ACM 模板 Ver 1.1

图论

树链剖分

```
int dfc, dfn[N], rnk[N], siz[N], top[N], dep[N], son[N], faz[N];
void dfs1(int u, int fa) {
    dep[u] = dep[fa] + 1;
    siz[u] = 1;
    son[u] = -1;
    faz[u] = fa;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if (son[u] == -1 || siz[son[u]] < siz[v]) son[u] = v;
    }
}
void dfs2(int u, int fa, int tp) {
    dfn[u] = ++dfc;
    rnk[dfc] = u;
    top[u] = tp;
    if (son[u] != -1) dfs2(son[u], u, tp);
    for (int v : G[u]) {
        if (v == fa || v == son[u]) continue;
        dfs2(v, u, v);
    }
}
int LCA(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]])
            u = faz[top[u]];
        else
            v = faz[top[v]];
    }
    return dep[u] > dep[v] ? v : u;
}
```

二分图匹配

```
int mch[maxn], vis[maxn];
std::vector<int> e[maxn];
bool dfs(const int u, const int tag) {
    for (auto v : e[u]) {
        if (vis[v] == tag) continue;
        vis[v] = tag;
        if (!mch[v] || dfs(mch[v], tag)) return mch[v] = u, 1;
    }
    return 0;
}
int main() {
```

```
    int ans = 0;
    for (int i = 1; i <= n; ++i) if (dfs(i, i)) ++ans;
}
```

*最大权完美二分图匹配

转化为费用流模型

在图中新增一个源点和一个汇点。

从源点向二分图的每个左部点连一条流量为 1，费用为 0 的边，从二分图的每个右部点向汇点连一条流量为 1，费用为 0 的边。

接下来对于二分图中每一条连接左部点 u 和右部点 v，边权为 w 的边，则连一条从 u 到 v，流量为 1，费用为 w 的边。

求这个网络的 最大费用最大流 即可得到答案。

有向图最小路径覆盖问题

1. 如果每个顶点只能在某一条路径上，对每条边(u, v)，构造新图，将 u -> v + n。新图为二分图，并且 n - 最大匹配 就是答案。
2. 如果顶点可以在不同的路径中，那么如果 (u, v) 可达，则存在边。

```
int n, m;
bitset<N> f[N];
int vis[N], mch[N];
bool dfs(int u, int dfc) {
 for (int v = 1; v <= n; v++) if (v != u && vis[v] != dfc && f[u][v]) {
 vis[v] = dfc;
 if (!mch[v] || dfs(mch[v], dfc)) return mch[v] = u, 1;
 }
 return 0;
}

void solve() {
 memset(vis, 0, sizeof vis);
 memset(mch, 0, sizeof mch);
 for (int i = 1; i <= n; i++) f[i].reset();
 for (int i = 1; i <= m; i++) {
 int u, v;
 scanf("%d %d", &u, &v);
 f[u].set(v);
 }
 for (int k = 1; k <= n; k++) {
 for (int i = 1; i <= n; i++) if (f[i][k]) f[i] |= f[k];
 }
 int res = n;
 for (int i = 1; i <= n; i++) res -= dfs(i, i);
 printf("%d\n", res);
}
```

最大流

```
#include <bits/stdc++.h>
using i64 = long long;

#define int long long
const int N = 1e5 + 5;
int head[N], cur[N], ecnt, d[N], s, t, n, m;
bool vis[N];
struct Edge {
    int nxt, v, flow, cap;
}e[N << 1];
void add_edge(int u, int v, int flow, int cap) {
    e[ecnt] = {head[u], v, flow, cap}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0}; head[v] = ecnt++;
}
bool bfs() {
    memset(vis, 0, sizeof vis);
    std::queue<int> q;
    q.push(s);
    vis[s] = 1;
    d[s] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if (vis[v] || e[i].flow >= e[i].cap) continue;
            d[v] = d[u] + 1;
            vis[v] = 1;
            q.push(v);
        }
    }
    return vis[t];
}
int dfs(int u, int a) {
    if (u == t || !a) return a;
    int b = 0;
    int flow = 0, f;
    for (int& i = cur[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (d[u] + 1 == d[v] && (f = dfs(v, std::min(a, e[i].cap - e[i].flow))) >
0) {
            e[i].flow += f;
            e[i ^ 1].flow -= f;
            flow += f;
            a -= f;
            if (!a) break;
        }
    }
    return flow;
}

signed main() {
    memset(head, -1, sizeof head);
```

```
    //freopen("in.txt", "r", stdin);
    scanf("%lld %lld %lld %lld", &n, &m, &s, &t);
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        scanf("%lld %lld %lld", &u, &v, &w);
        add_edge(u, v, 0, w);
    }
    int ans = 0;
    while (bfs()) {
        for (int i = 0; i <= n; i++) cur[i] = head[i];
        ans += dfs(s, 1000000000);
    }
    printf("%lld\n", ans);
    return 0;
}
```

最小费用最大流

```
#include <bits/stdc++.h>
using i64 = long long;

const int N = 5000 + 5;
const int inf = 1e9;
int head[N], cur[N], ecnt, dis[N], s, t, n, m, mincost;
bool vis[N];
struct Edge {
    int nxt, v, flow, cap, w;
}e[100002];
void add_edge(int u, int v, int flow, int cap, int w) {
    e[ecnt] = {head[u], v, flow, cap, w}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0, -w}; head[v] = ecnt++;
}
bool spfa(int s, int t) {
    memset(vis, 0, sizeof vis);
    std::fill(dis + 1, dis + n + 1, inf);
    std::queue<int> q;
    q.push(s);
    dis[s] = 0;
    vis[s] = 1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if (e[i].flow < e[i].cap && dis[u] + e[i].w < dis[v]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) vis[v] = 1, q.push(v);
            }
        }
    }
    return dis[t] != inf;
}
int dfs(int u, int a) {
    if (vis[u]) return 0;
```

```
    if (u == t || !a) return a;
    vis[u] = 1;
    int flow = 0, f;
    for (int& i = cur[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (dis[u] + e[i].w == dis[v] && (f = dfs(v, std::min(a, e[i].cap - 
e[i].flow))) > 0) {
            e[i].flow += f;
            e[i ^ 1].flow -= f;
            flow += f;
            mincost += e[i].w * f;
            a -= f;
            if (!a) break;
        }
    }
    vis[u] = 0;
    return flow;
}

signed main() {
    memset(head, -1, sizeof head);
    //freopen("in.txt", "r", stdin);
    scanf("%d %d %d %d", &n, &m, &s, &t);
    for (int i = 1; i <= m; i++) {
        int u, v, w, ww;
        scanf("%d %d %d %d", &u, &v, &w, &ww);
        add_edge(u, v, 0, w, ww);
    }
    int ans = 0;
    while (spfa(s, t)) {
        for (int i = 0; i <= n; i++) cur[i] = head[i];
        ans += dfs(s, inf);
    }
    printf("%d %d\n", ans, mincost);
    return 0;
}
```

