直接用公式
Cnm=n!/(m!(n-m)!)
不能取模，时间：预处理 O(1)，查询 O(n)
*/
ll C(ll n, ll m)
```

```
{
    if (n < m) return 0;
    ll ret = 1;
    rep(i, n - m + 1, n)
        ret *= i;
    rep(i, 2, m)
        ret /= i;
    return ret;
}

/*
3. 乘法逆元
可以取模, 适用于 P 比较大的题目
所需空间 O(n), 时间: 预处理 O(n), 查询 O(1)
*/
//N 的范围需要根据题目确定
int fac[N+5],inv[N+5];
ll qpow(ll bsc,ll y){
        ll ret = 1;
        while(y){
                if(y&1) ret = ret*bsc%mod;
                bsc = bsc*bsc%mod;
                y >>= 1;
        }
        return ret;
}
void init(){
        fac[0] = 1;
        for(int i=1;i<=N;i++)
                fac[i] = (ll)fac[i-1]*i%mod;
        inv[N] = qpow(fac[N],mod-2);
        for(int i=N-1;i>=0;i--)
                inv[i] = (ll)inv[i+1]*(i+1)%mod;
}
int C(int n,int m){
        if(m>n) return 0;
        return (ll)fac[n]*inv[m]%mod*inv[n-m]%mod;
}

/*
4.Lucas 定理
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
lucas(n,m,p) 返回 C(n,m) 对 p 取模的结果
```

```
*/
ll f[N],rf[N];
ll mul(ll x,ll y,ll p){
    return (x*y-(ll)(x/(long double)p*y+1e-3)*p+p)%p;
}
ll pow(ll a,ll b,ll p){
    ll t=1;
    for(;b;b>>=1,a=mul(a,a,p))
        if(b & 1) t=mul(t,a,p );
    return t;
}
//使用时为 init(p)
void init(int p){
    f[0]=1;
    for(int i=1;i<p;++i) f[i]=f[i-1]*i%p;
    rf[p-1]=pow(f[p-1],p-2,p);
    for(int i=p-1;i;i--) rf[i-1]=rf[i]*i%p;
}
ll C(int n,int m,int mod){
    if(m>n||m<0||n<0) return 0;
    return f[n]*rf[m]%mod*rf[n-m]%mod;
}
ll lucas(ll n,ll m,ll p){
    if(n<m) return 0;
    if(!m || n==m)  return 1;
    return C(n%p,m%p,p) *lucas(n/p,m/p,p )%p;
}
```

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：

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

用整除分块可以在 $O(\sqrt{n})$ 求出，代码如下：

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

3.3.2 莫比乌斯函数

```
const int MAXN=1000000;
bool check[MAXN+10];
int prime[MAXN+10];
int mu[MAXN+10];
void Moblus(){
    memset(check,false,sizeof(check));
    mu[1]=1;
    int tot=0;
    for(int i=2;i<=MAXN;i++){
        if(!check[i]){
            prime[tot++]=i;
            mu[i]=-1;
        }
        for(int j=0;j<tot;j++){
            if(i*prime[j]>MAXN) break;
            check[i*prime[j]]=true;
            if(i%prime[j]==0){
                mu[i*prime[j]]=0;
                break;
            }
            else mu[i*prime[j]]=-mu[i];
        }
    }
}
```

3.4 BSGS

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

```
int BSGS(int a,int b,int n){
    memset(head,-1,sizeof(head));
    top=1;
    if(b==1) return 0;
    int m=sqrt(n*1.0),j;
    long long x=1,p=1;
    for(int i=0;i<m;++i,p=p*a%n) insert(p*b%n,i);
    for(long long i=m;;i+=m){
        if((j=find(x=x*p%n))!=-1) return i-j;
        if(i>n) break;
    }
    return -1;
}
```

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 $n_1, n_2, \cdots, n_k$ 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:

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

