ACM-ICPC 代码模板

Andeviking(422563809@qq.com)

2023 年 3 月 23 日

The real voyage of algorithm consists,
not in seeking new landscapes,
but in having new eyes.

Contents

1 头文件

1.1 头文件（全）

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
#define iofast ios::sync_with_stdio(false),cin.tie(0)
    ,cout.tie(0)
#define lowbit(x) (x&(-x))
#define inv(x) qpow(x,mod-2)
#define ctz(x) __builtin_ctz(x) //末尾0个数
#define clz(x) __builtin_clz(x) //前导0个数
#define popcount(x) __builtin_popcount(x) //1的个数
#define ffs(x) __builtin_ffs(x) //最后一个1的位置
#define int128 __int128_t
const int iINF = 0x3f3f3f3f;
const ll llINF = 0x3f3f3f3f3f3f3f3f;
template<typename T>
void read(T& x)
{
    x = 0;
    int flag = 1;
    char c = getchar();
    while(!isdigit(c)){
        if (c == '-')
            flag = -1;
        c = getchar();
    }
    while(isdigit(c)){
        x = (x << 3) + (x << 1) + (c ^ 48);
        c = getchar();
    }
    x *= flag;
}

template<typename T,typename ...Arg>
void read(T& x,Arg& ...args)
{
    read(x);
    read(args...);
}
const ll mod = 998244353;
ll qpow(ll a,ll b)
{
    ll ans = 1;
    a %= mod;
    for (; b;b>>=1){
        if(b&1)
            ans = ans * a % mod;
        a = a * a % mod;
    }
    return ans % mod;
}

template<typename T>
void write(T x, char c = '\0') {
    if (x < 0) {
        x = -x;
        putchar('-');
    }
    if (x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
    if (c != '\0')
        putchar(c);
}
/*--------------------------------------*/

void solve()
{

}

/*--------------------------------------*/

int main()
{
    iofast;
    int t = 1;
    cin >> t;
    while (t--)
        solve();

    return 0;
}
```

1.2 头文件（赛）

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define int ll
#define endl '\n'
const int mod=998244353;


void solve()
{

}

signed main ()
{
    ios::sync_with_stdio(0);cin.tie(0);
    int t=1;

    while(t--){
        solve();
    }
    return 0;
}
```

2 图论

请注意，图论算法后续省略建图过程，默认链式前向星存图

2.1 建图

```
const int N = 100005;
const int M = 200005;

int head[N], ver[M], Next[M], edge[M];
int tot;
void add(int x,int y,int z)
{
```

```
    ver[++tot] = y;
    Next[tot] = head[x];
    edge[tot] = z;
    head[x] = tot;
}
```

2.2 SPFA 算法

```
/*寻找负环时添加cnt数组,并将队列替换为栈*/
/*注意队列操作与栈操作的替换*/

/*计算差分约束时
如果求的是最小值,则应该求最长路,如果求的是最大值,则应该求最短路
负环即无解
把每个x[i] ≤ x[j] + C[k]不等式转化为一条从x[j]走到x[i]长度为C[k]的边
从0号点向x[i]<=C[k]的i点连边
*/
//int cnt[N];
//stack<int>st;
int d[N];
bool v[N];
queue<int>q;

void spfa(int s)
{
    memset(d,0x3f,sizeof d);
    memset(v,0,sizeof v);
    //memset(cnt,0,sizeof cnt); 负环cnt数组初始化
    d[s]=0;
    v[s]=1;
    q.push(s);
    while(!q.empty()){
        int x=q.front();
        q.pop();
        v[x]=0;
        for(int i=head[x];i;i=Next[i]){
            int y=ver[i];
            int z=edge[i];
            if(d[y]>d[x]+z){
                d[y]=d[x]+z;

                /*负环操作
                cnt[y]=cnt[x]+1;
                if(cnt[y]>=n+1)
                    return true;
                    */

                if(!v[y]){
                    q.push(y);
                    v[y]=1;
                }
            }
        }

    }

    return;
}
```

2.3 Dijestra 算法

```
int d[N];
bool v[N];
typedef pair<int,int> pii;
priority_queue<pii,vector<pii>,greater<pii>>q;

void dij(int s)
{
    //初始化
    memset(d,0x3f,sizeof d);
    memset(v,0,sizeof v);
    while(!q.empty())
        q.pop();

    q.push({0,s});
    d[s]=0;
    while(!q.empty()){
        auto [dist,x]=q.top();
        q.pop();
        if(v[x])
            continue;
        v[x]=1;

        for(int i=head[x];i;i=Next[i]){
            int y=ver[i];
            int z=edge[i];
            if(d[y]>dist+z){
                d[y]=dist+z;
                q.push({d[y],y});
            }
        }

    }

    return;
}
```

2.4 Floyd 算法

```
int d[305][305];
for(int i=1;i<=m;i++){
    int x,y,z;
    cin>>x>>y>>z;
    d[x][y]=min(d[x][y],z);

    /*传递闭包
    d[x][y]=d[y][x]=1;
    */
}

for(int k=1;k<=n;k++)
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++){
            d[i][j]=min(d[i][j],d[i][k]+d[k][j]);

            /*传递闭包
            d[i][j]|=d[i][k]&d[k][j];
            */
        }
```

2.5 Kruscal 最小生成树

```cpp
/*-----并查集代码省略-----*/
typedef pair<int,pair<int,int>> e;
priority_queue<e,vector<e>,greater<e>>q;
int kruscal()
{
    int ans=0;
    while(!q.empty()){
        int x=q.top().second.first;
        int y=q.top().second.second;
        int z=q.top().first;
        q.pop();

        //get()与merge()均为并查集操作
        if(get(x)==get(y))
            continue;

        ans+=z;
        merge(x,y);

    }

    return ans;
}
int main()
{

    /*----初始化并查集省略----*/

    for(int i=1;i<=k;i++){
        int x,y,z;
        cin>>x>>y>>z;
        add(x,y,z);
        add(y,x,z);
        q.push({z,{x,y}});
    }

    int ans=Kruscal();
    return 0;
}
```

2.6 Kruscal 重构树

与 Kruscal 算法类似，其中每次操作建立虚点将合并的两个点集结合。
虚点的权值即为边权
两点间的 lca 所代表的点权即为两点路径中需要经过的最大边权最小值

2.7 树哈希

```cpp
//hah中保存以该点为子树的哈希值
//map不要清空
int id;
int hah[N];
map<vector<int>, int>mp;
void dfs(int x, int fa)
{
    vector<int>temp;
    for (int i = head[x];i;i = Next[i]) {
        int y = ver[i];
        if (y == fa)
            continue;
            dfs(y, x);
            temp.push_back(hah[y]);
    }

    sort(range(temp));
    if (!mp[temp])
        mp[temp] = ++id;
    hah[x] = mp[temp];
}
```

2.8 AUH 树同构算法

```cpp
//摘抄自OI_WIKI,判断两棵树是否同构的确定性算法
// Tree Isomorphism, O(nlogn)
// replace quick sort with radix sort ==> O(n)
// Author: _Backl1ght
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 5;
const int maxn = N << 1;

int n;

struct Edge {
    int v, nxt;
} e[maxn << 1];

int head[maxn], sz[maxn], f[maxn], maxv[maxn], tag[
    maxn], tot, Max;
vector<int> center[2], L[maxn], subtree_tags[maxn];

void addedge(int u, int v)
{ // 建图
    e[tot].v = v;
    e[tot].nxt = head[u];
    head[u] = tot++;
    e[tot].v = u;
    e[tot].nxt = head[v];
    head[v] = tot++;
}

void dfs_size(int u, int fa)
{ // 找到 size 值
    sz[u] = 1;
    maxv[u] = 0;
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa)
            continue;
        dfs_size(v, u);
        sz[u] += sz[v];
        maxv[u] = max(maxv[u], sz[v]);
    }
}

void dfs_center(int rt, int u, int fa, int id)
{
    maxv[u] = max(maxv[u], sz[rt] - sz[u]);
    if (Max > maxv[u]) {
        center[id].clear();
        Max = maxv[u];
    }
    if (Max == maxv[u])
```

```
        center[id].push_back(u); // 如果相等就push_back
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa)
            continue;
        dfs_center(rt, v, u, id);
    }
}

int dfs_height(int u, int fa, int depth)
{ // 递归查找 height
    L[depth].push_back(u);
    f[u] = fa;
    int h = 0;
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa)
            continue;
        h = max(h, dfs_height(v, u, depth + 1));
    }
    return h + 1;
}

void init(int n)
{ // 一开始的处理
    for (int i = 1; i <= 2 * n; i++)
        head[i] = 0;
    tot = 1;
    center[0].clear();
    center[1].clear();

    int u, v;
    for (int i = 1; i <= n - 1; i++) {
        scanf("%d %d", &u, &v);
        addedge(u, v);
    }
    dfs_size(1, -1);
    Max = n;
    dfs_center(1, 1, -1, 0);

    for (int i = 1; i <= n - 1; i++) {
        scanf("%d %d", &u, &v);
        addedge(u + n, v + n);
    }
    dfs_size(1 + n, -1);
    Max = n;
    dfs_center(1 + n, 1 + n, -1, 1);
}

bool cmp(int u, int v) { return subtree_tags[u] < subtree_tags[v]; }

bool rootedTreeIsomorphism(int rt1, int rt2)
{
    for (int i = 0; i <= 2 * n + 1; i++)
        L[i].clear(), subtree_tags[i].clear();
    int h1 = dfs_height(rt1, -1, 0);
    int h2 = dfs_height(rt2, -1, 0);

    if (h1 != h2)
        return false;
    int h = h1 - 1;

    for (int j = 0; j < (int)L[h].size(); j++)
        tag[L[h][j]] = 0;
    for (int i = h - 1; i >= 0; i--) {
        for (int j = 0; j < (int)L[i + 1].size(); j++) {
            int v = L[i + 1][j];
            subtree_tags[f[v]].push_back(tag[v]);
        }

        sort(L[i].begin(), L[i].end(), cmp);

        for (int j = 0, cnt = 0; j < (int)L[i].size(); j++) {
            if (j && subtree_tags[L[i][j]] != subtree_tags[L[i][j - 1]])
                ++cnt;
            tag[L[i][j]] = cnt;
        }
    }
    return subtree_tags[rt1] == subtree_tags[rt2];
}

bool treeIsomorphism()
{
    if (center[0].size() == center[1].size()) {
        if (rootedTreeIsomorphism(center[0][0], center[1][0]))
            return true;
        if (center[0].size() > 1)
            return rootedTreeIsomorphism(center[0][0], center[1][1]);
    }
    return false;
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        init(n);
        puts(treeIsomorphism() ? "YES" : "NO");
    }
    return 0;
}
```

