Ameikai 's 板子

基础

cin关流

```
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
```

快读

只能读int,long long类

```
inline ll read()
{
    ll s = 0, w = 1;
    char ch = getchar();
    while (ch > '9' || ch < '0')
    {
        if (ch == '-')
            w = -1;
        ch = getchar();
    }
    while (ch <= '9' && ch >= '0')
        s = (s << 1) + (s << 3) + (ch ^ 48), ch = getchar();
    return s * w;
}
```

数论

快速幂与快速乘法

```
long long q_mul(long long a, long long b, long long mod) //快速乘法,a*b%mod
{
    long long res = 0;
    while (b)
    {
        if (b & 1)
            res = (res + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return res;
}
long long q_pow(long long a, long long b, long long mod) //快速幂,a^b%mod
{
    long long res = 1;
```

```
    while (b)
    {
        if (b & 1)
            res = q_mul(res, a, mod);
        a = q_mul(a, a, mod);
        b >>= 1;
    }
    return res;
}
//普通写法
int qpow(int a, int b)
{
    int ans = 1;
    a %= mod;
    while (b)
    {
        if (b & 1)
            ans = (ans * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return ans;
}
```

gcd与lcm

```
int gcd(int a, int b)
{
    return b == 0? a : gcd(b, a % b);
}
int lcm(int a, int b)
{
    return a * b / gcd(a, b);
}
```

矩阵快速幂

```
inline void init()
{
    int c[105][105] = {0};
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            c[i][j] = 0;
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                c[i][j] = (c[i][j] + ans[i][k] * a[k][j]);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            ans[i][j] = c[i][j];
}
inline void init2()
{
    int c[105][105] = {0};
```

```
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            c[i][j] = 0;
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                c[i][j] = (c[i][j] + a[i][k] * a[k][j]) % mod;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            a[i][j] = c[i][j];
}
void gett()
{
    while (b)
    {
        if (b & 1)
            init();
        init2();
        b >>= 1;
    }
}
signed main()
{
    cin >> n >> b;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> a[i][j];
    for (int i = 1; i <= n; i++)
        ans[i][i] = 1;
    gett();
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cout << ans[i][j] % mod << " \n"[j == n];
    return 0;
}
```

欧拉筛

```
bool isprime[100005];
int prime[100005];
int cnt = 0;
void euler()
{
    isprime[0] = isprime[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (!isprime[i])
            prime[++cnt] = i;
        for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
        {
            isprime[i * prime[j]] = 1;
            if (i % prime[j] == 0)
                break;
        }
    }
```

```
}
```

中国剩余定理

求解同余方程组：

$x \equiv b1 \pmod{a1} x \equiv b2 \pmod{a2} \ldots x \equiv bn \pmod{an}$

$ai \leq 1012$。ai 的 lcm 不超过 1018。

```
#include <bits/stdc++.h>
using namespace std;
void exgcd(long long a, long long b, long long &x, long long &y)
{
    if (!b)
        return x = 1, y = 0, void();
    exgcd(b, a % b, y, x), y -= a / b * x;
}
long long ny(long long a, long long mod)
{
    a %= mod;
    long long x, y;
    exgcd(a, mod, x, y);
    return (x % mod + mod) % mod;
}
int n;
long long a[100011], b[100011];
long long p[111], mod[111], rem[111], idx;
void Insert(long long &x, long long y, int i)
{
    long long nmod = 1;
    while (x % p[i] == 0)
        x /= p[i], nmod *= p[i];
    if (nmod > mod[i])
        mod[i] = nmod, rem[i] = y % nmod;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i] >> b[i];
        for (int j = 1; j <= idx; j++)
            Insert(a[i], b[i], j);
        if (a[i] > 1)
        {
            for (long long np = 2; np * np <= a[i]; np++)
                if (a[i] % np == 0)
                    p[++idx] = np, Insert(a[i], b[i], idx);
            if (a[i] > 1)
                p[++idx] = a[i], Insert(a[i], b[i], idx);
        }
    }
```

```
    long long M = 1;
    for (int i = 1; i <= idx; i++)
        M *= mod[i];
    long long ans = 0;
    for (int i = 1; i <= idx; i++)
        (ans += (__int128)(M / mod[i]) * ny(M / mod[i], mod[i]) % M * rem[i] % M)
%= M;
    cout << ans;
    return 0;
}
```

搜索

DFS

```
int n, a[100005], book[100005];
void dfs(int step)
{
    int i;
    if (step == n + 1)
    {
        for (i = 1; i <= n; i++)
            cout << setw(5) << a[i];
        printf("\n");
        return;
    }
    for (int i = 1; i <= n; i++)
    {
        if (book[i] == 0)
        {
            a[step] = i;
            book[i] = 1;
            dfs(step + 1);
            book[i] = 0;
        }
    }
}
int main()
{
    scanf("%d", &n);
    dfs(1);
    return 0;
}
```