```
  }
  return (ans % mod + mod) % mod;
}
```

4 图论

4.1 BFS

4.1.1 伪代码

```
/*
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。
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。
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。
*/
bfs(s) {
  q = new queue()
  q.push(s), visited[s] = true
  while (!q.empty()) {
    u = q.pop()
    for each edge(u, v) {
      if (!visited[v]) {
        q.push(v)
        visited[v] = true
      }
    }
  }
}
```

4.1.2 BFS 记录距离和路径

```
/*
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。
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。
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。
restore(x) 输出的是从起点到 x 这个点所经过的点。

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。
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。
直到某一时刻，队列 Q 为空，这时 BFS 结束。

时间复杂度 O(n + m)
空间复杂度 O(n) (vis 数组和队列)
*/
```

```
void bfs(int u) {
  while (!Q.empty()) Q.pop();
  Q.push(u);
  vis[u] = 1;
  d[u] = 0;
  p[u] = -1;
  while (!Q.empty()) {
    u = Q.front();
    Q.pop();
    for (int i = head[u]; i; i = e[i].x) {
      if (!vis[e[i].t]) {
        Q.push(e[i].t);
        vis[e[i].t] = 1;
        d[e[i].t] = d[u] + 1;
        p[e[i].t] = u;
      }
    }
  }
}
void restore(int x) {
  vector<int> res;
  for (int v = x; v != -1; v = p[v]) {
    res.push_back(v);
  }
  std::reverse(res.begin(), res.end());
  for (int i = 0; i < res.size(); ++i) printf("%d", res[i]);
  puts("");
}
```

4.2 最短路

4.2.1 Dijkstra 算法单源最短路

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

```
priority_queue<pair<int,int> > q;
void add(int x,int y,int z){
    ver[++tot]=y,edge[tot]=z,Next[tot]=head[x],head[x]=tot;
}
void dijkstra(){
    memset(d,0x3f,sizeof(d));//dist 数组
    memset(v,0,sizeof(v));//节点标记
    //起始点为 1, 若更改要改两处
    d[1]=0;
    q.push(make_pair(0,1));
    while(q.size()){
        int x=q.top().second;q.pop();
        if(v[x]) continue;
        v[x]=1;
        for(int i=head[x];i;i=Next[i]){
            int y=ver[i],z=edge[i];
            if(d[y]>d[x]+z){
                d[y]=d[x]+z;
                q.push(make_pair(-d[y],y));
            }
        }
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        //如果是无向图还要 add(y,x,z)
        add(x,y,z);
    }
    dijkstra();
    //求单源最短路
    for(int i=1;i<=n;++i)
        printf("%d\n",d[i]);
    return 0;
}
```

4.2.2 bellman_ford 算法单源最短路

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

```
可以判断是否存在负环回路.
返回 true: 当且仅当图中不包含从源点可达的负权回路
vector<Edge> E; 先 E.clear() 初始化, 然后加入所有边
点的编号从 1 开始
*/
const int INF=0x3f3f3f3f;
const int N=2550;
int dist[N];
struct Edge
{
    int u,v;
    int cost;
    Edge(int _u=0,int _v=0,int _cost=0):u(_u),v(_v),cost(_cost){}
};
vector<Edge>E;
//点的编号从 1 开始
void addedge(int u,int v,int w){
    E.push_back(Edge(u,v,w));
}
bool bellman_ford(int start,int n){
    for(int i=1;i<=n;++i)   dist[i]=INF;
    dist[start]=0;
    for(int i=1;i<n;++i){
        bool flag=false;
        for(int j=0;j<E.size();j++){
            int u=E[j].u;
            int v=E[j].v;
            int cost=E[j].cost;
            if(dist[v]>dist[u]+cost){
                dist[v]=dist[u]+cost;
                flag=true;
            }
        }
        if(!flag) return true;//没有负环回路
    }
    for(int j=0;j<E.size();j++)
        if(dist[E[j].v]>dist[E[j].u]+E[j].cost)
            return false;//有负环回路
    return true;//没有负环回路
}
int main()
{
```

```
    int n,m;cin>>n>>m;
    E.clear();
    for(int i=1;i<=m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        addedge(x,y,z);
        addedge(y,x,z);
    }
    bellman_ford(s,n);
    for(int i=1;i<=n;++i)
        printf("%d\n",dist[i]);
    return 0;
}
```

4.2.3 Floyd 任意两点最短路径