2.9 虚树

```
//建立的虚树中只含有询问点以及他们的LCA
//在解决询问点的总数不大且只需要用到LCA和被询问点时使用
const int N = 100005;
const int M = 2 * N;

int head[N], ver[M], Next[M];
int tot;
int low[N], dfn[N];
void add(int x,int y)
{
    ver[++tot] = y;
    Next[tot] = head[x];
    head[x] = tot;
}

/*----求LCA过程省略,请自行补充----*/

```

```cpp
vector<int> v;
stack<int> st;
int s;
bool cmp(int a,int b)
{
    return dfn[a] < dfn[b];
}
void build()
{
    sort(v.begin(), v.end(),cmp);
    int sz = v.size();
    for (int i = sz - 2; ~i;i--)
        v.emplace_back(lca(v[i], v[i + 1]));

    sort(v.begin(), v.end(),cmp);
    v.erase(unique(v.begin(), v.end()), v.end());

    s = v[0];
    while(!st.empty())
        st.pop();
    sz = v.size();
    for (int i = 0; i < sz;i++){
        int u = v[i];
        while(!st.empty()&&low[st.top()]<dfn[u])
            st.pop();

        if(!st.empty()){
            add(u, st.top());
            add(st.top(), u);
        }
        st.push(u);
    }

    return;
}

//标记是否是被询问的节点
map<int, int> mp;
void clr()
{
    mp.clear();
    for(auto c:v)
        head[c] = 0;
    tot = 0;
    v.clear();
}

//求dfs序以及low数组
//low数组中存储点x的子树内的 最大 dfs序
int tim;
void dfs(int x,int fa)
{
    dfn[x] = ++tim;
    low[x] = dfn[x];
    for (int i = head[x]; i;i=Next[i]){
        int y = ver[i];
        if(y==fa)
            continue;

        dfs(y, x);
        low[x] = max(low[x], low[y]);
    }

    return;
}

int dp(int x,int fa)
{
    //树形dp部分
}
void solve()
{
    int n;
    cin >> n;

    for (int i = 1; i < n;i++){
        int u, v;
        cin >> u >> v;

        add(u, v);
        add(v, u);
    }

    dfs(1, 0);
    bfs(1);
    memset(head, 0, sizeof head);
    tot = 0;

    int q;
    cin >> q;
    while(q--){
        clr();
        int k;
        cin >> k;

        //一次询问k个点
        for (int i = 1; i <= k;i++){
            int x;
            cin >> x;
            v.emplace_back(x);
            mp[x] = 1;
        }

        build();

        //树形dp
        cout << dp(s, 0) << '\n';
    }
}
```

2.10 倍增求树上 lca

```cpp
/*----注意修改循环中k的大小---*/
const int N=500005;
queue<int>q;
int d[N];
int f[N][20];

//x指根节点编号
void bfs(int x)
{
    d[x]=1;
    q.push(x);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];i;i=Next[i]){
            int y=ver[i];
            if(d[y])
```

```
            continue;
         d[y]=d[u]+1;
         q.push(y);
         f[y][0]=u;
         for(int k=1;k<=15;k++){
            f[y][k]=f[f[y][k-1]][k-1];
         }
      }
   }
   return;
}

int lca(int x,int y)
{
   if(d[x]<d[y])
      swap(x,y);

   for(int k=15;k>=0;k--)
      if(d[f[x][k]]>=d[y])
         x=f[x][k];

   if(x==y)
      return x;

   for(int k=15;k>=0;k--)
      if(f[x][k]!=f[y][k])
         x=f[x][k],y=f[y][k];

   return f[x][0];


}
```

2.11 树链剖分

```
const int N=100005;
int sz[N],son[N],dep[N],fa[N];

void dfs1(int x)
{
   sz[x]=1;
   for(int i=head[x];i;i=Next[i]){
      int y=ver[i];
      if(y==fa[x])
         continue;
      dep[y]=dep[x]+1;
      fa[y]=x;
      dfs1(y);

      sz[x]+=sz[y];
      if(sz[y]>sz[son[x]])
         son[x]=y;
   }

   return;
}

int dfn[N],top[N],a[N],w[N];
int tim;
void dfs2(int x,int t)
{
   dfn[x]=++tim;
   top[x]=t;
   a[tim]=w[x];
   if(son[x])
      dfs2(son[x],t);

   for(int i=head[x];i;i=Next[i]){
      int y=ver[i];

      if(y==fa[x]||y==son[x])
         continue;

      dfs2(y,y);
   }

   return;
}

//询问与修改大致相同,仅需将区间操作改为区间询问即可
void change_path(int u,int v,int x)
{

   while(top[u]!=top[v]){
      if(dep[top[u]]<dep[top[v]])
         swap(u,v);

      //区间操作[dfn[top[u]],dfn[u]];
      change(1,dfn[top[u]],dfn[u],x);
      u=fa[top[u]];
   }

   if(dep[u]>dep[v])
      swap(u,v);

   //区间操作[dfn[u],dfn[v]];
   change(1,dfn[u],dfn[v],x);
   return;
}

int main()
{
   //树链剖分操作
   dep[1]=1;
   dfs1(1);
   dfs2(1,1);
}
```

2.12 有向图强连通分量

```
const int N=100005;
int dfn[N],low[N];
int tim;
stack<int>st;
int sz[N],cnt; //强连通分量的数目以及大小
bool v[N];
int id[N]; //每个点所属的强连通分量编号

void tarjan(int x)
{
   dfn[x]=low[x]=++tim;
   st.push(x),v[x]=1;
   for(int i=head[x];i;i=Next[i]){
      int y=ver[i];

      if(!dfn[y]){
         tarjan(y);
         low[x]=min(low[y],low[x]);
```

```
        }
        else if(v[y])
            low[x]=min(low[x],dfn[y]);
    }

    if(low[x]==dfn[x]){
        ++cnt;
        int y;
        do{
            y=st.top();
            st.pop();

            v[y]=0;
            id[y]=cnt;
            sz[cnt]++;
        }while(y!=x);
    }
    return;
}

int main()
{
    //使用方法
    for(int i=1;i<=n;i++)
        if(!dfn[i])
            tarjan(i);

}
```

2.13 2-SAT

连边方式:
$a \cup b \Rightarrow$!a→b 与!b→a
$a \cap b \Rightarrow$!a→a 与!b→b
a if $b \Rightarrow$ b→a 与!a→!b

若 a 与!a 位于同一个强连通分量, 则无解
否则一定有解, 且 a 的值取 0 和 1 中拓扑序靠后的值
(tarjan 算法求出的强连通分量编号为拓扑序逆序)

2.14 无向图双连通分量

2.15 拓扑排序

```
//使用前注意预处理出个点的入度
//答案存放在a数组中,共有cnt个点
//若 cnt<n 则说明图中有环
const int N=100005;
const int M=200005;
int n;
int a[N],cnt;
queue<int>q;
int deg[N]; //存储入度
void topsort()
{
    for(int i=1;i<=n;i++)
        if(deg[i]==0)
            q.push(i);

    while(!q.empty()){
        int x=q.front();
        q.pop();
        a[++cnt]=x;
        for(int i=head[x];i;i=Next[i]){
            int y=ver[i];
            deg[y]--;
            if(!deg[y])
                q.push(y);
        }
    }

    return;
}
```

2.16 匈牙利算法

```
//时间复杂度 O(NM)
//正确性:当一个节点成为匹配点后,至多因为找到增广路而更换匹
    配对象,并不会变为非匹配点
const int N=10005;
bool v[N];
int match[N];
bool dfs(int x)
{
    for(int i=head[x];i;i=Next[i]){
        int y=ver[i];
        if(v[y])
            continue;

        v[y]=1;
        if(!match[y]||dfs(match[y])){
            match[y]=x;
            return true;
        }

    }

    return false;
}

int main()
{
    int ans=0;
    for(int i=1;i<=n;i++){
        memset(v,0,sizeof v); //注意清空
        if(dfs(i))
            ans++;
    }
}
```

2.17 二分图最大权匹配 (KM 算法)

2.18 Dinic 最大流

```
//最大流==最小割
//无向边的反边不为零,边权与正边相同
const int N=100;
const int M=25010;
const int INF=0x7fffffff;
int s,t;

```

```
int head[N],cur[N],d[N],ver[M],edge[M],Next[M];
int tot=-1;

void add(int x,int y,int z)
{
    ver[++tot]=y;
    Next[tot]=head[x];
    edge[tot]=z;
    head[x]=tot;

    ver[++tot]=x;
    Next[tot]=head[y];
    edge[tot]=0;
    head[y]=tot;
    return;
}


queue<int>q;
bool bfs()
{
    memset(d,-1,sizeof d);
    while(!q.empty())
        q.pop();

    d[s]=0;
    q.push(s);
    cur[s]=head[s];

    while(!q.empty()){
        int x=q.front();
        q.pop();

        for(int i=head[x];~i;i=Next[i]){
            int y=ver[i];
            int z=edge[i];

            if(d[y]!=-1||!z)
                continue;

            d[y]=d[x]+1;
            cur[y]=head[y];
            if(y==t)
                return true;

            q.push(y);
        }
    }

    return false;
}

int dfs(int u,int limit)
{
    if(u==t)
        return limit;
    int flow=0;
    for(int i=cur[u];~i&&flow<limit;i=Next[i]){
        int y=ver[i];
        int z=edge[i];

        if(d[y]!=d[u]+1||!z)
            continue;

        int use=dfs(y,min(z,limit-flow));
        if(!use)
            d[y]=0;
        edge[i]-=use;
        edge[i^1]+=use;
        flow+=use;

    }

    return flow;
}
int dinic()
{
    int ans=0,flow;
    while(bfs())
        while(flow=dfs(s,INF))
            ans+=flow;

    return ans;
}
int main()
{
    //注意初始化head为-1
    memset(head,-1,sizeof head);
    t=n+1; //点数 +1

    /*---加边---*/

    int ans=dinic();
}
```