BFS

```
struct node
{
    int x, y, step;
};
int n, m;
```

```
int next1[8][2] = {{1, 2}, {2, 1}, {2, -1}, {1, -2}, {-1, -2}, {-2, -1}, {-2, 1},
{-1, 2}};
queue<node> q;
void bfs(int step)
{
    vector<vector<int>> vis(n + 1, vector<int>(m + 1, 0));
    int cnt = 1;
    while (!q.empty())
        q.pop();
    q.push({0, 0, 0});
    vis[0][0] = 1;
    node now = {0, 0, 0};
    node next;
    auto ok = [=](int x, int y) -> bool
    { return x >= 1 && x <= n && y >= 1 && y <= m && !vis[x][y]; };
    while (!q.empty())
    {
        now = q.front();
        q.pop();
        for (int i = 0; i < 8; i++)
        {
            next.x = now.x + next1[i][0];
            next.y = now.y + next1[i][1];
            next.step = now.step + 1;
            if (ok(next.x, next.y))
            {
                vis[next.x][next.y] = 1;
                q.push(next);
                cnt++;
            }
        }
    }
    cout << cnt << endl;
}
```

二分搜索

```
int binary_search1(int start, int end, int key)
{
    int ret = -1; // 未搜索到数据返回-1下标
    int mid;
    while (start <= end)
    {
        mid = start + ((end - start) >> 1); // 直接平均可能会溢出，所以用这个算法
        if (arr[mid] < key)
            start = mid + 1;
        else if (arr[mid] > key)
            end = mid - 1;
        else
        { // 最后检测相等是因为多数搜索情况不是大于就是小于
            ret = mid;
            break;
        }
    }
    return ret; // 单一出口
```

```
}
int binary_search2(vector<int> &arr, int key)
{
    return lower_bound(all(arr), key) - arr.begin();
    // 二分查找，返回下标,首个大于等于key的元素的下标
}
int binary_search3(vector<int> &arr, int key)
{
    return upper_bound(all(arr), key) - arr.begin();
    // 二分查找，返回下标,首个大于key的元素的下标
}
```

二分答案

这里是重点，一般是求最大a的最小b（反之）。

当while循环条件写的是l<r时，如果是l=mid,就需要加1，如果是r=mid就不需要。

```
// 求最大的最小值
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid))
            l = mid;
        else
            r = mid - 1;
    }
    return l;
}

// 求最小的最大值
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid))
            r = mid;
        else
            l = mid + 1;
    }
    return l;
}
```

DP

记忆化搜索

第一行有 2 个整数 T（$1 \le T \le 1000$）和 M（$1 \le M \le 100$），用一个空格隔开，T 代表总共能够用来采药的时间，M 代表山洞里的草药的数目。

接下来的 M 行每行包括两个在 1 到 100 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出在规定的时间内可以采到的草药的最大总价值。

```
int n, t;
int tcost[103], mget[103];
int mem[103][1003];

int dfs(int pos, int tleft)
{
    if (mem[pos][tleft] != -1)
        return mem[pos][tleft]; // 已经访问过的状态，直接返回之前记录的值
    if (pos == n + 1)
        return mem[pos][tleft] = 0;
    int dfs1, dfs2 = -INF;
    dfs1 = dfs(pos + 1, tleft);
    if (tleft >= tcost[pos])
        dfs2 = dfs(pos + 1, tleft - tcost[pos]) + mget[pos]; // 状态转移
    return mem[pos][tleft] = max(dfs1, dfs2);                // 最后将当前状态的值存
下来
}

int main()
{
    memset(mem, -1, sizeof(mem));
    cin >> t >> n;
    for (int i = 1; i <= n; i++)
        cin >> tcost[i] >> mget[i];
    cout << dfs(1, t) << endl;
    return 0;
}
```

背包DP

01背包

题意概要：有n个物品和一个容量为W的背包，每个物品有重量w_i和价值 v_i两种属性，要求选若干物品放入背包使背包中物品的总价值最大且背包中物品的总重量不超过背包的容量。

在上述例题中，由于每个物体只有两种可能的状态（取与不取），对应二进制中的 0 和 1，这类问题便被称为「0-1 背包问题」。

```
int main()
{
    cin >> n >> W;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i]; // 读入数据
    for (int i = 1; i <= n; i++)
        for (int j = W; j >= w[i]; j--) // 从W到小枚举背包容量
            f[j] = max(f[j], f[j - w[i]] + v[i]);
    cout << f[W];
    return 0;
}
```

完全背包

完全背包模型与 0-1 背包类似，与 0-1 背包的区别仅在于一个物品可以选取无限次，而非仅能选取一次。

我们可以借鉴 0-1 背包的思路，进行状态定义：设 f_ij 为只能选前 i 个物品时，容量为 j 的背包可以达到的最大价值。

需要注意的是，虽然定义与 0-1 背包类似，但是其状态转移方程与 0-1 背包并不相同。

```
int main()
{
    cin >> W >> n;
    for (int i = 1; i <= n; i++)
        cin >> w[i] >> v[i];
    for (int i = 1; i <= n; i++)
        for (int j = w[i]; j <= W; j++) // 从小到大枚举背包容量
            f[j] = max(f[j], f[j - w[i]] + v[i]);
    cout << f[W];
    return 0;
}
```