```
/*
Floyd 算法 计算任意两点间的距离 复杂度为 O(N^3)
*/
#define inf 0x3f3f3f3f
int dist[1000][1000];
int main()
{
    int k,i,j,n,m;///n 表示顶点个数, m 表示边的条数
    scanf("%d %d",&n,&m);
    for(i=1; i<=n; i++){//初始化
        for(j=1; j<=n; j++){
            if(i==j) dist[i][j]=0;
            else dist[i][j]=inf;
        }
    }
    int a,b,c;
    for(i=1; i<=m; i++){//有向图
        scanf("%d %d %d",&a,&b,&c);
        dist[a][b]=min(dist[a][b],c);
    }
    for(k=1; k<=n; k++){     //Floyd-Warshall 算法核心语句
        for(i=1; i<=n; i++){
            for(j=1; j<=n; j++){
                ↪ dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
    for(i=1; i<=n; i++){//输出最终的结果, 最终二维数组中存的即
    ↪ 使两点之间的最短距离
        for(j=1; j<=n; j++)
```

```
                printf("%10d",dist[i][j]);
            printf("\n");
        }
        return 0;
}
```

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

```
/*
Prim 求 MST
耗费矩阵 cost[][], 标号从 0 开始,0~n-1 注意注意!!!
返回最小生成树的权值，返回-1 则表示原图不连通
*/
const int INF=0x3f3f3f3f;
const int MAXN=110;
bool vis[MAXN];
int lowc[MAXN];
//点从 0 到 n-1
int Prim(int cost[][MAXN],int n){
    int ans=0;
    memset(vis,0,sizeof vis);
    vis[0]=true;
    for(int i=1;i<n;++i)    lowc[i]=cost[0][i];
    for(int i=1;i<n;++i){
        int minc=INF;
        int p=-1;
        for(int j=0;j<n;++j){
            if(!vis[j]&&minc>lowc[j]){
                minc=lowc[j];
                p=j;
            }
        }
        if(minc==INF) return -1;//原图不连通
        ans+=minc;
        vis[p]=true;
        for(int j=0;j<n;++j){
            if(!vis[j] && lowc[j]>cost[p][j])
                lowc[j]=cost[p][j];
        }
```

```
    }
    return ans;
}
int main()
{
    int cost[MAXN][MAXN];
    //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
    memset(cost,0x3f,sizeof(cost));
    while (k--){
        int u,v,w;
        cin>>u>>v>>w;
        //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
        u--;
        v--;
        cost[u][v]=w;
        cost[v][u]=w;
    }
    cout<<Prim(cost,n)<<endl;
    return 0;
}
```

4.3.2 Kruskal 算法

```
/*
Kruskal 算法求 MST
*/
//根据题目调试最大点数和边数
const int MAXN=1100;//最大点数
const int MAXM=200005;//最大边数
int F[MAXN];//并查集
struct Edge {
    int u,v,w;
}edge[MAXM];//储存边的信息: 起点, 终点, 权值
int tol=0;//边数, 记得赋值为 0
void addedge(int u,int v,int w){//加边
    edge[tol].u=u;
    edge[tol].v=v;
    edge[tol++].w=w;
}
bool cmp(Edge a,Edge b){//排序
    return a.w<b.w;
}
```

```
int find(int x){
    return F[x]==x?x:F[x]=find(F[x]);
}
//传入点数，返回最小生成树权值，如果不连通返回-1
int Kruskal(int n){
    //根据点的编号（从 0 或从 1 开始）初始并查集
    for(int i=1;i<=n;++i) F[i]=i;
    sort(edge,edge+tol,cmp);
    int cnt=0;//计算加入边数
    int ans=0;
    for(int i=0;i<tol;++i){
        int u=edge[i].u;
        int v=edge[i].v;
        int w=edge[i].w;
        int t1=find(u);
        int t2=find(v);
        if(t1!=t2){
            ans+=w;
            F[t1]=t2;
            cnt++;
        }
        if(cnt==n-1) break;
    }
    if(cnt<n-1) return -1;//不连通
    else return ans;
}
int main()
{
    int n,m;cin>>n>>m;//点数和边数
    for(int i=0;i<m;++i){
        int u,v,w;cin>>u>>v>>w;
        addedge(u,v,w);//只用读一次就行
    }
    cout<<Kruskal(n);//返回最小生成树权值
    return 0;
}
```

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])