最大闭权子图

正权点向 S 连边， 负权点向 T 连边。边权为点权的绝对值。原图的边容量设为 $+\infty$ 。

则最大收益为 $\sum_{v>0} v - mincost$

在最大闭权子图中的点是残量网络中 S 能到达的点。

树哈希

```
#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;

const int N = 1e6 + 5;
const ull mask = chrono::steady_clock::now().time_since_epoch().count();

ull shift(ull x) {
```

```
    x ^= mask;
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    x ^= mask;
    return x;
}
int n;
ull H[N];
vector<int> G[N];
set<ull> s;

void dfs(int u, int fa) {
    H[u] = 1;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs(v, u);
        H[u] += shift(H[v]);
    }
    s.emplace(H[u]);
}

void solve() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1, 0);
    printf("%d\n", (int)(s.size()));
}

int main() {
    int T = 1;
    while (T--) {
        solve();
    }
    return 0;
}
```

强连通分量

```
int dfn[N], low[N], dfs_clock, stk[N], top, scc, sccno[N], siz[N];
bool in_stk[N];
vector<int> e[N];
void tarjan(int u) {
    in_stk[u] = 1;
    stk[++top] = u;
    dfn[u] = low[u] = ++dfs_clock;
    for (int v : e[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
```

```cpp
        } else if (in_stk[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        int x;
        scc++;
        while (true) {
            x = stk[top];
            top--;
            sccno[x] = scc;
            in_stk[x] = 0;
            siz[scc]++;
            if (x == u) break;
        }
    }
}
```

割点和桥

```cpp
int dfn[N], low[N], dfs_clock;
bool iscut[N], vis[N];
void dfs(int u, int fa) {
    dfn[u] = low[u] = ++dfs_clock;
    vis[u] = 1;
    int child = 0;
    for (int v : e[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            child++;
            if (low[v] >= dfn[u]) iscut[u] = 1;
        } else if (dfn[u] > dfn[v] && v != fa) low[u] = min(low[u], dfn[v]);
        if (fa == 0 && child == 1) iscut[u] = 0;
    }
}
```

点双连通分量

```cpp
int bccno[N], bcc_cnt, siz_e[N], siz_p[N], dfs_clock, low[N], dfn[N], top;
pair<int, int> stk[N];
void dfs(int u, int fa) {
    low[u] = dfn[u] = ++dfs_clock;
    for(int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if(v == fa) continue;
        if(!dfn[v]) {
            stk[++top] = make_pair(u, v);
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) {
                bcc_cnt++;
                while(true) {
                    int x = stk[top].first, y = stk[top].second;
                    top--;
```

```
                    siz_e[bcc_cnt]++;
                    if(bccno[x] != bcc_cnt) {bccno[x] = bcc_cnt;
siz_p[bcc_cnt]++;}
                    if(bccno[y] != bcc_cnt) {bccno[y] = bcc_cnt;
siz_p[bcc_cnt]++;}
                    if(x == u && y == v) break;
                }
            }
        } else if(dfn[v] < dfn[u]) {stk[++top] = make_pair(u, v); low[u] =
min(low[u], dfn[v]);}
    }
}
```

边双连通分量

```
#include <bits/stdc++.h>
using namespace std;

const int N = 5000 + 5;
int n, m, stk[N], top, ccno, sc[N];
int dfn[N], dfc, low[N];
int mp[N][N];
int in[N];
int head[N], ecnt;
struct Edge {
    int nxt, v;
} e[N << 2];
void add_edge(int u, int v) {
    e[ecnt] = {head[u], v}; head[u] = ecnt++;
    e[ecnt] = {head[v], u}; head[v] = ecnt++;
}
void dfs(int u, int from) {
    stk[++top] = u;
    low[u] = dfn[u] = ++dfc;
    for (int i = head[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (!dfn[v]) {
            dfs(v, i);
            low[u] = min(low[u], low[v]);
        } else if ((i ^ 1) != from) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        ccno++;
        int x;
        while (true) {
            x = stk[top--];
            sc[x] = ccno;
            if (x == u) break;
        }
    }
}

void solve() {
    memset(head, -1, sizeof head);
    scanf("%d %d", &n, &m);
```

```
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        add_edge(u, v);
    }
    for (int i = 1; i <= n; i++) if (!dfn[i]) dfs(i, i);
    for (int i = 1; i <= n; i++) {
        for (int k = head[i]; k != -1; k = e[k].nxt) {
            int j = e[k].v;
            if (sc[i] != sc[j]) mp[sc[i]][sc[j]] = 1;
        }
    }

    for (int i = 1; i <= ccno; i++) {
        for (int j = 1; j <= ccno; j++) if (mp[i][j]) in[j]++;
    }
    int cnt = 0;
    for (int i = 1; i <= ccno; i++) if (in[i] == 1) cnt++;
    printf("%d\n", (cnt + 1) / 2);
}

int main() {
    int T = 1;
    while (T--) {
        solve();
    }
    return 0;
}
```

次短路计数

```
#include <iostream>
#include <cstdio>
#include <queue>
#include <cstring>
#include <algorithm>
#include <cmath>

using namespace std;

const int N = 1003, M = 10003;

int n, m, t;
int tot, head[N], ver[M], nxt[M], edge[M];
int S, T;
int dist[N][2], cnt[N][2];
bool vis[N][2];

struct Node
{
    int ver, ty, dis; //点的编号、类型（1 为次短路，0 为最短路） 和 从 1 到当前点的距离
    bool operator > (const Node &a) const
    {
        return dis > a.dis; //重载运算符
    }
```

```
} ;

inline void add(int u, int v, int w)
{
    ver[++tot] = v, edge[tot] = w, nxt[tot] = head[u], head[u] = tot;
}

inline int Dij()
{
    memset(cnt, 0, sizeof cnt);
    memset(dist, 0x3f, sizeof dist);
    memset(vis, false, sizeof vis);
    dist[S][0] = 0, cnt[S][0] = 1; //初始化时只有最短路
    priority_queue <Node, vector <Node>, greater <Node> > q;
    q.push((Node){S, 0, 0});
    while (!q.empty())
    {
        Node t = q.top(); q.pop();
        int u = t.ver, ty = t.ty, dis = t.dis, cntu = cnt[u][ty];
        if (vis[u][ty]) continue;
        vis[u][ty] = true;
        for (int i = head[u]; i; i = nxt[i])
        {
            int v = ver[i], w = edge[i];
            if (dist[v][0] == dis + w) cnt[v][0] += cntu; //与最短路长度相同
            else if (dist[v][0] > dis + w) //比最短路还短
            {
                dist[v][1] = dist[v][0], cnt[v][1] = cnt[v][0]; //先更新次短路为当前
的最短路
                q.push((Node){v, 1, dist[v][1]}); //放入堆中
                dist[v][0] = dis + w, cnt[v][0] = cntu; //更新最短路
                q.push((Node){v, 0, dist[v][0]}); //将最短路放入堆中
            }
            else if (dist[v][1] == dis + w) cnt[v][1] += cntu; //与次短路长度相同
            else if (dist[v][1] > dis + w) //比次短路短
            {
                dist[v][1] = dis + w, cnt[v][1] = cntu;
                q.push((Node){v, 1, dist[v][1]});
            }
        }
    }
    int ans = cnt[T][0];
    if (dist[T][0] + 1 == dist[T][1]) //存在比最短路长度多 1 的次短路
        ans += cnt[T][1];
    return ans;
}

int main()
{
    cin >> t;
    while (t--)
    {
        memset(head, 0, sizeof head);
        tot = 0;
        cin >> n >> m;
        for (int i = 1; i <= m; i+=1)
```

```
        {
            int u, v, w;
            cin >> u >> v >> w;
            add(u, v, w); //注意是单向边
        }
        cin >> S >> T;
        cout << Dij() << endl;
    }
    return 0;
}
```