2.19 最小费用最大流

```
//注意与最大流的区别
const int N=5005;
const int M=100005;
const int INF=0x3f3f3f3f;

int head[N],ver[M],Next[M],edge[M];
int w[M];
int tot=-1;
void add(int x,int y,int z,int d)
{
    ver[++tot]=y;
    edge[tot]=z;
    w[tot]=d;
    Next[tot]=head[x];
    head[x]=tot;
    ver[++tot]=x;
    edge[tot]=0;
    w[tot]=-d;
    Next[tot]=head[y];
    head[y]=tot;
}

int s,t;
int incf[N];
int d[N],pre[N];
bool v[N];
queue<int>q;
bool spfa()
{
    memset(d,0x3f,sizeof d);
    memset(incf,0,sizeof incf);
    while(!q.empty())
```

```
        v[q.front()]=0,q.pop();

    d[s]=0;
    incf[s]=INF;
    q.push(s);
    v[s]=1;
    while(!q.empty()){
        int x=q.front();
        q.pop();
        v[x]=0;
        for(int i=head[x];~i;i=Next[i]){
            int y=ver[i];
            int z=edge[i];
            int d1=w[i];

            if(z&&d[y]>d[x]+d1){
                d[y]=d[x]+d1;
                pre[y]=i;
                incf[y]=min(z,incf[x]);
                if(!v[y]){
                    q.push(y);
                    v[y]=1;
                }
            }
        }
    }

    return incf[t]>0;
}

void SFPA(int& flow,int& cost)
{
    flow=cost=0;
    while(spfa()){
        int now=incf[t];
        flow+=now;
        cost+=now*d[t];
        for(int i=t;i!=s;i=ver[pre[i]^1]){
            edge[pre[i]]-=now;
            edge[pre[i]^1]+=now;
        }
    }
    return;
}
int main()
{
    memset(head,-1,sizeof head);
    int n,m;
    cin>>n>>m>>s>>t;

    for(int i=1;i<=m;i++){
        int u,v,c,w;
        cin>>u>>v>>c>>w;
        add(u,v,c,w);
    }
    int flow,cost;
    SFPA(flow,cost);
    cout<<flow<<' '<<cost<<'\n';
    return 0;
}
```

2.20 原始对偶费用流

```
//费用流卡常专用，亲测好用
const int N = 5005;
const int M = 100005;

int head[N], ver[M], Next[M];
ll edge[M], w[M];
int tot = -1;
void add(int x, int y, int z, int d)
{
    ver[++tot] = y;
    edge[tot] = z;
    w[tot] = d;
    Next[tot] = head[x];
    head[x] = tot;
    ver[++tot] = x;
    edge[tot] = 0;
    w[tot] = -d;
    Next[tot] = head[y];
    head[y] = tot;
}
int s, t;
ll delta;
ll dist[N];
void Reduce()
{
    for (int i = 0;i <= tot;++i) {
        int x = ver[i ^ 1], y = ver[i];
        w[i] += dist[y] - dist[x];
    }

    delta += dist[s];
}

bool vis[N];
bool BellmanFord()
{
    queue<int>q;
    memset(dist, 0x3f, sizeof dist);
    dist[t] = 0;
    q.push(t);
    vis[t] = 1;
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for (int i = head[x];~i;i = Next[i]) {
            int y = ver[i];
            ll z = w[i ^ 1];
            if (edge[i ^ 1] && dist[y] > dist[x] + z) {
                dist[y] = dist[x] + z;
                if (!vis[y]) {
                    vis[y] = 1;
                    q.push(y);
                }
            }
        }
    }
    return dist[s] != llINF;
}

priority_queue<pair<ll, ll>, vector<pair<ll, ll>>,
    greater<pair<ll, ll>>>q;
bool Dijkstra()
{
    memset(dist, 0x3f, sizeof dist);
    memset(vis, 0, sizeof vis);
```

```
    dist[t] = 0;
    q.push({ dist[t],t });

    while (!q.empty()) {
        auto [dis, x] = q.top();
        q.pop();
        if (vis[x])
            continue;
        vis[x] = 1;

        for (int i = head[x];~i;i = Next[i]) {
            int y = ver[i];
            ll z = w[i ^ 1];
            if (edge[i ^ 1] && dist[y] > dist[x] + z) {
                dist[y] = dist[x] + z;
                q.push({ dist[y],y });
            }
        }
    }
    return dist[s] != llINF;
}

ll dfs(int x, ll flow)
{
    if (x == t || !flow)
        return flow;
    vis[x] = 1;
    ll res = flow;
    for (int i = head[x];~i;i = Next[i]) {
        int y = ver[i];
        ll z = w[i];
        if (!vis[y] && edge[i] && !z) {
            ll temp = dfs(y, min(res, edge[i]));
            edge[i] -= temp;
            edge[i ^ 1] += temp;
            res -= temp;
        }
    }
    return flow - res;
}

void Augment(ll& flow,ll& cost)
{
    ll cur = 0;
    while (memset(vis, 0, sizeof vis),cur = dfs(s,
        llINF)) {
        flow += cur;
        cost += delta * cur;
    }
    return;
}

void PrimalDual(ll& flow,ll& cost)
{
    flow = 0, cost = 0;
    if (!BellmanFord())
        return;
    Reduce();
    Augment(flow, cost);
    while (Dijkstra()) {
        Reduce();
        Augment(flow, cost);
    }
}

void init()
{
    memset(head, -1, sizeof head);
    tot = -1;
    delta = 0;
}
void solve()
{
    init();
    int n, m;
    cin >> n >> m >> s >> t;

    for (int i = 1;i <= m;i++) {
        int x, y;
        ll z, d;
        cin >> x >> y >> z >> d;
        add(x, y, z, d);
    }
    ll flow, cost;
    PrimalDual(flow, cost);
    cout << flow << ' ' << cost << '\n';
    return;
}
```

2.21 朱刘算法

```
```

3 数学

3.1 快速幂

```
//多项式快速幂与之类似,将a当作多项式做NTT即可
const ll mod = 998244353;
ll qpow(ll a,ll b)
{
    ll ans = 1;
    a %= mod;
    for (; b;b>>=1){
        if(b&1)
            ans = ans * a % mod;
        a = a * a % mod;
    }
    return ans % mod;
}
```

3.2 整除分块

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

//上取整
for(int l = 1, r; l <= n; l = r + 1)
{

    r = ((n+l-1)/l==1? n:(n-1)/((n+l-1)/l-1));
    /*----操作----*/
}
```

3.3 Eratosthenes 筛法

```
//筛出1-n之间的质数
const int N=100005;
bool v[N];
vector<int>prime;
void primes(int n)
{
    memset(v,0,sizeof v);
    for(int i=2;i<=n;i++){
        if(v[i])
            continue;
        prime.emplace_back(i);
        for(int j=1;j<=n/i;j++)
            v[i*j]=1;
    }
    return;
}
```

3.4 线性筛

```
const int N=500005;
int v[N],prime[N];
int cnt;
void primes(int n)
{
    memset(v,0,sizeof v);
    cnt=0;
    for(int i=2;i<=n;i++){
        if(!v[i]){
            v[i]=i;
            prime[++cnt]=i;
        }
        for(int j=1;j<=cnt;j++){
            if(prime[j]>v[i]||prime[j]>n/i)
                break;
            v[i*prime[j]]=prime[j];
        }
    }
    return;
}
```

3.5 质因数分解

```
//时间复杂度O(sqrt(n))
//p中存质因数,c中存幂次
int cnt;
int p[N],c[N];
void divide(int n)
{
    cnt=0;
    for(int i=2;i<=sqrt(n);i++)
        if(n%i==0){
            p[++cnt]=i;
            c[cnt]=0;
            while(n%i==0){
                n/=i;
                c[cnt]++;
            }
        }

    if(n>1)
        p[++cnt]=n,c[cnt]=1;

    return;
}
```

3.6 Pollard's Rho 质因数分解

```
//要重写qpow函数
#define int128 __int128_t
const int128 tag = 1;

int prime[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
    31, 37};
ll qpow(ll a,ll b,ll mod)
{
    ll ans = 1;
    a %= mod;
    for (; b;b>>=1){
        if(b&1)
            ans = (tag * ans * a) % mod;
        a = (tag * a * a) % mod;
    }
    return ans % mod;
}

//判断是否是素数
bool check(ll a,ll p)
{
    ll d = p - 1;
    ll get = qpow(a, d, p);
    if(get!=1)
        return 1;
    while((d&1)^1){
        d >>= 1;
        if((get=qpow(a,d,p))==p-1)
            return 0;
        else if(get!=1)
            return 1;
    }

    return 0;
}
bool Miller_Rabin(ll x)
{
    if(x>40){
        for (int i = 0; i < 12;i++)
            if(check(prime[i],x))
                return 0;
        return 1;
    }

    for (int i = 0; i < 12;i++)
        if(x==prime[i])
            return 1;
    return 0;
}

//结果存在factor中
//key为底数,val为指数
map<ll, int> factor;
ll pollard_rho(ll n,ll c)
{
    if(n==4)
        return 2;
```

```cpp
    ll x = rand() % (n - 1) + 1;
    ll y = x;
    x = (tag * x * x + c) % n;
    y = (tag * y * y + c) % n;
    y = (tag * y * y + c) % n;
    for (int lim = 1; x != y;lim=min(lim<<1,128)){
        ll cnt = 1;
        for (int i = 0; i < lim;i++){
            cnt = tag * cnt * abs(x - y) % n;
            if(!cnt)
                break;
            x = (tag * x * x + c) % n;
            y = (tag * y * y + c) % n;
            y = (tag * y * y + c) % n;
        }
        ll d = __gcd(cnt, n);
        if(d!=1)
            return d;
    }

    return n;
}

void findFac(ll n)
{
    if(Miller_Rabin(n)){
        factor[n]++;
        return;
    }
    ll p = n;
    while(p>=n)
        p = pollard_rho(p, rand() % (n - 1) + 1);
    findFac(p);
    findFac(n / p);
}
void solve()
{
    factor.clear();
    ll x;