```
/*
悬线法求最大全 1 子矩阵
*/
const int MAXN=2005;
char x[MAXN][MAXN];
int y[MAXN][MAXN],l[MAXN][MAXN],r[MAXN][MAXN],up[MAXN][MAXN];
int main(){
    int n,m; cin>>n>>m;
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            cin>>x[i][j];
            //将输入处理成 01 矩阵
            if(x[i][j]=='F'){
                //此处赋值需注意，只有 1 处才赋值，否则全 0 矩
                ↪ 阵也会输出面积为 1
                y[i][j]=1;
                l[i][j]=j;
                r[i][j]=j;
                up[i][j]=1;
            }
            else
                y[i][j]=0;
        }
    }
    //按行处理
    for(int i=1;i<=n;++i){
        for(int j=2;j<=m;++j){
            if(y[i][j-1] && y[i][j])
                l[i][j]=l[i][j-1];
        }
```

```
        for(int j=m-1;j>=1;--j){
            if(y[i][j+1] && y[i][j])
                r[i][j]=r[i][j+1];
        }
    }
    int len,ans=0;
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            //i>1 很关键，用于处理只有 1 行的情况
            if(i>1 && y[i-1][j] && y[i][j]){
                up[i][j]=up[i-1][j]+1;
                l[i][j]=max(l[i][j],l[i-1][j]);
                r[i][j]=min(r[i][j],r[i-1][j]);
            }
            len=r[i][j]-l[i][j]+1;
            ans=max(ans,len*up[i][j]);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

5.2 LIS 最长上升子序列

```
/*
动态规划求最长上升子序列，复杂度为 O(nlogn)
*/
int DP(int n){
    int i,len,pos;
    b[1]=a[1];
    len=1;
    for(i=2;i<=n;++i){
        //a[i] 大则可直接插到后面
        if(a[i]>=b[len]){
            len=len+1;
            b[len]=a[i];
        }
        //二分法在 b 数组中找出第一个比 a[i] 大的位置并让 a[i]
        ↪ 代替
        else {
            pos=lower_bound(b+1,b+1+len,a[i])-b;
            b[pos]=a[i];
        }
    }
```

```
    return len;
}
```

5.3 背包

5.3.1 基础背包

```
int nValue,nKind;
const int N=100005;
int dp[N],cost[N],value[N],num[N];
//0-1 背包, 代价为 cost, 价值为 value
void ZeroOnePack(int cost,int value){
    for(int i=nValue;i>=cost;i--)
        dp[i]=max(dp[i],dp[i-cost]+value);
}
//完全背包, 代价为 cost, 价值为 value
void CompletePack(int cost,int value){
    for(int i=cost;i<=nValue;++i)
        dp[i]=max(dp[i],dp[i-cost]+value);
}
//多重背包, 代价为 cost, 价值为 value, 数量为 amount
void MultiplePack(int cost,int value,int amount){
    if(cost*amount>=nValue)
        CompletePack(cost,value);
    else {
        int k=1;
        while(k<amount){
            ZeroOnePack(k*cost,k*value);
            amount-=k;
            k<<=1;
        }
        ZeroOnePack(amount*cost,amount*value);
    }
}
int main(){
    //记得清空数组
    memset(dp,0,sizeof dp);
    //0-1 背包
    rep(i,1,nKind)
        ZeroOnePack(cost[i],value[i]);
    printf("%d\n",dp[nValue]);
    //完全背包
    rep(i,1,nKind)
        CompletePack(cost[i],value[i]);
```