2-SAT

$2*u$ 代表不选择，$2*u+1$ 代表选择。

```
vector<int> G[N * 2];
bool mark[N * 2];
int stk[N], top;
void build_G() {
    for (int i = 1; i <= n; i++) {
        int u, v;
        G[2 * u + 1].push_back(2 * v);
        G[2 * v + 1].push_back(2 * u);
    }
}
bool dfs(int u) {
    if (mark[u ^ 1]) return false;
    if (mark[u]) return true;
    mark[u] = 1;
    stk[++top] = u;
    for (int v : G[u]) {
        if (!dfs(v)) return false;
    }
    return true;
}
bool 2_sat() {
    for (int i = 1; i <= n; i++) {
        if (!mark[i * 2] && !mark[i * 2 + 1]) {
            top = 0;
            if (!dfs(2 * i)) {
                while (top) mark[stk[top--]] = 0;
                if (!dfs(2 * i + 1)) return 0;
            }
        }
    }
    return 1;
}
```

也可以求强连通分量。

如果对于一个*x* `sccno` 比它的反状态 *x*^1 的 `sccno` 要小，那么我们用 *x* 这个状态当做答案，否则用它的反状态当做答案。

差分约束

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int mod = 998244353;
const int N = 168;

int head[N], ecnt;
struct Edge {
    int nxt, v, w;
} e[N << 2];
void add_edge(int u, int v, int w) {
    e[++ecnt] = (Edge){head[u], v, w}, head[u] = ecnt;
}

/*
详细解释一下。
为避免负数，时间计数1～24。令：
R[i] i时间需要的人数 （1<=i<=24）
T[i] i时间应聘的人数 （1<=i<=24）
x[i] i时间录用的人数 （0<=i<=24），其中令x[0]=0
再设s[i]=x[0]+x[1]+……+x[i] （0<=i<=24），
由题意，可得如下方程组：
(1) s[i]-s[i-8]>=R[i]        (8<=i<=24)
(2) s[i]-s[16+i]>=R[i]-s[24] (1<=i<=7)
(3) s[i]-s[i-1]>=0           (1<=i<=24)
(4) s[i-1]-s[i]>=-T[i]       (1<=i<=24)

这个差分约束有个特殊的地方，(2)的右边有未知数s[24]。
这时可以通过枚举s[24]=ans来判断是否有可行解。
即(2)变形为(2') s[i]-s[16+i]>=R[i]-ans (1<=i<=7)
再通过SPFA求解(1)(2')(3)(4)。

不过最后有可能出现这种情况：
(1)(2')(3)(4)虽然有解，但求出的s[24]小于代入(2')里的ans!
这时，显然得到的s[]不满足原来的(2)了（请仔细比较(2)与(2')）。
不过虽然得到的解不满足原方程组，但这并不代表(1)(2)(3)(4)在s[24]=ans时没有可行解!
此外，值得注意的是，当得到的s[24]>ans时，虽然s[24]不一定是最优解，但把ans置成s[24]后，确实是可行解。

所以，简单把(2)置换成(2')是有问题的!
为了等价原命题，必须再加上条件：s[24]>=ans
这就是所谓加出来的那条边（5） s[24]－s[0]>=ans

最后说一下，SPFA后判dis[24]==ans其实是没有必要的。
*/
int n, R[25];
int dis[N], qcnt[N], b[N];
bool inq[N];
bool BellmanFord(int start) {
    queue<int> q;
    memset(dis, -0x3f, sizeof dis);
    memset(qcnt, 0, sizeof qcnt);
    memset(inq, 0, sizeof inq);
```

```
    q.push(start);
    dis[start] = 0;
    inq[start] = 1;
    qcnt[start]++;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = 0;
        for (int i = head[u]; i; i = e[i].nxt) {
            int v = e[i].v, w = e[i].w;
            if (dis[v] < dis[u] + w) {
                dis[v] = dis[u] + w;
                if (!inq[v]) {
                    q.push(v);
                    inq[v] = 1;
                    qcnt[v]++;
                    if (qcnt[v] > n) return 0;
                }
            }
        }
    }
    return 1;
}

void rmain() {
    /*i - 7, i
    i + (n - (8 - i) + 1)
    n - 7 + i (i < 8)*/
    memset(b, 0, sizeof b);
    for (int i = 1; i <= 24; i++) scanf("%d", &R[i]);
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        int x;
        scanf("%d", &x);
        x++;
        b[x]++;
    }
    for (int re = 0; re <= n ; re++) {
        memset(head, 0, sizeof head);
        ecnt = 0;
        for (int i = 1; i <= 24; i++) {
            if (i < 8) {
                int l = 24 - 7 + i;
                add_edge(l - 1, i, R[i] - re);
            } else {
                add_edge(i - 8, i, R[i]);
            }
            add_edge(i, i - 1, -b[i]);
            add_edge(i - 1, i, 0);
        }
        add_edge(24, 0, -re);
        bool flag = BellmanFord(0) && (re == dis[24]);
        //printf("%d %d %d\n", re, flag, dis[24]);
        if (flag) {
            printf("%d\n", re);
            return;
```

```
        }
    }
    puts("No Solution");
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        rmain();
    }
    return 0;
}
```

最小生成树

prim

```
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        G[u].push_back({v, w});
        G[v].push_back({u, w});
    }
    priority_queue<pair<int, int> > q;
    ll res = 0;
    for (int i = 2; i <= n; i++) dis[i] = 0x3f3f3f3f;
    q.push({dis[1], 1});
    while (!q.empty()) {
        auto u= q.top().second;
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        res += dis[u];

        for (auto [v, w] : G[u]) if (!vis[v]) {
            if (dis[v] > w) {
                dis[v] = w;
                q.push({-dis[v], v});
            }
        }
    }
```

字符串

哈希

```
#include <bits/stdc++.h>
using namespace std;

const int M = 233337;
```

```
int H[M], pow17[N];
void Hash(char *s, int len) {
    for (int i = 1; i <= len; i++) {
        H[i] = H[i - 1] * 17 % M + s[i] - 'a';
    }
}
int strH(int l, int r) {
    return (H[r] + M - pow17[r - l + 1] * 1ll * H[l - 1] % M) % M;
}
```

允许 k 次失配的字符串匹配

二分查找下一个失配点。

最长回文字串

二分答案。

通过哈希同样可以 $O(n)$ 解决这个问题，具体方法就是记 R_i 表示以 i 作为结尾的最长回文的长度，那么答案就是 $\max_{i=1}^{n} R_i$。考虑到 $R_i \leq R_{i-1}+2$，因此我们只需要暴力从 $R_{i-1}+2$ 开始递减，直到找到第一个回文即可。记变量 z 表示当前枚举的 R_i，初始时为 0，则 z 在每次 i 增大的时候都会增大 2，之后每次暴力循环都会减少 1，故暴力循环最多发生 $2n$ 次，总的时间复杂度为 $O(n)$。

字典树