    //使用前先判断是否是素数
    //注意1的问题
    if(Miller_Rabin(x)){
        cout << "Prime\n";
        return;
    }

    findFac(x);
}
```

3.7 1-N 正约数集合

```cpp
//时间复杂度O(nlogn)
const int N=500005;
vector<int>factor[N];
int main()
{
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n/i;j++)
            factor[i*j].emplace_back(i);
}
```

3.8 欧拉函数

```cpp
int phi(int n)
{
    int ans=n;
    for(int i=2;i<=sprt(n);i++)
        if(n%i==0){
            ans=(ans/i)*(i-1);
            while(n%i==0)
                n/=i;
        }

    if(n>1)
        ans=(ans/n)*(n-1);
    return ans;
}
```

3.9 2-N 欧拉函数

```cpp
//时间复杂度O(nlogn)
const int N=200005;
int phi[N];
void euler(int n)
{
    for(int i=2;i<=n;i++)
        phi[i]=i;
    for(int i=2;i<=n;i++)
        if(phi[i]==i)
            for(int j=i;j<=n;j+=i)
                phi[j]=(phi[j]/i)*(i-1);

    return;
}
```

3.10 扩展欧几里得算法

计算 $ax + by = \gcd(a, b)$ 的通解
方程 $ax + by = c$ 的通解可表示为:
$x = \frac{c}{d}x_0 + k\frac{b}{d}, y = \frac{c}{d}y_0 - k\frac{a}{d}$ $(k \in \mathbb{Z})$

```cpp
//返回a与b的gcd,并将通解通过引用传出
int exgcd(int a,int b,int& x,int& y)
{
    if(!b){
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y);
    int z=x;
    x=y;
    y=z-y*(a/b);

    /*若求最小正整数解x
    x=x%b;
    if(x<0)
        x+=b;
    */
    return d;
}
```

3.11 BSGS 算法

```
```

3.12 矩阵运算

```
//注意构造函数传参
struct matrix {
   vector<vector<ll>>m;
   int r, c;
   matrix(int _r, int _c)
   {
      r = _r;
      c = _c;
      m.resize(r);
      for (auto& cc : m)
         cc.resize(c);
   }
};

matrix operator*(const matrix& a, const matrix& b)
{
   matrix c(a.r, b.c);
   for (int i = 0;i < a.r;++i)
      for (int j = 0;j < b.c;++j)
         for (int k = 0;k < a.c;++k)
            (c.m[i][j] += a.m[i][k] * b.m[k][j] %
                mod) %= mod;
   return c;
}

matrix matrix_pow(matrix a,ll b)
{
   matrix ans(a.r, a.c);
   for (int i = 0;i < a.r;i++)
      ans.m[i][i] = 1;

   for (; b;b >>= 1) {
      if (b & 1)
         ans = ans * a;
      a = a * a;
   }
   return ans;
}
```

3.13 高斯消元

```
//O(n^3)
int c[N][N],b[N];
void Gauss(int n)
{
   for (int i = 1; i <= n;i++){
      for (int j = i; j <= n;j++){

         //找非零元
         if(c[j][i]){
            for (int k = 1; k <= n;k++)
               swap(c[i][k], c[j][k]);
            swap(b[i], b[j]);
            break;
         }
      }

      for (int j = 1; j <= n;j++){
         if(i==j)
            continue;

         auto rate = c[j][i] / c[i][i];
         for (int k = i; k <= n;k++)
            c[j][k] -= c[i][k] * rate;

         b[j] -= b[i] * rate;
      }
   }

   //b[i]中存储的即为答案
   for(int i = 1; i <= n;i++)
      b[i]/=c[i][i];
}
```

3.14 线性基

```
//每次使用记得清空
//区间线性基需要保存以位置 i 为右端点的前缀线性基，此外，
    还要记录每个向量的位置，更新时尽可能靠右
ll p[64];
bool flag;
void insert(ll x)
{
   ll use = 1ll << 62;
   for (int i = 62;i >= 0;i--) {
      if (use & x) {
         if (p[i])
            x ^= p[i];
         else {
            p[i] = x;
            return;
         }
      }
      use >>= 1;
   }
   flag = 1;
   return;
}

ll ask_max()
{
   ll ans = 0;
   for (int i = 62;i >= 0;i--)
      if ((ans ^ p[i]) > ans)
         ans ^= p[i];
   return ans;
}

ll ask_min()
{
   if (flag)
      return 0;
   for (int i = 0;i <= 62;i++)
      if (p[i])
         return p[i];
   return -1;
}

bool ask_check(ll x)
{
   if (!x)
```

```
        return flag;

    ll use = 1ll << 62;
    for (int i = 62;i >= 0;i--) {
        if (x & use) {
            if (!p[i])
                return false;
            else
                x ^= p[i];
        }
        use >>= 1;
    }
    return true;
}

ll ask_kth(ll k)
{
    if (flag)
        k--;
    if (!k)
        return 0;

    ll ans = 0;
    ll use = 1ll << 62;
    int cnt = 0;
    for (int i = 62;i >= 0;i--)
        cnt += bool(p[i]);

    if ((1ll << cnt) - 1 < k)
        return -1;

    ll now = 1ll << (cnt - 1);
    for (int i = 62;i >= 0;i--) {
        if (p[i]) {
            if (bool(ans & use) != bool(k & now))
                ans ^= p[i];
            now >>= 1;
        }
        use >>= 1;
    }

    return ans;
}

/*---------区间线性基*----------*/
pii tag[2 * N][32];
void insert(int p, int x)
{
    int pos = p;
    for (int i = 30;~i;--i) {
        int use = (1 << i);
        if (!(x & use)) {
            tag[p][i] = tag[p - 1][i];
            continue;
        }

        if (!tag[p - 1][i].first) {
            tag[p][i].first = x;
            tag[p][i].second = pos;
            x = 0;
        }
        else if (pos > tag[p - 1][i].second) {
            tag[p][i].first = x;
            tag[p][i].second = pos;
            pos = tag[p - 1][i].second;
        }
        else
            tag[p][i] = tag[p - 1][i];
        x = x ^ tag[p - 1][i].first;
    }
}
ll ask(int l, int r)
{
    ll ans = 0;
    for (int i = 30;~i;--i) {
        if (tag[r][i].second < l)
            continue;
        if ((ans ^ tag[r][i].first) > ans)
            ans ^= tag[r][i].first;
    }
    return ans;
}
```

3.15 Lucas 定理

$O(p * \log_p n)$

```
//C代表组合数,求取过程省略
ll lucas(ll n,ll m)//递归lucas函数
{
    if(m==0)
        return 1ll;
    return lucas(n/mod,m/mod)*C(n%mod,m%mod)%mod;
}
```

3.16 莫比乌斯函数

```
const int N = 50005;
bool vis[N];
int prime[N], mo[N];
void get_mo()
{
    int cnt = 0;
    vis[1] = 1;
    mo[1] = 1;
    for (int i = 2;i <= 50000;i++) {
        if (!vis[i]) {
            prime[++cnt] = i;
            mo[i] = -1;
        }
        for (int j = 1;j <= cnt && 1ll * i * prime[j]
             <= 50000;j++) {
            vis[i * prime[j]] = 1;
            mo[i * prime[j]] = (i % prime[j] ? -mo[i] :
                0);
            if (i % prime[j] == 0)
                break;
        }
    }

}
```

3.17 莫比乌斯反演

第一种形式:
如果有 $F(n)=\sum_{d|n} f(d)$, 则 $f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$
第二种形式:

如果有 $F(n)=\sum_{n|d} f(d)$, 则 $f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$
套路与难点: 构造适当的 $f(n)$ 与 $F(n)$, 从而套用整除分块等技巧

3.18 0/1 分数规划

0/1 分数规划模型是指, 给定整数 $a_1,a_2,\cdots,a_n$ 以及 $b_1,b_2,\cdots,b_n$ 求一组解 $x_i(1\leqslant i\leqslant n, x_i=0$ 或 $1)$, 使下式最大化:

$$\frac{\sum_{i=1}^{n} a_i * x_i}{\sum_{i=1}^{n} b_i * x_i}$$

即从给定的 n 对整数 a_i,b_i 中选出若干对, 使得选出的数对的 a 之和与 b 之和的商最大

二分答案, 当二分的值为 mid 时, 我们就计算 $\sum_{i=1}^{n}(a_i - mid*b_i)*x_i$ 的最大值若非负, 则令 l=mid, 否则令 r=mid
最大值显然为 $a_i - mid*b_i$ 中所有正数的和

3.19 容斥原理

集合 S 不具有性质 $P_1,P_2,\cdots,P_m$ 的物体个数:

$$\left|\overline{A_1}\cap\overline{A_2}\cap\cdots\cap\overline{A_m}\right| = |S| - \sum_{i=1}^{m}|A_i| + \sum_{i,j:i<j}|A_i\cap A_j| - \sum_{i,j,k:i<j<k}|A_i\cap A_j\cap A_k| + \cdots + (-1)^m\,|A_1\cap A_2\cap\cdots\cap A_m|$$

推论: 至少具有性质 $P_1,P_2,\cdots,P_m$ 之一的物体个数:

$$|A_1\cup A_2\cup\cdots\cup A_m| = \sum_{i=1}^{m}|A_i| - \sum_{i,j:i<j}|A_i\cap A_j| + \sum_{i,j,k:i<j<k}|A_i\cap A_j\cap A_k| + \cdots + (-1)^{m+1}\,|A_1\cap A_2\cap\cdots\cap A_m|$$

其中 A_i 表示 S 中具有性质 P_i 的集合,$\overline{A_i}$ 表示 A_i 在 S 中的补集

3.20 多项式反演

3.21 Min25 筛

对于 Min25 筛, 其要求求和的函数满足以下性质:
1. 积性函数, 即对于任意的 $gcd(x,y)=1$, 都有 $f(x)f(y)=f(xy)$
2. $f(p)$,$p\in\mathbb{P}$ 能被表示为项数比较小的多项式
3. $f(p^c)$,$p\in\mathbb{P}$ 能够快速求值

Min25 筛基本步骤:
1. 将待求函数在质数情况下的表达式转化成 $\sum i^k$ 的形式
2. 线性筛出 $1\sim\sqrt{n}$ 之间的质数, 并求出其 k 次方前缀和
3. 对于形如 $\lfloor\frac{n}{x}\rfloor$ 的数, 算出 $g(\lfloor\frac{n}{x}\rfloor,0)$, 注意去掉 1^k
4. 套递归式计算 $g(n,j)$, 这里 g 可以用滚动数组
5. 递归计算 $S(n,x)$, 无需记忆化, 答案即为 $S(n,0)+1$

该模板以 $\sum_{i=1}^{n} f(i)$, 其中 $f(p^k)=p^k$ 为例