多重背包

多重背包也是 0-1 背包的一个变式。与 0-1 背包的区别在于每种物品有 k个，而非一个。

考虑**二进制优化**。

```
void erjz() // 二进制优化
{
    int a, b, s;
    cin >> b >> a >> s;
    int k = 1;
    while (k <= s)
    {
        cnt++;
        w[cnt] = a * k;
        v[cnt] = b * k;
        s -= k;
        k <<= 1;
    }
    if (s > 0)
    {
```

```
        cnt++;
        w[cnt] = a * s;
        v[cnt] = b * s;
    }
}
signed main()
{
    cin >> n >> W;
    for (int i = 1; i <= n; i++)
    {
        erjz();
    }
    n = cnt;
    for (int i = 1; i <= n; i++)
    {
        for (int j = W; j >= w[i]; j--)
        {
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
        }
    }
    cout << dp[W];
    return 0;
}
```

也可以**单调队列优化**。

```
constexpr int MAXV = 4e4 + 10;
constexpr int MAXN = 1e2 + 10;

using namespace std;

int n, W, last = 0, now = 1;
array<int, MAXN> v, w, k;
array<array<int, MAXV>, 2> f;
deque<int> q;

int main()
{
    ios::sync_with_stdio(false);
    cin >> n >> W;
    for (int i = 1; i <= n; i++)
        cin >> v[i] >> w[i] >> k[i];
    for (int i = 1; i <= n; i++)
    {
        for (int y = 0; y < w[i]; y++)
        {
            // 清空队列
            deque<int>().swap(q);
            for (int x = 0; x * w[i] + y <= W; x++)
            {
                // 弹出不在范围的元素
                while (!q.empty() && q.front() < x - k[i])
                {
                    q.pop_front();
                }
```

```
                // 保证队列单调
                while (!q.empty() &&
                       f[last][q.back() * w[i] + y] - q.back() * v[i] <
                           f[last][x * w[i] + y] - x * v[i])
                    q.pop_back();
                q.push_back(x);
                f[now][x * w[i] + y] =
                    f[last][q.front() * w[i] + y] - q.front() * v[i] + x * v[i];
            }
        }
        swap(last, now);
    }
    cout << f[last][W] << endl;
    return 0;
}
```

混合背包

混合背包就是将前面三种的背包问题混合起来，有的只能取一次，有的能取无限次，有的只能取 k 次。

这种题目看起来很吓人，可是只要领悟了前面几种背包的中心思想，并将其合并在一起就可以了。下面给出伪代码：

```
for (循环物品种类)
{
  if (是 0 - 1 背包)
    套用 0 - 1 背包代码;
  else if (是完全背包)
    套用完全背包代码;
  else if (是多重背包)
    套用多重背包代码;
}
```

有 n 种樱花树和长度为 T 的时间，有的樱花树只能看一遍，有的樱花树最多看 A_i 遍，有的樱花树可以看无数遍。每棵樱花树都有一个美学值 C_i，求在 T 的时间内看哪些樱花树能使美学值最高。

```
for (int i = 1; i <= n; i++)
{
    if (cnt[i] == 0)
    { // 如果数量没有限制使用完全背包的核心代码
        for (int weight = w[i]; weight <= W; weight++)
            dp[weight] = max(dp[weight], dp[weight - w[i]] + v[i]);
    }
    else
    { // 物品有限使用多重背包的核心代码，它也可以处理0-1背包问题
        for (int weight = W; weight >= w[i]; weight--)
            for (int k = 1; k * w[i] <= weight && k <= cnt[i]; k++)
                dp[weight] = max(dp[weight], dp[weight - k * w[i]] + k * v[i]);
    }
}
```

二维费用背包

有 n 个任务需要完成，完成第 i 个任务需要花费 t_i 分钟，产生c_i 元的开支。

现在有 T 分钟时间，W 元钱来处理这些任务，求最多能完成多少任务。

```
for (int k = 1; k <= n; k++)
  for (int i = m; i >= mi; i--)    // 对经费进行一层枚举
    for (int j = t; j >= ti; j--)  // 对时间进行一层枚举
      dp[i][j] = max(dp[i][j], dp[i - mi][j - ti] + 1);
```

分组背包

有 n 件物品和一个大小为 m 的背包，第 i 个物品的价值为 w_i，体积为 v_i。同时，每个物品属于一个组，同组内最多只能选择一个物品。求背包能装载物品的最大总价值。

这种题怎么想呢？其实是从「在所有物品中选择一件」变成了「从当前组中选择一件」，于是就对每一组进行一次 0-1 背包就可以了。

再说一说如何进行存储。我们可以将 t{k,i}表示第 k 组的第 i 件物品的编号是多少，再用 cnt_k 表示第 k 组物品有多少个。

```
for (int k = 1; k <= ts; k++)           // 循环每一组
  for (int i = m; i >= 0; i--)          // 循环背包容量
    for (int j = 1; j <= cnt[k]; j++)   // 循环该组的每一个物品
      if (i >= w[t[k][j]])              // 背包容量充足
        dp[i] = max(dp[i],
                    dp[i - w[t[k][j]]] + c[t[k][j]]);  // 像0-1背包一样状态转移
```