```
int tr[N * M][26], tot;
bool End[N * M];
char s[M];

void insert(char *s, int len) {
    int u = 0;
    for (int i = 1; i <= len; i++) {
        int p = s[i] - 'a';
        if (!tr[u][p]) tr[u][p] = ++tot;
        u = tr[u][p];
    }
    End[u] = 1;
}
bool check(char *s, int len) {
    int u = 0;
    for (int i = 1; i <= len; i++) {
        int p = s[i] - 'a';
        if (!tr[u][p]) return 0;
        u = tr[u][p];
    }
    return End[u];
}
```

维护异或和

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
```

```
const int N = 526010, MX = 22;
int ch[N * MX][2], tot, rt[N], w[N * MX], xorv[N * MX], val[N];
ll ans;

void pushup(int u) {
    w[u] = xorv[u] = 0;
    if (ch[u][0]) {
        w[u] += w[ch[u][0]];
        xorv[u] ^= (xorv[ch[u][0]] << 1);
    }
    if (ch[u][1]) {
        w[u] += w[ch[u][1]];
        xorv[u] ^= (xorv[ch[u][1]] << 1) | (w[ch[u][1]] & 1);
    }
    w[u] &= 1;
}
void insert(int &o, ll ux, int dep) {
    if (!o) o = ++tot;
    if (dep > MX) return (void)(w[o]++);
    insert(ch[o][ux & 1], ux >> 1, dep + 1);
    pushup(o);
}
void addall(int o) {
    swap(ch[o][0], ch[o][1]);
    if (ch[o][0]) addall(ch[o][0]);
    pushup(o);
}
int merge(int a, int b) {
    if (!b || !a) return a + b;
    xorv[a] ^= xorv[b];
    w[a] += w[b];
    ch[a][0] = merge(ch[a][0], ch[b][0]);
    ch[a][1] = merge(ch[a][1], ch[b][1]);
    return a;
}

vector<int> G[N];
int read() {
    int w = 0, f = 1; char ch = getchar();
    while (ch > '9' || ch < '0') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        w = w * 10 + ch - 48;
        ch = getchar();
    }
    return w * f;
}

void dfs(int u) {
    for (auto v : G[u]) {
        dfs(v);
        rt[u] = merge(rt[u], rt[v]);
    }
    addall(rt[u]);
```

```
    insert(rt[u], val[u], 0);
    ans += (ll)xorv[rt[u]];
}

int main() {
    int n = read();
    for (int i = 1; i <= n; i++) val[i] = read();
    for (int i = 2; i <= n; i++) G[read()].push_back(i);
    dfs(1);
    printf("%lld\n", ans);
    return 0;
}
```

KMP

```
int n = strlen(s + 1);
for (int i = 2; i <= n; i++) {
    int j = k[i - 1];
    while (j != 0 && s[i] != s[j + 1]) j = k[j];
    if (s[i] == s[j + 1]) k[i] = j + 1;
    else k[i] = 0;
}
```

字符串最小周期

设 border 长度为 r

则 $s[i] = s[n - r + i]$

$|T| = n - r$

统计每个前缀的出现次数

1. 统计每个前缀在自身的出现次数

```
vector<int> ans(n + 1);
for (int i = 1; i <= n; i++) ans[k[i]]++;
for (int i = n; i >= 1; i--) ans[k[i]] += ans[i];
for (int i = 1; i <= n; i++) ans[i]++;
```

2. 统计每个前缀在其他串的出现次数

   我们应用来自 Knuth-Morris-Pratt 的技巧：构造一个字符串 $s + \# + t$ 并计算其前缀函数。与第一个问题唯一的不同之处在于，我们只关心与字符串 t 相关的前缀函数值，即 $i \geq n + 1$ 的 $\pi[i]$。有了这些值之后，我们可以同样应用在第一个问题中的算法来解决该问题。

一个字符串中本质不同子串的数目

给定一个长度为 n 的字符串 s，我们希望计算其本质不同子串的数目。

我们将迭代的解决该问题。换句话说，在知道了当前的本质不同子串的数目的情况下，我们要找出一种在 s 末尾添加一个字符后重新计算该数目的方法。

令 k 为当前 s 的本质不同子串数量。我们添加一个新的字符 c 至 s。显然，会有一些新的子串以字符 c 结尾。我们希望对这些以该字符结尾且我们之前未曾遇到的子串计数。

构造字符串 $t = s + c$ 并将其反转得到字符串 $t^{\sim}$。现在我们的任务变为计算有多少 $t^{\sim}$ 的前缀未在 $t^{\sim}$ 的其余任何地方出现。如果我们计算了 $t^{\sim}$ 的前缀函数最大值 $\pi_{\max}$，那么最长的出现在 s 中的前缀其长度为 $\pi_{\max}$。自然的，所有更短的前缀也出现了。

因此，当添加了一个新字符后新出现的子串数目为 $|s| + 1 - \pi_{\max}$。

所以对于每个添加的字符，我们可以在 $O(n)$ 的时间内计算新子串的数目，故最终复杂度为 $O(n^2)$。

值得注意的是，我们也可以重新计算在头部添加一个字符，或者从尾或者头移除一个字符时的本质不同子串数目。

AC 自动机

```
namespace AC {
    int ch[N][26], tot, fail[N], e[N];
    void insert(const char *s) {
        int u = 0, n = strlen(s + 1);
        for (int i = 1; i <= n; i++) {
            if (!ch[u][s[i] - 'a']) ch[u][s[i] - 'a'] = ++tot;
            u = ch[u][s[i] - 'a'];
        }
        e[u] += 1;
    }
    void build() {
        queue<int> q;
        for (int i = 0; i <= 25; i++) if (ch[0][i]) q.push(ch[0][i]);
        while (!q.empty()) {
            int now = q.front(); q.pop();
            for (int i = 0; i < 26; i++) {
                if (ch[now][i]) fail[ch[now][i]] = ch[fail[now]][i],
q.push(ch[now][i]);
                else ch[now][i] = ch[fail[now]][i];
            }
        }
    }
    int query(const char *s) {
        int u = 0, n = strlen(s + 1), res = 0;
        for (int i = 1; i <= n; i++){
            u = ch[u][s[i] - 'a'];
            for (int j = u; j && e[j] != -1; j = fail[j]) {
                res += e[j];
                e[j] = -1;
            }
        }
        return res;
    }
}
```

后缀数组

后缀数组（Suffix Array）主要是两个数组：sa 和 rk。

其中，$sa[i]$ 表示将所有后缀排序后第 i 小的后缀的编号。$rk[i]$ 表示后缀 i 的排名。

$height[i] = lcp(sa[i], sa[i-1])$，即第 i 名的后缀与它前一名的后缀的最长公共前缀。

$height[1]$ 可以视作 0。

```
const int N = 2e5 + 5;
int sa[N << 1], ork[N << 1], rk[N << 1], cnt[N], id[N << 1], M, n;
char s[N];

int main() {
    scanf("%s", s + 1);
    n = strlen(s + 1);
    for (int i = n + 1; i <= (n << 1); i++) s[i] = s[i - n], M = max(M, (int)s[i]);
    n <<= 1;
    for (int i = 1; i <= n; i++) if ((int)(s[i]) > M) M = (int)(s[i]);
    for (int i = 1; i <= n; i++) cnt[rk[i] = s[i]]++;
    for (int i = 0; i <= M; i++) cnt[i] += cnt[i - 1];
    for (int i = n; i; i--) sa[cnt[rk[i]]--] = i;
    for (int w = 1, p; w < n; w <<= 1, M = p) {
        p = 0;
        for (int i = n; i > n - w; i--) id[++p] = i;
        for (int i = 1; i <= n; i++) if (sa[i] > w) id[++p] = sa[i] - w;
        for (int i = 0; i <= M; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) cnt[rk[i]]++;
        for (int i = 1; i <= M; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i; i--) sa[cnt[rk[id[i]]]--] = id[i];
        p = 0;
        for (int i = 0; i <= n; i++) ork[i] = rk[i];
        for (int i = 1; i <= n; i++) {
            if (ork[sa[i]] == ork[sa[i - 1]] && ork[sa[i] + w] == ork[sa[i - 1] + w]) rk[sa[i]] = p;
            else rk[sa[i]] = ++p;
        }
        if (p == n) break;
    }
    for (int i = 1, k = 0; i <= n; i++) {
        if (rk[i] == 1) continue;
        if (k) k--;
        while (s[i + k] == s[sa[rk[i] - 1] + k]) k++;
        h[rk[i]] = k;
    }
    return 0;
}
```