```
//当题目有高次项时,仅需在标记位置做出添加或修改即可
//sp数组为质数高次前缀和
//具体方式为添加新的g和sp数组,当作新的项处理,原式结果即为项的线性组合
//g数组中存储的为 1^m + 2^m +... 的前tn项和,在该模板中给出的是m=1的情况
//其他部位模仿模板添加即可
const int N = 1000005;
const ll inv2 = 500000004;
const ll mod = 1000000007;
int cnt;
int v[N], prime[N];
void primes(int n)
{
    for (int i = 2; i <= n;i++){
        if(!v[i]){
            v[i] = i;
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt;j++){
            if(prime[j]>v[i]||prime[j]>n/i)
                break;
            v[i * prime[j]] = prime[j];
        }
    }
    return;
}

//题目的f(x)函数,其中x代指 p^k
ll f(ll x)
{
    return x % mod;
}

ll sp[N], g[N];
ll w[N];
int idx1[N], idx2[N];
ll n;
int sqrn;
ll S(ll x,ll y)
{
    if(prime[y]>=x)
        return 0;
    int k;
    if(x<=sqrn)
        k = idx1[x];
    else
        k = idx2[n / x];
    ll ans = g[k] - sp[y] + mod; //记得修改
    ans %= mod;

    for (int i = y + 1; i <= cnt;i++){
        if(1ll*prime[i]*prime[i]>x)
            break;

        ll pe = prime[i];
        for (int e = 1; pe <= x;e++,pe*=prime[i]){
            ll nx = pe % mod;
            ans = (ans + f(nx) * (S(x / pe, i) + (e != 1)) % mod) % mod;
        }
    }

    return ans;
}

void solve()
{
    read(n);
    sqrn = sqrt(n);

    primes(sqrn);

```

```
    //注意可能的添加
    for (int i = 1; i <= cnt;i++)
        sp[i] = (sp[i - 1] + prime[i]) % mod;

    int tot = 0;
    for (ll l = 1, r, tn; l <= n;l=r+1){
        r = n / (n / l);
        w[++tot] = n / l; //注意不要取模
        tn = w[tot] % mod;

        //注意可能的添加
        g[tot] = tn * (tn + 1) % mod * inv2 % mod - 1;
        if(w[tot]<=sqrn)
            idx1[w[tot]] = tot;
        else
            idx2[n / w[tot]] = tot;
    }

    for (int i = 1; i <= cnt;i++)
        for (int j = 1; j <= tot;j++){
            if (1ll * prime[i] * prime[i] > w[j])
                break;

            int k;
            if(w[j]/prime[i]<=sqrn)
                k = idx1[w[j] / prime[i]];
            else
                k = idx2[n / (w[j] / prime[i])];

            //注意可能的添加
            (g[j] -= 1ll * prime[i] * (g[k] - sp[i - 1]
                 + mod) % mod) %= mod;
            (g[j] += mod) %= mod;
        }

    ll ans = S(n, 0) + 1; \\计算答案
    ans %= mod;
}
```

3.22 杜教筛

3.23 Powerful Number 筛

3.24 快速数论变换 (NTT)

```
const int N=300005;
int n,m; //项数分别为n和m
int rev[N],tot,bit;
ll a[N],b[N]; //多项式 a ,b 大小为最高幂次的二倍
const ll g=3;
const ll mod = 998244353;
void ntt(ll a[],int flag)
{
    for(int i=0;i<tot;i++)
        if(i<rev[i])
            swap(a[i],a[rev[i]]);
    for(int mid=1;mid<tot;mid<<=1){
        ll len=mid*2;
        ll g1=qpow(g,(mod-1)/len)%mod;
        if(flag==-1)
            g1=qpow(g1,mod-2);
        for(int i=0;i<tot;i+=len){
            ll gk=1;
            for(int j=0;j<mid;j++,gk=gk*g1%mod){
                int x=a[i+j],y=a[i+j+mid]*gk%mod;
                a[i+j]=(x+y)%mod;
                a[i+j+mid]=(x-y+mod)%mod;
            }
        }
    }
    if(flag==-1){
        ll inv=qpow(tot,mod-2);
        for(int i=0;i<tot;i++)
            a[i]=1ll*a[i]*inv%mod;
    }
}

void poly_mul(ll a[],ll b[])
{
    ntt(a,1),ntt(b,1);
    for(int i=0;i<tot;i++)
        a[i]=a[i]*b[i]%mod;
    ntt(a,-1);
}

void poly_init()
{
    while((1<<bit)<n+m+1)
        bit++;
    tot=(1<<bit);
    for(int i=0;i<tot;i++)
        rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));
}
int main()
{
    /****使用方式****/
    poly_init(); //初始化蝴蝶变换数组
    poly_mul(a,b); //多项式a,b相乘,结果保存在a中
    return 0;
}
```

3.25 快速沃尔什变换 (FWT)

```
//NTT可以处理下标相加, 而FWT主要是处理下标的其他逻辑运算
//用法与NTT相同
//记得初始化bit与tot
int bit,tot;
void fwt_or(int a[],int op)
{
    for(int mid=1;mid<tot;mid<<=1){
        int len=mid<<1;
        for(int i=0;i<tot;i+=len)
            for(int j=0;j<mid;j++)
                (a[mid+i+j]+=a[i+j]*op+mod)%=mod;
    }
    return;
}

void fwt_and(int a[],int op)
{
    for(int mid=1;mid<tot;mid<<=1){
        int len=mid<<1;
        for(int i=0;i<tot;i+=len)
```

```
            for(int j=0;j<mid;j++)
               (a[i+j]+=a[i+j+mid]*op+mod)%=mod;
   }
   return;
}

void fwt_xor(int a[],int op)
{
   if(op==-1)
      op=qpow(2,mod-2);
   for(int mid=1;mid<tot;mid<<=1){
      int len=mid<<1;
      for(int i=0;i<tot;i+=len)
         for(int j=0;j<mid;j++){
            int x=a[i+j],y=a[mid+i+j];
            a[i+j]=(x+y)%mod;
            a[mid+i+j]=(x-y+mod)%mod;
            a[i+j]=op*a[i+j]%mod;
            a[mid+i+j]=a[mid+i+j]*op%mod;
         }
   }

   return;
}
```

3.26 常用组合公式

$$\sum_{k=1}^{n}(2k-1)^2=\frac{n(4n^2-1)}{3}$$
$$\sum_{k=1}^{n}k^3=\left(\frac{n(n+1)}{2}\right)^2$$
$$\sum_{k=1}^{n}(2k-1)^3=n^2(2n^2-1)$$
$$\sum_{k=1}^{n}k^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$
$$\sum_{k=1}^{n}k^5=\frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$
$$\sum_{k=1}^{n}k(k+1)=\frac{n(n+1)(n+2)}{3}$$
$$\sum_{k=1}^{n}k(k+1)(k+2)=\frac{n(n+1)(n+2)(n+3)}{4}$$
$$\sum_{k=1}^{n}k(k+1)(k+2)(k+3)=\frac{n(n+1)(n+2)(n+3)(n+4)}{5}$$

错排 (排列中所有的元素都在不正确的位置上的排列种数) 公式:
$D_n=n!\left(1-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+\cdots+\frac{(-1)^n}{n!}\right)=(n-1)(D_{n-2}-D_{n-1})$

3.27 Bertrand 猜想

对任意 $n>3$, 都存在 $n<p<2\times n$, 其中 p 为质数

3.28 威尔逊定理

$$(p-1)!\equiv -1(mod \quad p)$$

3.29 最小二乘法

$$k=\frac{\sum_{i=1}^{n}x_iy_i-n\cdot\overline{xy}}{\sum_{i=1}^{n}x_i^2-n\cdot(\overline{x})^2},\ b=\overline{y}-k\overline{x}$$

3.30 数相关结论

10^9 内所有数的最多因子个数为 1344 个 10^{18} 内最多有 103680 个
10^7 内所有数的因子大小的和最大约为 5×10^7

3.31 卡特兰数

3.32 斯特林数

表示将 n 个不同元素构成 m 个圆排列的数目

3.33 第二类斯特林数

表示将 n 个不同元素分成 m 个集合的方案数
$S(n,m)=\frac{1}{m!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(m-k)^n$

3.34 欧拉数

3.35 复数操作

3.36 康托展开

```
//O(nlogn)
//add与ask函数为树状数组操作
int n;
ll cantor()
{
   ll ans = 0;
   for (int i = 1; i <= n;i++){
      ans = (ans + (ask(a[i]) - 1) * fac[n - i] %
          mod) % mod;
      add(a[i], -1);
   }

   return ans;
}
void solve()
{
   cin >> n;
   fac[0]=1;
   for(int i=1;i<n;++i)
      fac[i] = fac[i - 1] * i % mod;

   for (int i = 1; i <= n;++i)
      add(i, 1);
   for (int i = 1; i <= n;++i)
      cin >> a[i];

```

```
    cout << cantor() + 1 << '\n';
    return ;
}
```

3.37 逆康托展开

```
//排名为x的排列
vector<int> incantor(int x,int n){
    x--;
    vector<int> res;
    int cnt;
    for(int i = 0;i < n; ++i){
        cnt = x / fact[n - i - 1];
        x %= fact[n - i - 1];