有依赖的背包

金明有 n 元钱，想要买 m 个物品，第 i 件物品的价格为 v_i，重要度为 p_i。有些物品是从属于某个主件物品的附件，要买这个物品，必须购买它的主件。

即**树形DP**。

树形DP

基础

某大学有 *n* 个职员，编号为 1...*n*。

他们之间有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。

现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数 r_i，但是呢，如果某个职员的直接上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。

求最大的快乐指数。

```
void dfs(int u, vector<int> &vis, vector<vector<int>> &dp, vector<vector<int>>
&g)
{
    vis[u] = 1;
    for (int v : g[u])
```

```
    {
        if (vis[v])
            continue;
        dfs(v, vis, dp, g);
        dp[u][1] += dp[v][0];
        dp[u][0] += max(dp[v][0], dp[v][1]);
    }
}
void solve()
{
    int n;
    cin >> n;
    vector<vector<int>> dp(n + 1, vector<int>(2));
    vector<int> du(n + 1), vis(n + 1);
    vector<vector<int>> g(n + 1);
    for (int i = 1; i <= n; i++)
        cin >> dp[i][1];
    for (int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        du[u]++;
        g[v].pb(u);
    }
    for (int i = 1; i <= n; i++)
    {
        if (du[i])
            continue;
        dfs(i, vis, dp, g);
        cout << max(dp[i][0], dp[i][1]) << endl;
        return;
    }
}
```

树上背包

现在有 n 门课程，第 i 门课程的学分为 a_i，每门课程有零门或一门先修课，有先修课的课程需要先学完其先修课，才能学习该课程。

一位学生要学习 m 门课程，求其能获得的最多学分数。

```
vc<int> g[100005];
vc<int> t(100005, 0);
int n, m;
int dp[1005][1005];
int dfs(int u)
{
    int p = 1;
    for (auto v : g[u])
    {
        int siz = dfs(v);
        for (int i = min(p, m + 1); i; i--)
            for (int j = 1; j <= siz && i + j <= m + 1; j++)
```

```
                dp[u][i + j] = max(dp[u][i + j], dp[u][i] + dp[v][j]);
        p += siz;
    }
    return p;
}
void solve()
{
    cin >> n >> m;
    ff(i, 1, n + 1)
    {
        int u;
        cin >> u >> dp[i][1];
        g[u].pb(i);
    }
    dfs(0);
    cout << dp[0][m + 1] << endl;
}
```

最长上升子序列

```
#include <vector>
#include <algorithm>
using namespace std;
int lengthOfLIS(vector<int>& nums) {
    if (nums.empty()) return 0;
    vector<int> dp(nums.size(), 1);
    int max_len = 1;
    for (int i = 0; i < nums.size(); ++i) {
        for (int j = 0; j < i; ++j) {
            if (nums[j] < nums[i]) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        max_len = max(max_len, dp[i]);
    }
    return max_len;
}

//贪心 + 二分查找：
//lower_bound 函数（需包含 <algorithm> 头文件）用于在有序的 tails 数组中查找插入位置。
//直接操作 tails 数组维护最小末尾元素，时间复杂度优化到 O(n log n)，适用于大规模数据。
int lengthOfLIS(vector<int>& nums) {
    vector<int> tails;
    for (int num : nums) {
        // 使用 lower_bound 找到第一个 >= num 的位置（类似 bisect_left）
        auto it = lower_bound(tails.begin(), tails.end(), num);
        if (it == tails.end()) {
            tails.push_back(num);
        } else {
            *it = num;
        }
    }
    return tails.size();
```

```
}
```

ST表

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
int f[100001][40], a, x, LC, n, m, p, len, l, r;
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &a);
        f[i][0] = a;
    }
    LC = (int)(log(n) / log(2));
    for (int j = 1; j <= LC; 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]);
    for (int i = 1; i <= m; i++)
    {
        scanf("%d%d", &l, &r);
        p = (int)(log(r - l + 1) / log(2));
        printf("%d\n", max(f[l][p], f[r - (1 << p) + 1][p]));
    }
    return 0;
}
```

图论

链式前向星

```
struct edge
{
    int to, w, next;
} e[100005];
int head[100005], tot = 0;
void add(int u, int v, int w)
{
    e[++tot].to = v;
    e[tot].w = w;
    e[tot].next = head[u];
    head[u] = tot;
}
//遍历
for (int i = head[u]; i; i = e[i].next)
{
    int v = e[i].to;
```

```
}
```

并查集

```
int fa[100005];
int find(int x)
{
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}
void merge(int x, int y)
{
    int fx = find(x), fy = find(y);
    if (fx != fy)
        fa[fx] = fy;
}
void init(int n)
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}
```

拓扑排序

给你一个食物网，你要求出这个食物网中最大食物链的数量。