Manacher

对于第 i 个字符为对称轴:

1. 如果回文串长为奇数, $d[2*i]/2$ 是半径加上自己的长度
2. 如果长为偶数, $d[2*i-1]/2$ 是半径的长度, 方向向右.

```cpp
int n, d[N * 2];
char s[N];

for (int i = 1; i <= n; i++) t[i * 2] = s[i], t[i * 2 - 1] = '#';
t[n * 2 + 1] = '#';
m = n * 2 + 1;
for (int i = 1, l = 0, r = 0; i <= m; i++) {
      int k = i <= r ? min(d[r - i + l], r - i + 1) : 1;
       while (i + k <= m && i - k >= 1 && t[i + k] == t[i - k]) k++;
       d[i] = k--;
       if (i + k > r) r = i + k, l = i - k;
}
```

Z 函数

$z[i] = lcp(suf_1, suf_i)$

```cpp
for (int i = 2, l = 0, r = 0; i <= n; i++) {
    if (r >= i && r - i + 1 > z[i - l + 1]) {
        z[i] = z[i - l + 1];
    } else {
        z[i] = max(0, r - i + 1);
        while (z[i] < n - i + 1 && s[z[i] + 1] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
}
```

数据结构

线段树

```cpp
#define lson (x<<1)
#define rson ((x<<1)|1)
#define mid (l + r >> 1)

const int N = 1e5 + 5;
int n, m, a[N], P;
int sum[N << 2], sumt[N << 2], mult[N << 2];

void add(int x, int l, int r, int d) {
    sumt[x] += d;
    sumt[x] %= P;
    sum[x] += d * (r - l + 1) % P;
    sum[x] %= P;
}
void mul(int x, int d) {
    sumt[x] *= d;
    sumt[x] %= P;
    mult[x] *= d;
    mult[x] %= P;
    sum[x] *= d;
    sum[x] %= P;
}
```

```
void pushdown(int x, int l, int r) {
    if (mult[x] != 1) {
        mul(lson, mult[x]);
        mul(rson, mult[x]);
        mult[x] = 1;
    }
    if (sumt[x] != 0) {
        add(lson, l, mid, sumt[x]);
        add(rson, mid + 1, r, sumt[x]);
        sumt[x] = 0;
    }
}
void pushup(int x) { sum[x] = sum[lson] + sum[rson]; }
void build(int x, int l, int r) {
    mult[x] = 1;
    if (l == r) {
        sum[x] = a[l];
        return;
    }
    build(lson, l, mid);
    build(rson, mid + 1, r);
    pushup(x);
    return;
}
void add_modify(int x, int l, int r, const int L, const int R, const int d) {
    if (l > R || r < L) return;
    if (l >= L && r <= R) {
        add(x, l, r, d);
        return;
    }
    pushdown(x, l ,r);
    add_modify(lson, l, mid, L, R, d);
    add_modify(rson, mid + 1, r, L, R, d);
    pushup(x);
    return;
}
void mul_modify(int x, int l, int r, const int L, const int R, const int d) {
    if (l > R || r < L) return;
    if (l >= L && r <= R) {
        mul(x, d);
        return;
    }
    pushdown(x, l, r);
    mul_modify(lson, l, mid, L, R, d);
    mul_modify(rson, mid + 1, r, L, R, d);
    pushup(x);
    return;
}
int query(int x, int l, int r, const int L, const int R) {
    if (r < L || l > R) return 0;
    if (l >= L && r <= R) return sum[x];
    pushdown(x, l, r);
    return (query(rson, mid + 1, r, L, R) + query(lson, l, mid, L, R)) % P;
}
```

重载加号

```
struct node {
    int len, lc, fc, rc, LC, FC, RC, lv, rv;
    void init(int x = 0) {
        lc = fc = rc = LC = RC = FC = len = 1;
        lv = rv = x;
    }
    void reverse() {
        swap(rc, LC);
        swap(lc, RC);
        swap(FC, fc);
        swap(lv, rv);
    }
    node operator + (const node &x) const {
        if (x.lc == 0) return *this;
        if (lc == 0) return x;
        node b;
        b.len = len + x.len;
        b.lv = lv;
        b.rv = x.rv;
        b.lc = (lc == len) && (x.lv > rv) ? lc + x.lc : lc;
        b.rc = (x.rc == x.len) && (x.lv > rv) ? rc + x.rc : x.rc;
        b.fc = max({fc, x.fc, (x.lv > rv ? x.lc + rc : 0)});
        b.LC = (LC == len) && (x.lv < rv) ? LC + x.LC : LC;
        b.RC = (x.RC == x.len) && (x.lv < rv) ? RC + x.RC : x.RC;
        b.FC = max({FC, x.FC, (x.lv < rv ? x.LC + RC : 0)});
        return b;
    }
};
```

可持久化线段树

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e7 + 5;
int root[N], n, m, tot, a[N];

struct node {
    int lc, rc, val;
} tr[N << 2];
void build(int& p, int l, int r) {
    p = ++tot;
    if (l == r) {
        tr[tot].val = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(tr[p].lc, l, mid);
    build(tr[p].rc, mid + 1, r);
}
void update(int &p, int l, int r, int pos, int v) {
    tr[++tot] = tr[p];
```

```
    p = tot;
    if (l == r) {
        tr[p].val = v;
        return;
    }
    int mid = (l + r) >> 1;
    if (pos <= mid) update(tr[p].lc, l, mid, pos, v);
    else update(tr[p].rc, mid + 1, r, pos, v);
}
int query(int p, int l, int r, int pos) {
    //printf("l = %d r = %d\n", l, r);
    if (l == r) return tr[p].val;
    int mid = (l + r) >> 1;
    if (pos <= mid) return query(tr[p].lc, l, mid, pos);
    else return query(tr[p].rc, mid + 1, r, pos);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    build(root[0], 1, n);
    for (int i = 1; i <= m; i++) {
        int op, vi, loc, val;
        scanf("%d %d %d", &vi, &op, &loc);
        root[i] = root[vi];
        if (op == 1) {
            scanf("%d", &val);
            update(root[i], 1, n, loc, val);
        } else {
            printf("%d\n", query(root[i], 1, n, loc));
        }
    }
    return 0;
}
```

吉老师线段树

```
int mn[N << 2], se[N << 2], tag[N << 2];
#define lson (p << 1)
#define rson ((p << 1) | 1)
#define mid ((l + r) >> 1)
void pushup(int p) {
    if (mn[lson] == mn[rson]) {
        mn[p] = mn[lson];
        se[p] = min(se[lson], se[rson]);
    } else if (mn[lson] < mn[rson]) {
        mn[p] = mn[lson];
        se[p] = min(se[lson], mn[rson]);
    } else {
        mn[p] = mn[rson];
        se[p] = min(se[rson], mn[lson]);
    }
}
void add(int p, int v) {
    if (v <= mn[p]) return;
```