        /*------寻找第cnt+1大数的大小,记作ans--------*/
        /*-----可使用权值线段树二分或者树状数组倍增-----*/
        res.emplace_back(ans);
    }
    return res;
}
```

3.38 生成函数

指数型生成函数适用于解决多重集选排列问题。
常用替换式:

$$\sum_{i\geq 0}\frac{x^n}{n!}=e^x$$

$$1+\frac{x^2}{2!}+\frac{x^4}{4!}+\cdots=\frac{e^x+e^{-x}}{2}$$

$\frac{x^m}{m!}$ 前的系数即为选 m 个元素排列的值

3.39 拉格朗日插值

4 数据结构

4.1 并查集

```
int fa[N];
int get(int x)
{
    if(x==fa[x])
        return x;
    return fa[x]=get(fa[x]);
}

void merge(int x,int y)
{
    int a=get(x);
    int b=get(y);
    if(a==b)
        return;
    fa[b]=a;
    return;
}

//初始化
for(int i=1;i<=n;i++)
    fa[i]=i;
```

4.2 并查集跳跃

```
//当每个元素的操作次数有上限时，此方法可以保证不重复操作
for (int i = 1;i <= n + 1;i++)
    fa[i] = i;
if(EQUAL)
    fa[i] = get(i + 1);
for (int i = get(l);i <= r;i = get(i + 1)) {
    /*操作*/
    if (EQUAL)
        fa[i] = get(i + 1);
}
```

4.3 可持久化并查集

4.4 树状数组

```
//树状数组主要是用来处理前缀和
//除前缀和外,所有前缀性质都可以维护
//如前缀最值等操作
#define lowbit(x) (x&(-x))
const int N=200005;
int c[N];
void add(int p,int x)
{
    for(int i=p;i<=N;i+=lowbit(i))
        c[i]+=x; //或其他操作
    return;
}

int ask(int p)
{
    int ans=0;
    for(int i=p;i;i-=lowbit(i))
        ans+=c[i]; //或其他操作

    return ans;
}
```

4.5 线段树

```
//以区间和为例
const int N=100010;
typedef long long ll;

int a[N];
struct{
    int l,r;
    ll add,sum;
}tr[N*4];
void pushup(int p)
{
    tr[p].sum=tr[p<<1].sum+tr[p<<1|1].sum;
    return;
}
void build(int p,int l,int r)
{
    if(l==r){
        tr[p].l=tr[p].r=l;
```

```
        tr[p].sum=a[l];
        return;
    }

    int mid=(l+r)>>1;
    tr[p].l=l;
    tr[p].r=r;
    build(p<<1,l,mid);
    build(p<<1|1,mid+1,r);
    pushup(p);
    return;
}
void pushdown(int p)
{
    if(tr[p].l==tr[p].r)
        return;
    tr[p<<1].add+=tr[p].add;
    tr[p<<1|1].add+=tr[p].add;

    tr[p<<1].sum+=1ll*(tr[p<<1].r-tr[p<<1].l+1)*tr[p].add;
    tr[p<<1|1].sum+=1ll*(tr[p<<1|1].r-tr[p<<1|1].l+1)*tr[p].add;
    tr[p].add=0;
    return;
}
void change(int p,int l,int r,int x)
{
    pushdown(p);
    if(l<=tr[p].l&&r>=tr[p].r){
        //打标记,计算该线段修改后的值
        tr[p].add+=x;
        tr[p].sum+=1ll*x*(tr[p].r-tr[p].l+1);
        return;
    }

    int mid=(tr[p].r+tr[p].l)>>1;
    if(l<=mid)
        change(p<<1,l,r,x);
    if(r>mid)
        change(p<<1|1,l,r,x);

    pushup(p);
    return;
}

ll ask(int p,int l,int r)
{
    pushdown(p);
    ll sum=0;
    if(l<=tr[p].l&&r>=tr[p].r)
        return tr[p].sum;

    int mid=(tr[p].l+tr[p].r)>>1;
    if(l<=mid)
        sum+=ask(p<<1,l,r);
    if(r>mid)
        sum+=ask(p<<1|1,l,r);

    return sum;

}
```

4.6 李超线段树

```
//支持添加线性函数，询问每个横坐标中值最大的函数编号
//注意精度，能用 int 尽量用
//本题以编号最小为例，使用时仅需要add线段
const int N = 100005;
const double eps = 1e-9;
struct node {
    int l, r, id;
}tr[N << 2];

int cmp(double x,double y)
{
    if (x - y > eps)
        return 1;
    if (y - x > eps)
        return -1;
    return 0;
}
int cnt;
struct line {
    double k, b;
}seg[N];

double f(int id, int x)
{
    return seg[id].k * x + seg[id].b;
}

void build(int p, int l, int r)
{
    if (l == r) {
        tr[p].l = tr[p].r = l;
        return;
    }
    tr[p].l = l;
    tr[p].r = r;
    int mid = (l + r) >> 1;
    build(p << 1, l, mid);
    build(p << 1 | 1, mid + 1, r);
    return;
}

void update(int p, int id)
{
    int& v = tr[p].id;
    int u = id;
    int mid = (tr[p].l + tr[p].r) >> 1;
    if (cmp(f(u, mid), f(v, mid)) == 1)
        swap(u, v);
    int tagl = cmp(f(u, tr[p].l), f(v, tr[p].l));
    int tagr = cmp(f(u, tr[p].r), f(v, tr[p].r));

    //其中 u < v 含义为保留标号最小
    if (tagl == 1 || (!tagl && u < v))
        update(p << 1, u);
    if (tagr == 1 || (!tagr && u < v))
        update(p << 1 | 1, u);
    return;
}

void change(int p, int l, int r, int u)
{
    if (l <= tr[p].l && r >= tr[p].r) {
        update(p, u);
```

```
        return;
    }
    int mid = (tr[p].l + tr[p].r) >> 1;
    if (l <= mid)
        change(p << 1, l, r, u);
    if (r > mid)
        change(p << 1 | 1, l, r, u);
}
void add(int x0, int y0, int x1, int y1)
{
    cnt++;
    if (x0 == x1) {
        seg[cnt].k = 0;
        seg[cnt].b = max(y0, y1);
    }
    else {
        seg[cnt].k = 1. * (y1 - y0) / (x1 - x0);
        seg[cnt].b = y0 - seg[cnt].k * x0;
    }
    change(1, x0, x1, cnt);
}

//ask 时会遍历所有区间，可以进行标号最小操作等询问
//以标号最小为例
typedef pair<double, int> pdi;
pdi get_max(pdi a, pdi b)
{
    if (cmp(a.first, b.first) == 1)
        return a;
    if (cmp(a.first, b.first) == -1)
        return b;
    if (a.second > b.second)
        return b;
    else
        return a;
}
pdi ask(int p, int x)
{
    //注意编号
    pdi now = { f(tr[p].id, x) ,tr[p].id };
    if (tr[p].l == tr[p].r)
        return now;

    int mid = (tr[p].l + tr[p].r) >> 1;
    if (x <= mid && x >= tr[p].l)
        return get_max(now, ask(p << 1, x));
    else
        return get_max(now, ask(p << 1 | 1, x));
}
```

4.7 主席树

```
//以区间第K大数为例
//idx根节点编号,每次修改都会建立一个新的根节点
const int N=100005;
int idx ,root[N];
struct{
    int l,r,cnt;
}tr[N*4+N*17];
int a[N];

int build(int l,int r)
{
    int p=++idx;
    if(l==r)
        return p;

    int mid=(l+r)>>1;
    tr[p].l=build(l,mid);
    tr[p].r=build(mid+1,r);

    return p;
}

int change(int p,int l,int r,int x)
{
    int q=++idx;
    tr[q]=tr[p];
    if(l==r){
        tr[q].cnt++;
        return q;
    }

    int mid=(l+r)>>1;
    if(x<=mid)
        tr[q].l=change(tr[p].l,l,mid,x);
    else
        tr[q].r=change(tr[p].r,mid+1,r,x);

    tr[q].cnt=tr[tr[q].l].cnt+tr[tr[q].r].cnt;

    return q;
}

int ask(int p,int q,int l,int r,int k)
{
    if(l==r)
        return l;

    int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;

    int mid=(l+r)>>1;
    if(cnt>=k)
        return ask(tr[p].l,tr[q].l,l,mid,k);
    else
        return ask(tr[p].r,tr[q].r,mid+1,r,k-cnt);

}
int main()
{
    /*---主席树操作---*/
    root[0]=build(0,num.size()-1);

    //在第i-1代树上添加a[i],得到第i代树
    for(int i=1;i<=n;i++)
        root[i]=change(root[i-1],0,num.size()-1,a[i]);

    ans=ask(root[l-1],root[r],0,num.size()-1,k);
    return 0;
}
```

4.8 动态开点线段树

```
```

4.9 线段树分裂与合并

```
```

4.10 平衡树

4.11 Splay

4.12 AC 自动机

4.13 分块

```cpp
//分块中的预处理
//区间处理时，可以先特判做右端点在同一块中，后从pos[L]+1
    到pos[R]-1处理，最后再处理两端的小段
int st[N], ed[N];
int pos[N];
int block, cnt;
void init(int n)
{
    block = sqrt(n);
    cnt = n / block + bool(n % block);

    for (int i = 1;i <= cnt;i++) {
        st[i] = (i - 1) * block + 1;
        ed[i] = i * block;
    }

    ed[cnt] = n;
    for (int i = 1;i <= n;i++)
        pos[i] = (i - 1) / block + 1;
}
```

4.14 莫队

```cpp
//程序基本上只需要添加辅助数组以及编写add与del函数即可
const int N=50000;
int a[N],belong[N];
struct query{
    int l,r,id;
}q[N];

int cmp(const query& a,const query& b)
{
    if(belong[a.l]^belong[b.l])
        return belong[a.l]<belong[b.l];
    else if(belong[a.l]&1)
        return a.r<b.r;
    else
        return a.r>b.r;
}

int now; //记录当前答案
void add(int pos)
{
    //添加第pos位后的答案
    //操作省略
}

void del(int pos)
{
    //删除第pos位的答案
    //操作省略
}

int main()
{
    int n,m; //n为数据个数, m为询问数
    read(n,m);
    int sz=sqrt(n);
    int bnum=ceil(1.*n/sz);
    for(int i=1;i<=bnum;i++)
        for(int j=(i-1)*sz+1;j<=i*sz;j++)
            belong[j]=i;