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

6 杂项

6.1 输入输出

6.1.1 快读

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

6.1.2 关闭同步

```
//关闭同步
ios::sync_with_stdio(0);
cin.tie(0);
```

6.1.3 快读快写

```
#include <bits/stdc++.h>
using namespace std;
//快读
inline int read() {
    int num=0, w=0;
    char ch=0;
    while (!isdigit(ch)) {
        w|=ch=='-';
        ch = getchar();
    }
    while (isdigit(ch)) {
        num = (num<<3) + (num<<1) + (ch^48);
        ch = getchar();
    }
    return w? -num: num;
}
//快写
inline void write(int x)
{
```

```
    if(x<0) {
        putchar('-');
        x = -x;
    }
    if(x>9) write(x / 10);
    putchar(x % 10 + '0');
}


int main(){
    int a;
    a = read();     //读入到 t 中
    write(t);       //输出 t
    putchar('\n');
}
```

6.1.4 8.11 所用板子

```
#include<bits/stdc++.h>
using namespace std;
#define ll  long long
#define ull unsigned long long
#define ld long double
#define rep(i, s, t) for (int i = (int)(s); i <= (int)(t); ++i)
#define debug(x) cerr<<#x<<" = "<<(x)<<endl
template <typename T> T &read(T &r) {
    r = 0; bool w = 0; char ch = getchar();
    while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch = getchar();
    while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch ^ 48), ch = getchar();
    return r = w ? -r : r;
}
ll mod=1e9+7;
ll INF=1e15;
ll Inf=0x3f3f3f3f;
double pi=acos(-1.0);


int  main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
```

```

    return 0;
}
```

6.2 高精度

6.2.1 int128

```
//int128 是 ll 的两倍，需要自己写输入输出
#include <bits/stdc++.h>
using namespace std;
inline __int128 read(){
    __int128 x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=x*10+ch-'0';
        ch=getchar();
    }
    return x*f;
}
inline void print(__int128 x){
    if(x<0){
        putchar('-');
        x=-x;
    }
    if(x>9)
        print(x/10);
    putchar(x%10+'0');
}
int main(){
    __int128 a = read();
    __int128 b = read();
    print(a + b);
    cout<<endl;
    return 0;
}
```

6.2.2 简单高精度

```
static const int LEN = 1004;
int a[LEN], b[LEN], c[LEN], d[LEN];

void clear(int a[]) {
  for (int i = 0; i < LEN; ++i) a[i] = 0;
}

void read(int a[]) {
  static char s[LEN + 1];
  scanf("%s", s);
  clear(a);
  int len = strlen(s);
  for (int i = 0; i < len; ++i) a[len - i - 1] = s[i] - '0';
}

void print(int a[]) {
  int i;
  for (i = LEN - 1; i >= 1; --i)
    if (a[i] != 0) break;
  for (; i >= 0; --i) putchar(a[i] + '0');
  putchar('\n');
}

void add(int a[], int b[], int c[]) {
  clear(c);
  for (int i = 0; i < LEN - 1; ++i) {
    c[i] += a[i] + b[i];
    if (c[i] >= 10) {
      c[i + 1] += 1;
      c[i] -= 10;
    }
  }
}

void sub(int a[], int b[], int c[]) {
    clear(c);
    for (int i = 0; i < LEN - 1; ++i) {
        c[i] += a[i] - b[i];
        if (c[i] < 0) {
            c[i + 1] -= 1;
            c[i] += 10;
        }
```

```
    }
}

void mul(int a[], int b[], int c[]) {
  clear(c);
    for (int i = 0; i < LEN - 1; ++i) {
    for (int j = 0; j <= i; ++j) c[i] += a[j] * b[i - j];
    if (c[i] >= 10) {
        c[i + 1] += c[i] / 10;
        c[i] %= 10;
    }
    }
}

inline bool greater_eq(int a[], int b[], int last_dg, int len) {
    if (a[last_dg + len] != 0) return true;
    for (int i = len - 1; i >= 0; --i) {
        if (a[last_dg + i] > b[i]) return true;
        if (a[last_dg + i] < b[i]) return false;
    }
    return true;
}

void div(int a[], int b[], int c[], int d[]) {
    clear(c);clear(d);
    int la, lb;
    for (la = LEN - 1; la > 0; --la)
        if (a[la - 1] != 0) break;
    for (lb = LEN - 1; lb > 0; --lb)
        if (b[lb - 1] != 0) break;
    if (lb == 0) {
        puts("> <");
        return;
    }
    for (int i = 0; i < la; ++i) d[i] = a[i];
    for (int i = la - lb; i >= 0; --i) {
        while (greater_eq(d, b, i, lb)) {
            for (int j = 0; j < lb; ++j) {
                d[i + j] -= b[j];
                if (d[i + j] < 0) {
                    d[i + j + 1] -= 1;
                    d[i + j] += 10;
                }
```

```
            }
            c[i] += 1;
        }
    }
}

int main() {
  read(a);
  char op[4],scanf("%s", op);
  read(b);
  switch (op[0]) {
    case '+':
      add(a, b, c);print(c);
      break;
    case '-':
      sub(a, b, c);print(c);
      break;
    case '*':
      mul(a, b, c);print(c);
      break;
    case '/':
      div(a, b, c, d);
      print(c);print(d);
      break;
  }
  return 0;
}
```