```
    mn[p] = v;
    tag[p] = v;
}
void pushdown(int p) {
    if (tag[p] != -1) {
        add(lson, tag[p]);
        add(rson, tag[p]);
        tag[p] = -1;
    }
}
void build(int p, int l, int r) {
    tag[p] = -1;
    if (l == r) {
        mn[p] = 0;
        se[p] = MAX;
        return;
    }
    build(lson, l, mid);
    build(rson, mid + 1, r);
    pushup(p);
}
void modify(int p, int l, int r, int L, int R, int v) {
    if (l > R || r < L) return;
    if (mn[p] >= v) return;
    if (l >= L && r <= R && se[p] >= v) {
        add(p, v);
        return;
    }
    pushdown(p);
    modify(lson, l, mid, L, R, v);
    modify(rson, mid + 1, r, L, R, v);
    pushup(p);
}
int query(int p, int l, int r, int pos) {
    if (l == r) return mn[p];
    pushdown(p);
    if (pos <= mid) return query(lson, l, mid, pos);
    else return query(rson, mid + 1, r, pos);
}
```

静态区间第 k 小

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e7 + 5;
int n, m, a[N], root[N], tot;
vector<int> v;

struct node {
    int lc, rc, s;
} tr[N];

void build(int &p, int l, int r) {
    p = ++tot;
```

```cpp
    if (l == r) {
        tr[p].s = 1;
        return;
    }
    int mid = (l + r) >> 1;
    build(tr[p].lc, l, mid);
    build(tr[p].rc, mid + 1, r);
    tr[p].s = tr[tr[p].lc].s + tr[tr[p].rc].s;
}
void modify(int &p, int l, int r, int pos) {
    tr[++tot] = tr[p];
    p = tot;
    if (l == r) {
        tr[p].s += 1;
        return;
    }
    int mid = (l + r) >> 1;
    if (pos <= mid) modify(tr[p].lc, l, mid, pos);
    else modify(tr[p].rc, mid + 1, r, pos);
    tr[p].s = tr[tr[p].lc].s + tr[tr[p].rc].s;
}
int query(int l, int r, int L, int R, int k) {
    if (l == r) return l;
    int res = tr[tr[R].lc].s - tr[tr[L].lc].s;
    int mid = (l + r) >> 1;
    if (res >= k) return query(l, mid, tr[L].lc, tr[R].lc, k);
    else return query(mid + 1, r, tr[L].rc, tr[R].rc, k - res);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) v.push_back(a[i]);
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for (int i = 1; i <= n; i++) a[i] = lower_bound(v.begin(), v.end(), a[i]) -
v.begin() + 1;
    for (int i = 1; i <= n; i++) {
        root[i] = root[i - 1];
        modify(root[i], 1, n, a[i]);
    }
    for (int i = 1; i <= m; i++) {
        int l, r, k;
        scanf("%d %d %d", &l, &r, &k);
        printf("%d\n", v[query(1, n, root[l - 1], root[r], k) - 1]);
    }
    return 0;
}
```

树状数组

```
#define lowbit(x) (x & (-x))
int t[N], a[N], n;

void add(int x, int v) {
    for(; x <= n; x += lowbit(x)) t[x] += v;
}
int query(int x) {
    int ans = 0;
    for(; x; x -= lowbit(x)) ans += t[x];
    return ans;
}
```

```
const int N = (1 << 12) + 5;
int op, x, y, a, b, c, d, n, m;
long long tr[N][N];

void upd(int x, int y, int v) {
    for (int i = x; i <= n; i += i & -i)
        for (int j = y; j <= m; j += j & -j) tr[i][j] += v;
}
long long qry(int x, int y) {
    long long res = 0;
    for (int i = x; i; i -= i & -i)
        for (int j = y; j; j -= j & -j) res += tr[i][j];
    return res;
}
```

哈希表

```
namespace H {
    const int M = 19260817;
    int hd[M] = {}, tot = 0;
    struct E{int nxt, a, b;}e[N];
    int f(int a, int b) {return (a * 1000000000 % M + b) % M;}
    void ins(int a, int b) {int t = f(a, b); e[++tot] = E{hd[t], a, b};hd[t] =
tot;}
    int qry(int a, int b) {for(int i = hd[f(a, b)]; i; i = e[i].nxt) if(e[i].a ==
a && e[i].b == b) return i;return 0;}
}
```

splay

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

#define rank abcdefg
const int mod = 998244353;
const int N = 1e5 + 5;
```

```
int tot, fa[N], tr[N][2], sz[N], cnt[N], val[N], rt;

void maintain(int x) {
    sz[x] = sz[tr[x][0]] + sz[tr[x][1]] + cnt[x];
}
int getdir(int x) {
    return tr[fa[x]][1] == x;
}
void clear(int x) {
    fa[x] = sz[x] = cnt[x] = tr[x][0] = tr[x][1] = val[x] = 0;
}
int create(int v) {
    ++tot;
    val[tot] = v;
    sz[tot] = cnt[tot] = 1;
    return tot;
}
void rotate(int x) {
    if (x == rt) return;
    int y = fa[x], z = fa[y], d = getdir(x);
    tr[y][d] = tr[x][d ^ 1];
    if (tr[x][d ^ 1]) fa[tr[x][d ^ 1]] = y;
    fa[y] = x;
    tr[x][d ^ 1] = y;
    fa[x] = z;
    if (z) tr[z][y == tr[z][1]] = x;
    maintain(y);
    maintain(x);
}
void splay(int x) {
    for (int f = fa[x]; f = fa[x], f; rotate(x)) {
        if (fa[f]) rotate(getdir(f) == getdir(x) ? f : x);
    }
    rt = x;
}
void insert(int v) {
    if (!rt) {
        rt = create(v);
        return;
    }
    int u = rt, f = 0;
    while (true) {
        if (val[u] == v) {
            cnt[u]++;
            maintain(u);
            maintain(f);
            splay(u);
            return;
        }
        f = u, u = tr[u][v > val[u]];
        if (u == 0) {
            int id;
            fa[id = create(v)] = f;
            tr[f][v > val[f]] = id;
            maintain(f);
            splay(id);
```

```cpp
            return;
        }
    }
}

int rank(int v) {
    int rk = 0;
    int u = rt;
    while (u) {
        if (val[u] == v) {
            rk += sz[tr[u][0]];
            splay(u);
            return rk + 1;
        }
        if (v < val[u]) {
            u = tr[u][0];
        } else {
            rk += sz[tr[u][0]] + cnt[u];
            u = tr[u][1];
        }
    }
    return -1;
}

int kth(int x) {
    int u = rt;
    while (u) {
        if (sz[tr[u][0]] + cnt[u] >= x && sz[tr[u][0]] < x) return val[u];
        if (x <= sz[tr[u][0]]) {
            u = tr[u][0];
        } else {
            x -= sz[tr[u][0]] + cnt[u];
            u = tr[u][1];
        }
    }
    return u ? val[u] : -1;
}
int pre() {
    int u = tr[rt][0];
    if (!u) return val[rt];
    while (true) {
        if (tr[u][1] == 0) return splay(u), val[u];
        u = tr[u][1];
    }
    return 233;
}
int suf() {
    int u = tr[rt][1];
    if (!u) return val[rt];
    while (true) {
        if (tr[u][0] == 0) return splay(u), val[u];
        u = tr[u][0];
    }
    return 233;
}
void del(int v) {
```