    //读入原始数据
    for(int i=1;i<=n;i++)
        read(a[i]);

    for(int i=1;i<=m;i++){
        read(q[i].l,q[i].r);
        q[i].id=i;
    }

    sort(q+1,q+m+1,cmp);
    int l=1,r=0;

    for(int i=1;i<=m;i++){
        int ql=q[i].l,qr=q[i].r;
        while(l<ql)
            del(l++);
        while(l>ql)
            add(--l);
        while(r<qr)
            add(++r);
        while(r>qr)
            del(r--);

        ans[q[i].id]= now;
    }

    for(int i=1;i<=m;i++)
        cout<<ans[i]<<'\n';
    return 0;
}
```

4.15 点分治

4.16 CDQ 分治

4.17 动态树

4.18 左偏树

4.19 仙人掌

4.20 哈希

```
//h1 与 h2 分别储存两次的哈希值
//set 用来判重
//模数与底数选取质数
//hash[l,r]=hash[r]-hash[l-1]*(base^(r-l+1))

const ll mod1 = /*private*/;
const ll mod2 = /*private*/;
const ll base = /*private*/;
set< pair< ll, ll > > st;
ll h1[N];
ll h2[N];

for (int j = 0; j < m; j++) {
    h1[i] = (h1[i] * base + s[j]) % mod1;
    h2[i] = (h2[i] * base + s[j]) % mod2;
}
```

4.21 KMP 模式匹配

```
//f[i]表示B中以i结尾的子串与A的前缀能够匹配的最长长度
//Next[i]表示A中以i结尾的非前缀子串与A的前缀能够匹配的最长长度
//下标从1开始
//[f[i]==n] 时即为B在A中第一次出现
const int N=100005;
string a,b;
int Next[N],f[N];
void pre()
{
    Next[1]=0;
    for(int i=2,j=0;i<=n;i++){
        while(j>0&&a[i]!=a[j+1])
            j=Next[j];
        if(a[i]==a[j+1])
            j++;
        Next[i]=j;
    }

    for(int i=1,j=0;i<=m;i++){
        while(j>0&&(j==n||b[i]!=a[j+1]))
            j=Next[j];
        if(b[i]==a[j+1])
            j++;
        f[i]=j;
    }
}
```

4.22 扩展 KMP 算法

4.23 manacher 算法

```
//N的大小为字符串二倍
//p[i]中存的是以i为中心字符的回文串半径(中心字符不算)
//p[i]-1即为回文串长度
const int N=20000005;
int p[N];
void manacher(string& s)
{
    int l=0,r=0;
    int n=s.size();
    string use="|";
    for(int i=0;i<n;i++)
        use+="#",use+=s[i];
    use+="#^";

    for(int i=1;i<use.size();i++){
        if(i<=r)
            p[i]=min(p[l+r-i],r-i+1);
        while(use[i+p[i]]==use[i-p[i]])
            p[i]++;
        if(p[i]+i-1>r)
            l=i-p[i]+1,r=i+p[i]-1;
    }

    return;
}
```

4.24 Trie 树

```
//以字符串出现次数为例
int tr[100005][30];
int cnt[100005];
int tot;
void insert(const string& s)
{
    int p=0;
    for(auto c:s){
        if(!tr[p][c-'a'])
            tr[p][c-'a']=++tot,p=tot;
        else
            p=tr[p][c-'a'];
    }

    cnt[p]++;
}

int ask(const string& s)
{
    int p=0;
    for(auto c:s){
        if(!tr[p][c-'a'])
            return 0;
        else
            p=tr[p][c-'a'];
    }

    return cnt[p];
}
```

4.25 可持久化 Trie 树

4.26 后缀数组

定义编号为 i 的后缀与编号为 j 的后缀的最长前缀长度为 $LCP(i,j)$
$LCP(i,j) = min_{i+1 \leqslant p \leqslant j} height[p]$

字符串中不同子串的数目为每一个后缀的长度减去其 height 之和
判断子串：
跑出 sa，然后从最小的后缀开始，一个个往后枚举，记录下当前匹配到的位置，如果匹配不上就下一个后缀，否则位置向后移一位。如果枚举完了后缀还没有完全匹配则不是原串子串。

两串的最长公共子串：将两串拼接，求出 sa 和 height。枚举 sa，对于每个串找到其后第一个“起点在后一个串上的后缀”，求出 LCP 后取最大

```
const int N = 1000010;

int n, m;
//rk数组存放编号为i的后缀的排名
//sa数组存放排名为i的后缀的编号
//height数组存放排名为i的后缀与排名为i-1的后缀的最长相同
    前缀长度
int sa[N], x[N], y[N], c[N], rk[N], height[N];

void get_sa(const string& s)
{
    for (int i = 1; i <= n; i ++ )
        c[x[i] = s[i]] ++ ;
    for (int i = 2; i <= m; i ++ )
        c[i] += c[i - 1];
    for (int i = n; i; i -- )
        sa[c[x[i]] -- ] = i;

    for (int k = 1; k <= n; k <<= 1){
        int num = 0;
        for (int i = n - k + 1; i <= n; ++i )
            y[ ++ num] = i;
        for (int i = 1; i <= n; ++i )
            if (sa[i] > k)
                y[ ++ num] = sa[i] - k;
        for (int i = 1; i <= m; ++i )
            c[i] = 0;
        for (int i = 1; i <= n; ++i )
            ++c[x[i]] ;
        for (int i = 2; i <= m; ++i )
            c[i] += c[i - 1];
        for (int i = n; i; --i ){
            sa[c[x[y[i]]] -- ] = y[i];
            y[i] = 0;
        }

        swap(x, y);
        x[sa[1]] = 1;
        num = 1;
        for (int i = 2; i <= n; ++i )
            x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[
                sa[i] + k] == y[sa[i - 1] + k]) ? num :
                 ++ num;
        if (num == n)
            break;
        m = num;
    }

    return;
}

void get_height(const string& s)
{
    for (int i = 1; i <= n; i ++ )
        rk[sa[i]] = i;
    for (int i = 1, k = 0; i <= n; i ++ ){
        if (rk[i] == 1)
            continue;

        if (k)
            k -- ;
        int j = sa[rk[i] - 1];
        while (i + k <= n && j + k <= n && s[i + k] ==
            s[j + k])
            k ++ ;

        height[rk[i]] = k;
    }
    return;
}

int main()
{
    string s;
    cin>>s;
    n = s.size();

    //m为字符元素的最大值
    m = 122;

    s.insert(s.begin(),'&');
    get_sa(s);
    get_height(s);
}
```

4.27 后缀自动机

```
//注意修改 insert 中的标识字符
//tot 和 last 的初始值为 1
//np 代表的是前缀所在的等价类
//若计算每个节点处出现的次数，则将取消注释处的注释后dfs
const int N = 100005;
struct node {
    int ch[26];
    int len, fa;
}sam[N << 1];
int tot = 1, last = 1;
//int f[N << 1];
void insert(char cc)
{
    int c = cc - 'A';
    int p = last;
    int np = last = ++tot;
//  f[np] = 1;
    sam[np].len = sam[p].len + 1;
    memset(sam[np].ch, 0, sizeof sam[np].ch);
    for (;p && !sam[p].ch[c];p = sam[p].fa)
        sam[p].ch[c] = np;
    if (!p)
        sam[np].fa = 1;
    else {
        int q = sam[p].ch[c];
        if (sam[q].len == sam[p].len + 1)
            sam[np].fa = q;
```

```cpp
        else {
            int nq = ++tot;
            sam[nq] = sam[q];
            sam[nq].len = sam[p].len + 1;
            sam[q].fa = sam[np].fa = nq;
            for (;p && sam[p].ch[c] == q;p = sam[p].fa)
                sam[p].ch[c] = nq;
        }
    }
}
void solve()
{
    int n, m;
    cin >> n >> m;
    string s;
    cin >> s;
    memset(sam[1].ch, 0, sizeof sam[1].ch);
    tot = last = 1;
    for (const auto& c : s)
        insert(c);

}
```

4.28 回文自动机

4.29 lyndon 分解

4.30 笛卡尔树

4.31 Dance Links 精确覆盖

```cpp
//选法作为行，限制作为列
//精确覆盖是指从中选取一些行，使得每一列有且仅有一个1
//注意，只能解决限制为1的问题
//与网络流较为类似，关键在于如何构建矩阵
//ans中存选哪些行
const int N = 5510;

int n, m;
int l[N], r[N], u[N], d[N], s[N], row[N], col[N], idx;
int ans[N], top;

void init()
{
    for (int i = 0; i <= m; i ++ )
    {
        l[i] = i - 1, r[i] = i + 1;
        u[i] = d[i] = i;
    }
    l[0] = m, r[m] = 0;
    idx = m + 1;
}

void add(int& hh, int& tt, int x, int y)
{
    row[idx] = x, col[idx] = y, s[y] ++ ;
    u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
    r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;
    tt = idx ++ ;
}

void remove(int p)
{
    r[l[p]] = r[p], l[r[p]] = l[p];
    for (int i = d[p]; i != p; i = d[i])
        for (int j = r[i]; j != i; j = r[j]){
            s[col[j]] -- ;
            u[d[j]] = u[j], d[u[j]] = d[j];
        }
}

void resume(int p)
{
    for (int i = u[p]; i != p; i = u[i])
        for (int j = l[i]; j != i; j = l[j]){
            u[d[j]] = j, d[u[j]] = j;
            s[col[j]] ++ ;
        }
    r[l[p]] = p, l[r[p]] = p;
}

bool dfs()
{
    if (!r[0])
        return true;
    int p = r[0];
    for (int i = r[0]; i; i = r[i])
        if (s[i] < s[p])
            p = i;
    remove(p);
    for (int i = d[p]; i != p; i = d[i]){
        ans[ ++ top] = row[i];
        for (int j = r[i]; j != i; j = r[j])
            remove(col[j]);

        if (dfs())
            return true;
        for (int j = l[i]; j != i; j = l[j])
            resume(col[j]);
        top -- ;
    }
    resume(p);
    return false;
}

void solve()
{
    cin >> n >> m;
    init();
    for (int i = 1; i <= n;i++){

        //每次插入新行时都需要执行
        int hh = idx, tt = idx;
        for (int j = 1; j <= m;j++){
            int x;
            cin >> x;