6.2.3 压位高精度

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

```
            a[len++]=v%mod;
            v/=mod;
        }while(v);
    }
    BigInt(const char s[]){
        memset(a,0,sizeof(a));
        int L=strlen(s);
        len=L/DLEN;
        if(L%DLEN)  len++;
        int index = 0;
        for(int i=L-1;i>=0;i-=DLEN){
            int t=0;
            int k=i-DLEN+1;
            if(k<0) k=0;
            for(int j=k;j<=i;++j)
                t=t*10+s[j]-'0';
            a[index++]=t;
        }
    }
    BigInt operator +(const BigInt &b)const {
        BigInt res;
        res.len=max(len,b.len);
        for(int i=0;i<=res.len;++i)
            res.a[i]=0;
        for(int i=0;i<res.len;++i){
            res.a[i]+=((i<len)?a[i]:0)+((i<b.len)?b.a[i]:0);
            res.a[i+1]+=res.a[i]/mod;
            res.a[i]%=mod;
        }
        if(res.a[res.len]>0) res.len++;
        return res;
    }
    BigInt operator *(const BigInt &b)const {
        BigInt res;
        for(int i=0;i<len;++i){
            int up= 0;
            for(int j=0;j<b.len;++j){
                int temp=a[i]*b.a[j]+res.a[i+j]+up;
                res.a[i+j]=temp%mod;
                up=temp/mod;
            }
            if(up!=0)
                res.a[i+b.len]=up;
        }
```

```
        res.len=len+b.len;
        while(res.a[res.len-1]==0 && res.len>1) res.len--;
        return res;
    }
    void output(){
        printf("%d",a[len-1]);
        for(int i=len-2;i>=0;--i)
            printf("%04d",a[i]);
        printf("\n");
    }
};
int  main()
{
    //字符串读入
    char a[2005],b[2005];
    cin>>a>>b;
    BigInt A,B;
    A=BigInt(a),B=BigInt(b);
    //可以直接用 cout 输出 char 数组内容
    cout<<a<<" "<<b<<endl;
    (A+B).output();//加法
    (A*B).output();//乘法
    return 0;
}
```

6.3 离散化

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

```
                b[i]=lower_bound(t+1,t+1+m,a[i])-t;
        return 0;
}

/*
较上一种复杂度低
对于:
1. 包含重复元素，并且相同元素离散化后不相同
2. 不包含重复元素，并且不同元素离散化后不同
例:
a:7 4 7 3 4 10
b:4 2 5 1 3 6
*/
struct A
{
    int x, idx;
    bool operator < (const A &rhs) const
    {
        return x < rhs.x;
    }
};
A a[N];
int b[N];
int main()
{
        int n;
        scanf("%d",&n);
        for(int i = 1; i <= n; ++i){
                scanf("%d", &a[i].x);
                a[i].idx = i;
        }
        sort(a + 1, a + n + 1);
        for(int i = 1; i <= n; ++i)
            b[a[i].idx] = i;
        return 0;
}
```

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 $<=$, 注意特判如 $n = 1$