```
const int N = 5e3 + 2; //定义常量大小
const int mod = 80112002; //定义最终答案mod的值

int n, m; //n个点 m条边
int in[N], out[N]; //每个点的入度和出度
vector<int>nei[N]; //存图，即每个点相邻的点
queue<int>q; //拓扑排序模板所需队列
int ans; //答案
int num[N]; //记录到这个点的类食物连的数量，可参考图

signed main()
{
    n = read(), m = read();
    for(rg int i = 1; i <= m; ++i)
    { //输入边
        int x = read(), y = read();
        ++in[y], ++out[x]; //右节点入度+1,左节点出度+1
        nei[x].push_back(y); //建立一条单向边
    }
    for(rg int i = 1; i <= n; ++i) //初次寻找入度为0的点(最佳生产者)
        if(!in[i])
        { //是最佳生产者
            num[i] = 1; //初始化
            q.push(i); //压入队列
        }
    while(!q.empty())
    { //只要还可以继续Topo排序
        int tot = q.front();//取出队首
        q.pop();//弹出
```

```
        int len = nei[tot].size();
        for(rg int i = 0;i < len; ++i)
        { //枚举这个点相邻的所有点
            int next = nei[tot][i]; //取出目前枚举到的点
            --in[next];//将这个点的入度-1(因为目前要删除第tot个点)
            num[next] = (num[next] + num[tot]) % mod;//更新到下一个点的路径数量
            if(in[next] == 0)q.push(nei[tot][i]);//如果这个点的入度为0了,那么压入队列
        }
    }
    for(rg int i = 1; i <= n; ++i) //寻找出度为0的点(最佳消费者)
        if(!out[i]) //符合要求
            ans = (ans + num[i]) % mod;//累加答案
    write(ans);//输出
    return 0;//end
}
```

最小生成树

Prim 算法

```
// 使用二叉堆优化的 Prim 算法。
#include <cstring>
#include <iostream>
#include <queue>
using namespace std;
constexpr int N = 5050, M = 2e5 + 10;
struct E
{
    int v, w, x;
} e[M * 2];
int n, m, h[N], cnte;
void adde(int u, int v, int w) { e[++cnte] = E{v, w, h[u]}, h[u] = cnte; }
struct S
{
    int u, d;
};
bool operator<(const S &x, const S &y) { return x.d > y.d; }
priority_queue<S> q;
int dis[N];
bool vis[N];
int res = 0, cnt = 0;
void Prim()
{
    memset(dis, 0x3f, sizeof(dis));
    dis[1] = 0;
    q.push({1, 0});
    while (!q.empty())
    {
        if (cnt >= n)
            break;
        int u = q.top().u, d = q.top().d;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = true;
```

```
        ++cnt;
        res += d;
        for (int i = h[u]; i; i = e[i].x)
        {
            int v = e[i].v, w = e[i].w;
            if (w < dis[v])
                dis[v] = w, q.push({v, w});
        }
    }
}
int main()
{
    cin >> n >> m;
    for (int i = 1, u, v, w; i <= m; ++i)
    {
        cin >> u >> v >> w, adde(u, v, w), adde(v, u, w);
    }
    Prim();
    if (cnt == n)
        cout << res;
    else
        cout << "No MST.";
    return 0;
}
```

Kruskal

```
struct Edge
{
    int u, v, w;
} edge[200005];
int fa[5005], n, m, ans, eu, ev, cnt;
il bool cmp(Edge a, Edge b)
{
    return a.w < b.w;
}
// 快排的依据（按边权排序）
il int find(int x)
{
    while (x != fa[x])
        x = fa[x] = fa[fa[x]];
    return x;
}
il void kruskal()
{
    sort(edge, edge + m, cmp);
    // 将边的权值排序
    for (re int i = 0; i < m; i++)
    {
        eu = find(edge[i].u), ev = find(edge[i].v);
        if (eu == ev)
            continue;
        // 若出现两个点已经联通了，则说明这一条边不需要了
```

```
            ans += edge[i].w;
            // 将此边权计入答案
            fa[ev] = eu;
            // 将eu、ev合并
            if (++cnt == n - 1)
                break;
            // 循环结束条件，及边数为点数减一时
        }
    }
    int main()
    {
        n = read(), m = read();
        for (re int i = 1; i <= n; i++)
            fa[i] = i;
        // 初始化并查集
        for (re int i = 0; i < m; i++)
            edge[i].u = read(), edge[i].v = read(), edge[i].w = read();
        kruskal();
        printf("%d", ans);
        return 0;
    }
```

DP 求最长（短）路

```
struct edge
{
    int v, w;
};
int n, m;
vector<edge> e[MAXN];
vector<int> L;                                  // 存储拓扑排序结果
int max_dis[MAXN], min_dis[MAXN], in[MAXN]; // in 存储每个节点的入度
void toposort()
{ // 拓扑排序
    queue<int> S;
    memset(in, 0, sizeof(in));
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j < e[i].size(); j++)
            in[e[i][j].v]++;
    }
    for (int i = 1; i <= n; i++)
        if (in[i] == 0)
            S.push(i);
    while (!S.empty())
    {
        int u = S.front();
        S.pop();
        L.push_back(u);
        for (int i = 0; i < e[u].size(); i++)
        {
            if (--in[e[u][i].v] == 0)
                S.push(e[u][i].v);
```