            //只有1需要插入
            if(x)
```

```
            add(hh, tt, i, j);
        }
    }

    if(dfs()){
        for (int i = 1; i <= top;i++)
            cout << ans[i] << " \n"[i == top];
    }
    else
        cout << "No Solution!\n";

    return ;
}
```

4.32 Dance Links 重复覆盖

```
//重复覆盖解决的是选出行的数量最小问题，并且可以重复覆盖
//需保证答案较小，因为基于IDA*算法
const int N = 10010;

int n, m;
int l[N], r[N], u[N], d[N], col[N], row[N], s[N], idx;
int ans[N];
bool st[110];

void init()
{
    for (int i = 0; i <= m; i ++ ){
        l[i] = i - 1, r[i] = i + 1;
        col[i] = u[i] = d[i] = i;
        s[i] = 0;
    }
    l[0] = m, r[m] = 0;
    idx = m + 1;
}

void add(int& hh, int& tt, int x, int y)
{
    row[idx] = x, col[idx] = y, s[y] ++ ;
    u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
    r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;
    tt = idx ++ ;
}

int h()
{
    int cnt = 0;
    memset(st, 0, sizeof st);
    for (int i = r[0]; i; i = r[i]){
        if (st[col[i]])
            continue;
        cnt ++ ;
        st[col[i]] = true;
        for (int j = d[i]; j != i; j = d[j])
            for (int k = r[j]; k != j; k = r[k])
                st[col[k]] = true;
    }
    return cnt;
}

void remove(int p)
{
    for (int i = d[p]; i != p; i = d[i]){
        r[l[i]] = r[i];
        l[r[i]] = l[i];
    }
}

void resume(int p)
{
    for (int i = u[p]; i != p; i = u[i]){
        r[l[i]] = i;
        l[r[i]] = i;
    }
}

bool dfs(int k, int depth)
{
    if (k + h() > depth)
        return false;
    if (!r[0])
        return true;
    int p = r[0];
    for (int i = r[0]; i; i = r[i])
        if (s[p] > s[i])
            p = i;

    for (int i = d[p]; i != p; i = d[i]){
        ans[k] = row[i];
        remove(i);
        for (int j = r[i]; j != i; j = r[j])
            remove(j);
        if (dfs(k + 1, depth))
            return true;
        for (int j = l[i]; j != i; j = l[j])
            resume(j);
        resume(i);
    }
    return false;
}

void solve()
{
    cin >> n >> m;
    init();
    for (int i = 1; i <= n; i ++ ){
        int hh = idx, tt = idx;
        for (int j = 1; j <= m; j ++ ){
            int x;
            cin >> x;
            if (x)
                add(hh, tt, i, j);
        }
    }

    int depth = 0;
    while (!dfs(0, depth))
        depth ++ ;
    cout << depth << '\n';
    for (int i = 0; i < depth; i ++ )
        cout << ans[i] << '\n';
    return;
}
```

5 动态规划

5.1 0/1 背包

```
const int N=105;
const int M=100005;
int f[M];
int v[N],w[N];
int main()
{
    memset(f,0,sizeof f);
    //memset(f,0xcf,sizeof f);
    f[0]=0;
    for(int i=1;i<=n;i++)
        for(int j=m;j>=v[i];j--)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
}
```

5.2 完全背包

```
const int N=105;
const int M=100005;
int f[M];
int v[N],w[N];
int main()
{
    memset(f,0,sizeof f);
    //memset(f,0xcf,sizeof f);
    f[0]=0;
    for(int i=1;i<=n;i++)
        for(int j=v[i];j<=m;j++)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
}
```

5.3 多重背包

5.4 分组背包

5.5 回退背包

```
//适用于计算方案数问题
//退去第 x 个物品后满足总价值为 i 的方案数
//01背包
for(int i = w[x];i <= m;++i)
    dp[i] -= dp[i - w[x]];

//多重背包
for(int i = m;i >= w[x];--i)
    dp[i] -= dp[i - w[x]];
```

5.6 高维前缀和 SOSDP

```
//相当于每一位是 0/1 的 bit 维前缀和
for (int i= 0; i < (1 << bit); ++i)
    f[i] = a[i];
for (int i= 0; i < bit; ++i){
    for (int mask= 0; mask < (1 << bit); ++mask) {
        if (mask & (1 << i))
            f[mask] += f[mask ^ (1 << i)];
    }
}
```

5.7 换根 DP

5.8 状压 DP

```
//取出x的第i位
y=(x>>(i-1))&1;
//将x第i位取反
x^=1<<(i-1);
//将x第i位变为1
x|=1<<(i-1);
//将x第i位变为0
x&= ~(1<<(i-1));
//将x最靠右的1变成0
x=x&(x-1);
//取出x最靠右的1
y=x&(~x);
//把最靠右的0变成1
x|=(x+1);
//判断是否有两个连续的1, n个连续的1与之类似
if(x&(x<<1))
    cout<<"YES\n";
//枚举子集
for(int i=sta;i;i=((i-1)&sta)){
    //i即为子集
}
```

5.9 四边形不等式优化 DP

5.10 斜率优化 DP

6 计算几何

6.1 基础操作

7 博弈

7.1 Nim 游戏

7.2 反 Nim 游戏

7.3 SG 函数

7.4 纳什均衡

8 杂项算法

8.1 离散化

```
vector<int> nums;
sort(nums.begin(),nums.end());
nums.erase(unique(nums.begin(),nums.end()),nums.end());

int get(int x)
{
   return lower_bound(nums.begin(),nums.end(),x)-nums.begin();
}
```

8.2 二分

```
//>= x的数中最小的一个
while(l<r){
   int mid=(l+r)>>1;
   if(a[mid]>=x)
      r=mid;
   else
      l=mid+1;
}

//<= x的数中最大的一个
while(l<r){
   int mid=(l+r+1)>>1;
   if(a[mid]<=x)
      l=mid;
   else
      r=mid-1;
}

//结果储存在l中
```

8.3 三分

```
//整数三分
int l = 1,r = 100;
while(l < r) {
   int lmid = l + (r - l) / 3;
   int rmid = r - (r - l) / 3;
   lans = f(lmid),rans = f(rmid);
   // 求凹函数的极小值
   if(lans <= rans)
      r = rmid - 1;
   else
      l = lmid + 1;
   // 求凸函数的极大值
   if(lans >= rans)
      l = lmid + 1;
   else
      r = rmid - 1;
}
// 求凹函数的极小值
cout << min(lans,rans) << endl;
// 求凸函数的极大值
cout << max(lans,rans) << endl;


//浮点三分
const double EPS = 1e-9;
while(r - l > EPS) {
   double lmid = l + (r - l) / 3;
   double rmid = r - (r - l) / 3;
   lans = f(lmid),rans = f(rmid);
   // 求凹函数的极小值
   if(lans <= rans)
      r = rmid;
   else
      l = lmid;
   // 求凸函数的极大值
   if(lans >= rans)
      l = lmid;
   else
      r = rmid;
}
// 输出 l 或 r 都可
cout << l << endl;
```

8.4 倍增

8.5 ST 表

```
//解决可重复问题
const int N=100005;
int f[N][20];
void pre()
{
   for(int i=1;i<=n;i++)
      f[i][0]=a[i];

   int t=log(n)/log(2)+1;
   for(int j=1;j<t;j++)
      for(int i=1;i<=n-(1<<j)+1;i++)
         f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
}

int ask(int l,int r)
{
   int k=log(r-l+1)/log(2);
   return max(f[l][k],f[r-(1<<k)+1][k]);
}
```

8.6 启发式合并

每次合并均将小集合合并至大集合中
时间复杂度 O(nlogn)

8.7 dsu on tree

```
const int N=100005;
const int M=200005;
/*---建树操作省略---*/
int sz[N],son[N];
ll sum,cnt[N];
int mx=0;
bool v[N];
void dfs_son(int x)
{
    v[x]=1;
    sz[x]=1;
    for(int i=head[x];i;i=Next[i]){
        int y=ver[i];
        if(v[y])
            continue;
        dfs_son(y);
        sz[x]+=sz[y];
        if(sz[y]>sz[son[x]])
            son[x]=y;
    }

    return;
}

ll ans[N];
void update(int x,int father,int flag,int pson)
{
    /*维护并统计答案
    此处以出现次数最多元素编号之和为例
    int color=c[x];
    cnt[color]+=flag;
    if(cnt[color]>mx)
        mx=cnt[color],sum=color;
    else if(cnt[color]==mx)
        sum+=color;
    */
    for(int i=head[x];i;i=Next[i]){
        int y=ver[i];
        if(y==father||y==pson)
            continue;
        update(y,x,flag,pson);
    }

    return;
}

void dfs(int x,int father,int op)
{
    for(int i=head[x];i;i=Next[i]){
        int y=ver[i];
        if(y==father||y==son[x])
            continue;

        dfs(y,x,0);
    }

    if(son[x])
        dfs(son[x],x,1);
    update(x,father,1,son[x]);

    ans[x]=sum;
    if(!op)
        update(x,father,-1,0),sum=mx=0;

    return;

}
int main()
{
    //主要操作过程在dfs()中实现
    //dfs_son()仅为预处理
    dfs_son(1);
    dfs(1,0,1);
}
```

8.8 切比雪夫距离与曼哈顿距离转化

切比雪夫距离：$max(|x_1 - x_2|, |y_1 - y_2|)$
切比雪夫距离转换为曼哈顿距离：坐标变换为 $(x + y, x - y)$ 反之：坐标变换为 $(\frac{x+y}{2}, \frac{x-y}{2})$

8.9 高精度加法

```
```

8.10 高精度减法

```
```

8.11 高精度乘法

```
```

8.12 高精度除法

```
```

8.13 NTT 优化高精度乘法

```
```