```
    if (rank(v) == -1) return;
    if (cnt[rt] > 1) {
        cnt[rt]--;
        return;
    }
    if (!tr[rt][1] && !tr[rt][0]) {
        clear(rt), rt = 0;
    } else if (!tr[rt][0]) {
        int x = rt;
        rt = tr[x][1];
        fa[rt] = 0;
        clear(x);
    } else if (!tr[rt][1]) {
        int x = rt;
        rt = tr[x][0];
        fa[rt] = 0;
        clear(x);
    } else {
        int cur = rt, y = tr[cur][1];
        pre();
        tr[rt][1] = y;
        fa[y] = rt;
        clear(cur);
        maintain(rt);
    }
}



int main() {
    int n, opt, x;

    for (scanf("%d", &n); n; --n) {
        scanf("%d%d", &opt, &x);

        if (opt == 1)
            insert(x);
        else if (opt == 2)
            del(x);
        else if (opt == 3)
            printf("%d\n", rank(x));
        else if (opt == 4)
            printf("%d\n", kth(x));
        else if (opt == 5)
            insert(x), printf("%d\n", pre()), del(x);
        else
            insert(x), printf("%d\n", suf()), del(x);
    }

    return 0;
}
```

数学

基本预处理

关于 exgcd： 求解 $ax + by = gcd(a, b)$ 的一个特解。且该特解满足 $|x| \le b, |y| \le a$ 。

$gcd(x + k, y + k) = gcd(x + k, y - x)$

```
int fpow(int a, int b) {
    int res = 1;
    for (; b; b >>= 1, a = a * 1ll * a % mod) if (b & 1) res = res * 1ll * a %
mod;
    return res;
}
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b) {
        ll d = exgcd(b, a % b, y, x);
        return y -= a / b * x, d;
    } else return x = 1, y = 0, a;
}
int getinv(int v) {
    //return fpow(v, mod - 2);
    // ll x, y;
    // exgcd(v, mod, x, y);
    // return (x % mod + mod) % mod;
}
int fac[N], ifac[N];
void init_binom(int n) {
    fac[0] = ifac[0] = 1
    for (int i = 1; i <= n; i++) fac[i] = fac[i - 1] * 1ll * i % mod;
    ifac[n] = getinv(fac[n]);
    for (int i = n; i > 1; i--) ifac[i - 1] = ifac[i] * 1ll * i % mod;
}
int binom(int a, int b) {
    if (b < 0 || a < 0 || b > a) return 0;
    return fac[a] * 1ll * ifac[b] % mod * ifac[a - b] % mod;
}
int getphi(int x) {
    int res = 1;
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            x /= i;
            res *= (i - 1);
            while (x % i == 0) {
                x /= i;
                res *= i;
            }
        }
    }
    if (x > 1) res *= (x - 1);
    return res;
}
int prime[N], pcnt;
bool isp[N];
int get_prime(int n) {
```

```
    for (int i = 2; i <= n; i++) {
        if (!isp[i]) prime[++pcnt] = i;
        for (int j = 1; j <= pcnt && i * prime[j] <= n; j++) {
            isp[prime[j] * i] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}
s[0] = 1;
for (int i = 1; i <= n; ++i) s[i] = s[i - 1] * a[i] % p;
sv[n] = qpow(s[n], p - 2);
// 当然这里也可以用 exgcd 来求逆元,视个人喜好而定.
for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] * a[i] % p;
for (int i = 1; i <= n; ++i) inv[i] = sv[i] * s[i - 1] % p;
```

维护 GCD 值的种类

```
int main() {
    for (int i = 1; i <= n; i++) {
        v.push_back({i, a[i]});
        for (int j = (int)(v.size()) - 2; j >= 0; j--) {
            v[j].second = gcd(v[j].second, a[i]);
            if (v[j].second == v[j + 1].second) v.erase(v.begin() + j + 1);
        }
        mp[v[(int)(v.size()) - 1].second] += i - v[(int)(v.size()) - 1].first +
1;
        for (int j = (int)(v.size()) - 2; j >= 0; j--) {
            mp[v[j].second] += v[j + 1].first - v[j].first;
        }
    }
}
```

多项式乘法

```
#include <bits/stdc++.h>
using namespace std;

typedef complex<double> cp;
const int N = 1e6 + 7;
const double pi = acos(-1.0);
int n, m, len = 1, l, rev[N];
cp a[N], b[N];

void fft(cp *a, int n, int inv) {
    for (int i = 0; i < n; i++) if (i > rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        cp wn(cos(pi / k), inv * sin(pi / k));
        for (int i = 0; i < n; i += k * 2) {
            cp w(1, 0);
            for (int j = 0; j < k; j++, w *= wn) {
                cp x = a[i + j], y = a[i + j + k] * w;
                a[i + j] = x + y, a[i + j + k] = x - y;
            }
```

```
        }
    }
    if (inv < 0) for (int i = 0; i <= n; i++) a[i] /= n;
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i <= n; i++) scanf("%lf", &a[i]);
    for (int i = 0; i <= m; i++) scanf("%lf", &b[i]);
    while (len <= m + n) len <<= 1, l += 1;
    for (int i = 0; i < len; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (l -
1));
    fft(a, len, 1), fft(b, len, 1);
    for (int i = 0; i < len; i++) a[i] *= b[i];
    fft(a, len, -1);
    for (int i = 0; i <= n + m; i++) printf("%d ", (int)(a[i].real() + 0.5));
    return 0;
}
```

高斯消元

解线性方程组

```
void gauss() {
    for (int i = 0; i < n; i++) {
        int id = i;
        for (int j = i + 1; j < n; j++) if (fabs(a[j][i]) > fabs(a[id][i])) id =
j;
        for (int j = i; j <= n; j++) swap(a[id][j], a[i][j]);
        if (a[i][i] == 0) {
            puts("No Solution");
            return;
        }
        for (int j = 0; j < n; j++) {
            if (j == i) continue;
            double t = a[j][i] / a[i][i];
            for (int k = i; k <= n; k++) a[j][k] -= a[i][k] * t;
        }
    }
    for (int i = 0; i < n; i++) printf("%.2lf\n", a[i][n] / a[i][i]);
}
```

求行阶梯矩阵

```
void gauss() {
    int k = 1, re = 0;
    for (int i = 1; i <= m; i++) {
        if (k > n) break;
        if (a[k][i] == 0) {
            for (int j = k + 1; j <= n; j++) if (a[j][i] != 0) {
                for (int l = 1; l <= m + 1; l++) swap(a[j][l], a[k][l]);
                break;
            }
        }
```

```
        if (a[k][i] == 0) {
            re++;
        }
        for (int j = k + 1; j <= n; j++) if (a[j][i] == 1) {
            for (int l = i; l <= m + 1; l++) a[j][l] ^= a[k][l];
        }
        k++;
    }
    for (int i = k; i <= n; i++) if (a[i][m + 1] == 1) return;
}
```

线性基