```
        }
    }
}
void dp(int s)
{               // 以 s 为起点求单源最长（短）路
    toposort(); // 先进行拓扑排序
    memset(min_dis, 0x3f, sizeof(min_dis));
    memset(max_dis, 0, sizeof(max_dis));
    min_dis[s] = 0;
    for (int i = 0; i < L.size(); i++)
    {
        int u = L[i];
        for (int j = 0; j < e[u].size(); j++)
        {
            min_dis[e[u][j].v] = min(min_dis[e[u][j].v], min_dis[u] + e[u][j].w);
            max_dis[e[u][j].v] = max(max_dis[e[u][j].v], max_dis[u] + e[u][j].w);
        }
    }
}
```

Dijkstra

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii; // 存储 {距离, 节点}
void dijkstra(vector<vector<pii>> &graph, int start, vector<int> &dist)
{
    int n = graph.size();
    dist.assign(n, INT_MAX);
    dist[start] = 0;
    // 最小堆优先队列，按距离从小到大排序
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, start});
    while (!pq.empty())
    {
        int u = pq.top().second;
        int d = pq.top().first;
        pq.pop();
        // 如果当前距离大于已记录的最短距离，跳过
        if (d > dist[u])
            continue;
        // 遍历所有邻接节点
        for (auto &edge : graph[u])
        {
            int v = edge.first;
            int w = edge.second;
            // 松弛操作（Relaxation）
            if (dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
```

```
        }
    }
}
int main()
{
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    vector<vector<pii>> graph(n);
    for (int i = 0; i < m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
        graph[v].push_back({u, w});
    }
    vector<int> dist;
    dijkstra(graph, s, dist);
    cout << dist[t] << endl;
    return 0;
}
```

树状数组

单点修改，区域查询

```
ll n, m;
ll bit[1000005];
ll arr[1000005];
ll lowbit(ll x) { return x & -x; }
void add(ll x, ll y)
{
    for (int i = x; i <= n; i += lowbit(i))
        bit[i] += y;
}
ll ask(ll l, ll r)
{
    ll sum = 0;
    for (int i = l - 1; i >= 1; i -= lowbit(i))
        sum -= bit[i];
    for (int i = r; i >= 1; i -= lowbit(i))
        sum += bit[i];
    return sum;
}
void solve()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> arr[i];
        add(i, arr[i]);
    }
    while (m--)
    {
```

```
        int op;
        cin >> op;
        if (op == 1)
        {
            ll a, b;
            cin >> a >> b;
            add(a, b);
        }
        else if (op == 2)
        {
            ll a, b;
            cin >> a >> b;
            cout << ask(a, b) << endl;
        }
    }
}
```

区域修改，单点查询

```
ll n, m;
ll arr[1000010];
ll tree[1000010];
ll lowbit(ll x) { return x & -x; }
void add(ll x, ll val)
{
    while (x <= n)
    {
        tree[x] += val;
        x += lowbit(x);
    }
}
ll ask(ll x)
{
    ll sum = 0;
    while (x > 0)
    {
        sum += tree[x];
        x -= lowbit(x);
    }
    return sum;
}
void solve()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> arr[i];
        add(i, arr[i] - arr[i - 1]);
    }
    while (m--)
    {
        ll op;
        cin >> op;
```

```
        if (op == 1)
        {
            ll l, r, val;
            cin >> l >> r >> val;
            add(l, val);
            add(r + 1, -val);
        }
        else
        {
            ll x;
            cin >> x;
            cout << ask(x) << endl;
        }
    }
}
```