```
void insert(ll x) {
    for (int i = 60; i >= 0; i--) if ((x >> i) & 1) {
        if (!a[i]) {
            a[i] = x;
            break;
        }
        x ^= a[i];
    }
}
```

欧拉函数

欧拉函数（Euler's totient function），即 $\varphi(n)$，表示的是小于等于 n 和 n 互质的数的个数。

比如说 $\varphi(1) = 1$。

当 n 是质数的时候，显然有 $\varphi(n) = n - 1$。

- 欧拉函数是积性函数。

  积性是什么意思呢？如果有 $\gcd(a, b) = 1$，那么 $\varphi(a \times b) = \varphi(a) \times \varphi(b)$。

  特别地，当 n 是奇数时 $\varphi(2n) = \varphi(n)$。
- $n = \sum_{d \mid n} \varphi(d)$。

欧拉定理

与欧拉函数紧密相关的一个定理就是欧拉定理。其描述如下：

若 $\gcd(a, m) = 1$，则 $a^{\varphi(m)} \equiv 1 \pmod{m}$。

扩展欧拉定理

当然也有扩展欧拉定理

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)}, & \gcd(a,\,p) = 1 \\ a^b, & \gcd(a,\,p) \ne 1,\, b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)}, & \gcd(a,\,p) \ne 1,\, b \ge \varphi(p) \end{cases} \pmod{p}$$

球放盒子模型

第二类斯特林数（斯特林子集数）$\begin{Bmatrix}n\\k\end{Bmatrix}$，也可记做 $S(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数。

$\begin{Bmatrix}n\\k\end{Bmatrix}=\begin{Bmatrix}n-1\\k-1\end{Bmatrix}+k\begin{Bmatrix}n-1\\k\end{Bmatrix}$

边界是 $\begin{Bmatrix}n\\0\end{Bmatrix}=[n=0]$。

假设小球个数为 n, 盒子个数为 m

1. 小球无标号，盒子有标号，不允许空盒。

   即求解方程 $\sum\limits_{i=1}^{m}x_i=n$ 解的个数

   即 $\binom{n-1}{m-1}$
2. 小球无标号，盒子有标号，允许空盒。

   令 $y_i=x_i+1$

   即求解方程 $\sum\limits_{i=1}^{m}y_i=n$ 解的个数

   即 $\binom{n+m-1}{m-1}$
3. 小球有标号，盒子有标号，允许空盒。

   即 m^n
4. 小球有标号，盒子有标号，不允许空盒。

   $m!\times\begin{Bmatrix}n\\m\end{Bmatrix}$
5. 小球有标号，盒子无标号，不允许空盒。

   $\begin{Bmatrix}n\\m\end{Bmatrix}$
6. 小球有标号，盒子无标号，允许空盒。

   $\sum\limits_{i=1}^{m}\begin{Bmatrix}n\\i\end{Bmatrix}$
7. 小球无标号，盒子无标号，允许空盒。

   设 $f[i][j]$ 表示 i 个球放入 j 个盒子的方案数。

   1. $i=0$ 或者 $j=1$, 方案数为 1
   2. $i<j$, $f[i][j]=f[i][i]$
   3. $i\geq j$, $f[i][j]=f[i-j][j]+f[i][j-1]$
8. 小球无标号，盒子无标号，不允许空盒。

   用 7 的结论，提前在每个盒子放 1 个球。

   方案数就是 $f[n-m][m]$

动态规划

树上背包上下界优化

```
for (j=min(m+1,siz[u]+siz[v]);j>=1;--j) {
    for (k=max(1,j-siz[u]);k<=siz[v]&&k<j;++k) {
        f[u][j]=max(f[u][j],f[u][j-k]+f[v][k]);
    }
}
siz[u]+=siz[v];
```

DP 优化

滑动窗口

```
while (ql <= qr && some conditions) ql++;
while (ql <= qr && some conditions) qr--;
q[++qr] = i;
```

单调队列优化背包

```
#include <bits/stdc++.h>
using namespace std;

const int N = 20005;

int n, m;
int dp[2][N], v[N], w[N], s[N], q[N];

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d %d %d", &v[i], &w[i], &s[i]);
    }
    int cur = 0;
    for (int i = 1; i <= n; i++) {
        cur ^= 1;
        for (int r = 0; r < v[i]; r++) {
            int ql = 1, qr = 0;
            for (int k = 0; k * v[i] + r <= m; k++) {
                while (ql <= qr && q[ql] < k - s[i]) ql++;
                while (ql <= qr && dp[cur ^ 1][q[qr] * v[i] + r] - q[qr] * w[i]
<= dp[cur ^ 1][k * v[i] + r] - k * w[i]) qr--;
                q[++qr] = k;
                dp[cur][k * v[i] + r] = dp[cur ^ 1][q[ql] * v[i] + r] + (k -
q[ql]) * w[i];
            }
        }
    }
    printf("%d\n", dp[cur][m]);
    return 0;
}
```

斜率优化

转移方程 $dp[i] = \min\{dp[j] + (s[i] - s[j]) - a[j+1] * (i - j)\}$

考虑斜率优化: $y = kx + b$:

1. $b = dp[i] - s[i]$
2. $k = i$
3. $x = a[j+1]$
4. $y = dp[j] - s[j] + a[j+1] * j$

那么相当于有一些点 (x, y) , 求一个点使得一条斜率为 k 的斜率过该点时截距最小.

我们假设 $slope(i)$ 表示 (x_i, y_i) 和 (x_{i+1}, y_{i+1}) 连线的斜率.

1. 如果只有一个点, 那么该点则为决策点.
2. 如果 $slope(i) \le k$ 则该点没有下一个点优.
3. 如果 $slope(i) > k$ 则该点比下一个点优.

所以需要维护一个下凸壳, 即 $slope(i)$ 单增的点集.

比较斜率的时候要注意符号.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

const int mod = 998244353;
const int N = 4e5 + 5;

int n, T;
ll dp[N], s[N], a[N];
int q[N], ql, qr;

ll y(int i) { return dp[i] - s[i] + i * a[i + 1]; }
ll x(int i) { return a[i + 1]; }

void solve() {
    for (int i = 1; i <= n; i++) cin >> a[i], dp[i] = 0;
    sort(a + 1, a + n + 1);
    for (int i = 1; i <= n; i++) s[i] = s[i - 1] + a[i];
    q[ql = qr = 1] = 0;
    for (int i = T; i <= n; i++) {
        while (ql < qr && (y(q[ql + 1]) - y(q[ql]) <= (x(q[ql + 1]) - x(q[ql])) * i)) ql++;
        //cout << i << ' ' << q[ql] << '\n';
        dp[i] = y(q[ql]) - i * x(q[ql]) + s[i];
        if (i - T + 1 < T) continue;
        while (ql < qr && (y(i - T + 1) - y(q[qr])) * (x(q[qr]) - x(q[qr - 1])) <=
            (x(i - T + 1) - x(q[qr])) * (y(q[qr]) - y(q[qr - 1]))) qr--;
        q[++qr] = i - T + 1;
    }
    //for (int i = 1; i <= n; i++) cout << i << ' ' << dp[i] << '\n';
    cout << dp[n] << '\n';
```

```
}

int main() {
    while (cin >> n >> T) {
        solve();
    }
    return 0;
}
```

当然还有一类不能把先前的点弹出的，这时候我们需要在凸包上二分。

比如，用 $s[i]$ 表示前缀和, 求 $max\{\frac{s[i]-s[j]}{i-j}\}$

*四边形不等式

其他

自定义哈希方法

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```