线段树

```
// 区域操作都用if,单点操作用if else
struct treeNode
{
    int l, r;       // 都有
    int sum, maxx; // 单点修改,区间查询
    int num;        // 区域修改,单点查询
    int lazy;
};
treeNode tree[2000005];
int a[500005];

void build(int i, int l, int r)
{
    tree[i].l = l;
    tree[i].r = r;
    tree[i].num = 0;
    if (l == r)
    {
        tree[i].sum = a[l];
        tree[i].maxx = a[l];
        tree[i].num = a[l];
        return;
    }
    int mid = l + r >> 1;
    build(i << 1, l, mid);
    build(i << 1 | 1, mid + 1, r);
    tree[i].sum = tree[i << 1].sum + tree[i << 1 | 1].sum;
    tree[i].maxx = max(tree[i << 1].maxx, tree[i << 1 | 1].maxx);
}

int search_range_sum(int i, int l, int r)
{
    if (tree[i].l >= l && tree[i].r <= r)
        return tree[i].sum;
    int mid = tree[i].l + tree[i].r >> 1;
```

```
    int ans = 0;
    if (tree[i].r >= l)
        ans += search_range_sum(i << 1, l, r);
    if (tree[i].l <= r)
        ans += search_range_sum(i << 1 | 1, l, r);
    return ans;
}

int search_range_max(int i, int l, int r)
{
    if (tree[i].l >= l && tree[i].r <= r)
        return tree[i].maxx;
    int mid = tree[i].l + tree[i].r >> 1;
    int ans = -INF;
    if (tree[i].r >= l)
        ans = max(ans, search_range_max(i << 1, l, r));
    if (tree[i].l <= r)
        ans = max(ans, search_range_max(i << 1 | 1, l, r));
    return ans;
}

void update_point(int i, int x, int v)
{
    if (tree[i].l == tree[i].r)
    {
        tree[i].sum += v;
        tree[i].maxx += v;
        return;
    }
    if (x <= tree[i << 1].r)
        update_point(i << 1, x, v);
    else
        update_point(i << 1 | 1, x, v);
    tree[i].sum = tree[i << 1].sum + tree[i << 1 | 1].sum;
    tree[i].maxx = max(tree[i << 1].maxx, tree[i << 1 | 1].maxx);
}

void update_range(int i, int l, int r, int v)
{
    if (tree[i].l >= l && tree[i].r <= r)
    {
        tree[i].num += v;
        return;
    }
    int mid = tree[i].l + tree[i].r >> 1;
    if (l <= mid)
        update_range(i << 1, l, r, v);
    if (r > mid)
        update_range(i << 1 | 1, l, r, v);
}

int s = 0;
void search_point(int i, int x)
{
    s += tree[i].num;
    if (tree[i].l == tree[i].r)
```

```
        return;
    int mid = tree[i].l + tree[i].r >> 1;
    if (x <= mid)
        search_point(i << 1, x);
    else
        search_point(i << 1 | 1, x);
}

void push_down(int i)
{
    if (tree[i].lazy)
    {
        tree[i << 1].lazy += tree[i].lazy;
        tree[i << 1 | 1].lazy += tree[i].lazy;
        int mid = tree[i].l + tree[i].r >> 1;
        tree[i << 1].sum += tree[i].lazy * (mid - tree[i << 1].l + 1);
        tree[i << 1 | 1].sum += tree[i].lazy * (tree[i << 1 | 1].r - mid);
        tree[i << 1].maxx += tree[i].lazy;
        tree[i << 1 | 1].maxx += tree[i].lazy;
        tree[i].lazy = 0;
    }
}

void update_range_lazy(int i, int l, int r, int k)
{
    if (tree[i].l >= l && tree[i].r <= r)
    {
        tree[i].sum += k * (tree[i].r - tree[i].l + 1);
        tree[i].maxx += k;
        tree[i].lazy += k;
        return;
    }
    push_down(i);
    if (tree[i << 1].r >= l)
        update_range_lazy(i << 1, l, r, k);
    if (tree[i << 1 | 1].l <= r)
        update_range_lazy(i << 1 | 1, l, r, k);
    tree[i].sum = tree[i << 1].sum + tree[i << 1 | 1].sum;
    tree[i].maxx = max(tree[i << 1].maxx, tree[i << 1 | 1].maxx);
}

int search_range_sum_lazy(int i, int l, int r)
{
    if (tree[i].l >= l && tree[i].r <= r)
        return tree[i].sum;
    if (tree[i].r < l || tree[i].l > r)
        return 0;
    push_down(i);
    int ans = 0;
    if (tree[i << 1].r >= l)
        ans += search_range_sum_lazy(i << 1, l, r);
    if (tree[i << 1 | 1].l <= r)
        ans += search_range_sum_lazy(i << 1 | 1, l, r);
    return ans;
}
```

```
int search_range_max_lazy(int i, int l, int r)
{
    if (tree[i].l >= l && tree[i].r <= r)
        return tree[i].maxx;
    if (tree[i].r < l || tree[i].l > r)
        return -INF;
    push_down(i);
    int ans = -INF;
    if (tree[i << 1].r >= l)
        ans = max(ans, search_range_max_lazy(i << 1, l, r));
    if (tree[i << 1 | 1].l <= r)
        ans = max(ans, search_range_max_lazy(i << 1 | 1, l, r));
    return ans;
}

void solve()
{
    int n, m, k;
    cin >> n >> m >> k;
    ff(i, 1, n + 1)
    {
        cin >> a[i];
    }
    build(1, 1, n);
    ff(i, 1, m)
    {
        int op, l, r, x;
        cin >> op >> l >> r;
        if (op == 1)
        {
            cin >> x;
            update_range_lazy(1, l, r, x);
        }
        else if (op == 2)
        {
            cout << search_range_sum_lazy(1, l, r) << endl;
        }
        else if (op == 3)
        {
            cout << search_range_max_lazy(1, l, r) << endl;
        }
        else if (op == 4)
        {
            search_point(1, l);
            cout << s << endl;
            s = 0;
        }
    }
}
```

字符串

KMP

```
// 计算部分匹配表（next数组）
void getNext(const string &pattern, vector<int> &next)
{
    int m = pattern.size();
    next.resize(m);
    int j = 0;
    for (int i = 1; i < m; ++i)
    {
        while (j > 0 && pattern[i] != pattern[j])
            j = next[j - 1];
        if (pattern[i] == pattern[j])
            ++j;
        next[i] = j;
    }
}

// KMP算法搜索匹配串出现的所有位置
vector<int> kmpSearch(const string &text, const string &pattern)
{
    int n = text.size();
    int m = pattern.size();
    vector<int> next;
    getNext(pattern, next);
    vector<int> positions;
    int j = 0;
    for (int i = 0; i < n; ++i)
    {
        while (j > 0 && text[i] != pattern[j])
            j = next[j - 1];
        if (text[i] == pattern[j])
            ++j;
        if (j == m)
        {
            positions.push_back(i - m + 1);
            j = next[j - 1];
        }
    }
    return positions;
}
```

字典树

```
struct trie
{
    int nex[100000][26], cnt;
    bool exist[100000]; // 该结点结尾的字符串是否存在

    void insert(char *s, int l)
    { // 插入字符串
        int p = 0;
```

```
        for (int i = 0; i < l; i++)
        {
            int c = s[i] - 'a';
            if (!nex[p][c])
                nex[p][c] = ++cnt; // 如果没有，就添加结点
            p = nex[p][c];
        }
        exist[p] = true;
    }

    bool find(char *s, int l)
    { // 查找字符串
        int p = 0;
        for (int i = 0; i < l; i++)
        {
            int c = s[i] - 'a';
            if (!nex[p][c])
                return 0;
            p = nex[p][c];
        }
        return exist[p];
    }
};
```

常用工具

离散化

```
// 离散化主函数：返回排序去重后的数组
vector<int> discretize(vector<int> &nums)
{
    vector<int> sorted = nums;
    sort(sorted.begin(), sorted.end());                                  // 1. 排序
    sorted.erase(unique(sorted.begin(), sorted.end()), sorted.end()); // 2. 去重
    return sorted;
}
// 查询数值 x 的离散化后索引（从 0 开始）
int get_index(vector<int> &sorted, int x)
{
    return lower_bound(sorted.begin(), sorted.end(), x) - sorted.begin(); // 3.
二分查找
}
// 示例用法
int main()
{
    vector<int> nums = {3, 1, 100, 2, 1, 200, 3}; // 原始数据
    vector<int> sorted = discretize(nums);        // 离散化数组：[1,2,3,100,200]

    int idx_3 = get_index(sorted, 3);     // 返回索引 2
    int idx_200 = get_index(sorted, 200); // 返回索引 4
    return 0;
}
```

哈希表

```
const int SZ = 1000003; // 定义哈希表大小
struct hash_map {  // 哈希表模板

  struct data {
    long long u;
    int v, nex;
  };  // 前向星结构

  data e[SZ << 1];  // SZ 是 const int 表示大小
  int h[SZ], cnt;

  int hash(long long u) { return (u % SZ + SZ) % SZ; }

  // 这里使用 (u % SZ + SZ) % SZ 而非 u % SZ 的原因是
  // C++ 中的 % 运算无法将负数转为正数

  int& operator[](long long u) {
    int hu = hash(u);  // 获取头指针
    for (int i = h[hu]; i; i = e[i].nex)
      if (e[i].u == u) return e[i].v;
    return e[++cnt] = data{u, -1, h[hu]}, h[hu] = cnt, e[cnt].v;
  }

  hash_map() {
    cnt = 0;
    memset(h, 0, sizeof(h));
  }
};
int main()
{
    hash_map mp;
    mp[123] = 10;
    mp[456] = 20;
    mp[-789] = 30; // 支持负键
    // 更新数据
    mp[123] = 100;
    // 查找数据
    cout << "键 123 的值: " << mp[123] << endl; // 输出 100
    cout << "键 999 的值: " << mp[999] << endl; // 输出 -1（未找到）
    // 遍历所有键值对
    cout << "\n所有键值对: " << endl;
    for (int i = 1; i <= mp.cnt; ++i)
    {
        cout << "键: " << mp.e[i].u << ", 值: " << mp.e[i].v << endl;
    }
}
```

单调栈

```
int ans[3000005];
int a[3000005];
void solve()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    stack<int> st;
    for (int i = n; i >= 1; i--)
    {
        while (!st.empty() && a[st.top()] <= a[i])
            st.pop();
        ans[i] = st.empty() ? 0 : st.top();
        st.push(i);
    }
    for (int i = 1; i <= n; i++)
        cout << ans[i] << " ";
    cout << endl;
}
```

单调队列

有一个长为 *n* 的序列 *a*，以及一个大小为 *k* 的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值和最小值。

```
int n, a;
int arr[1000005];
int ans[1000005][2];
deque<pair<int, int>> q1, q2;
void solve()
{
    cin >> n >> a;
    for (int i = 1; i <= n; i++)
    {
        int x;
        cin >> x;
        while (!q1.empty() && q1.back().fi <= x)
            q1.pop_back();
        while (!q2.empty() && q2.back().fi >= x)
            q2.pop_back();
        q1.push_back({x, i});
        q2.push_back({x, i});
        while (i - a >= q1.front().se)
            q1.pop_front();
        while (i - a >= q2.front().se)
            q2.pop_front();
        if (i >= a)
        {
            ans[i - a + 1][0] = q1.front().fi;
```

```
            ans[i - a + 1][1] = q2.front().fi;
        }
    }
    for (int i = 1; i <= n - a + 1; i++)
        cout << ans[i][1] << " ";
    cout << endl;
    for (int i = 1; i <= n - a + 1; i++)
        cout << ans[i][0] << " ";
    cout << endl;
}
```

__int128

```

```

常用函数

C++头文件汇总

```

```