

算法汇总

算法

建图

三种建图方式

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 11; // 最大点数
4 const int MAXM = 21; // 最大边数 (无向图需 2 倍)
5 // 邻接矩阵
6 int graph1[MAXN][MAXN];
7 // 邻接表
8 vector<vector<pair<int, int>>> graph2;
9 // 链式前向星
10 int head[MAXN], nxt[MAXM], to[MAXM], weight[MAXM], cnt;
11 void build(int n) {
12     // 清空邻接矩阵
13     for (int i = 1; i <= n; ++i)
14         for (int j = 1; j <= n; ++j)
15             graph1[i][j] = 0;
16     // 清空邻接表
17     graph2.clear();
18     graph2.resize(n + 1);
19     // 清空链式前向星
20     cnt = 0;
21     memset(head, 0, sizeof(head));
22 }
23 void addEdge(int u, int v, int w) {
24     nxt[++cnt] = head[u];
25     to[cnt] = v;
26     weight[cnt] = w;
27     head[u] = cnt;
28 }
29 //有向
30 void directGraph(const vector<vector<int>>& edges) {
31     for (const auto& edge : edges) {
32         graph1[edge[0]][edge[1]] = edge[2];
33         graph2[edge[0]].emplace_back(edge[1], edge[2]);
34         addEdge(edge[0], edge[1], edge[2]);
35     }
36 }
37 // 无向
```

```

38 void undirectGraph(const vector<vector<int>>& edges) {
39     for (const auto& edge : edges) {
40         graph1[edge[0]][edge[1]] = edge[2];
41         graph1[edge[1]][edge[0]] = edge[2];
42
43         graph2[edge[0]].emplace_back(edge[1], edge[2]);
44         graph2[edge[1]].emplace_back(edge[0], edge[2]);
45
46         addEdge(edge[0], edge[1], edge[2]);
47         addEdge(edge[1], edge[0], edge[2]);
48     }
49 }
50 void traversal(int n) {
51     cout << "邻接矩阵遍历:" << endl;
52     for (int i = 1; i <= n; ++i) {
53         for (int j = 1; j <= n; ++j) {
54             cout << graph1[i][j] << " ";
55         }
56         cout << endl;
57     }
58     cout << "邻接表遍历:" << endl;
59     for (int i = 1; i <= n; ++i) {
60         cout << i << " (邻居,边权): ";
61         for (auto& edge : graph2[i]) {
62             cout << "(" << edge.first << "," << edge.second << ") ";
63         }
64         cout << endl;
65     }
66     cout << "链式前向星遍历:" << endl;
67     for (int i = 1; i <= n; ++i) {
68         cout << i << " (邻居,边权): ";
69         for (int ei = head[i]; ei > 0; ei = nxt[ei]) {
70             cout << "(" << to[ei] << "," << weight[ei] << ") ";
71         }
72         cout << endl;
73     }
74 }
75 int main() {
76     int n1 = 4;
77     vector<vector<int>> edges1 = {{1, 3, 6}, {4, 3, 4}, {2, 4, 2}, {1, 2, 7},
78     {2, 3, 5}, {3, 1, 1}};
79     build(n1);
80     directGraph(edges1);
81     traversal(n1);
82     cout << "======" << endl;
83     int n2 = 5;
84     vector<vector<int>> edges2 = {{3, 5, 4}, {4, 1, 1}, {3, 4, 2}, {5, 2, 4},
85     {2, 3, 7}, {1, 5, 5}, {4, 2, 6}};
86     build(n2);
87     undirectGraph(edges2);
88     traversal(n2);
89     return 0;
90 }
```

欧拉回路与欧拉路径

七桥问题源自柯尼斯堡城（现加里宁格勒）的七座桥，询问能否从某处出发，恰好一次走过每座桥并回到起点（或不回到起点）。

度（degree）：无向图中与顶点相 incident 的边条数，记作 $\deg(v)$ 。

欧拉路径（Euler trail/path）：经过每条边恰好一次的路。

欧拉回路（Euler circuit/tour）：经过每条边恰好一次并回到起点的环。

连通性：忽略度为 0 的顶点后，图在无向意义下应当连通。

判定定理（必要且充分）

无向图（允许多重边）：

1. 非零度顶点诱导的子图连通；

2. 设奇度顶点个数为 k ：

- $k = 0 \Rightarrow$ 存在欧拉回路（也存在路径）；
- $k = 2 \Rightarrow$ 存在欧拉路径但无回路（起点/终点为这两个奇度顶点之一）；
- 其他 $k \Rightarrow$ 不存在欧拉路径。

直观理解：进入与离开同一顶点消耗 2 条 incident 边，所以除起点/终点外应为偶度；若回路（起点=终点），则所有顶点偶度。

图最短路

从某一基点到其他点的最短路径

Dijkstra 算法

普通堆

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int n, m, st;
5     cin >> n >> m >> st;
6     vector<int> distance(n + 1, INT_MAX);
7     vector<int> visited(n + 1, false);
8     vector<vector<pair<int,int>>> g(n + 1);
9     for(int i = 0, u, v, w; i < m; i ++){
10         cin >> u >> v >> w;
11         g[u].push_back({v, w});
12     }
13     priority_queue<pair<int,int>,vector<pair<int,int>>,greater<>> q1;
```

```

14     for(auto it : g[st]){
15         q1.push({it.second,it.first});
16     }
17     distance[st] = 0;
18     pair<int,int> temp;
19     int u, w, v;
20     while(!q1.empty()){
21         temp = q1.top();
22         q1.pop();
23         u = temp.second;
24         w = temp.first;
25         if(visited[u]) continue;
26         visited[u] = true;
27         distance[u] = w;
28         for(auto it : g[u]){
29             v = it.first;
30             if(!visited[v] && it.second + distance[u] < distance[v]){
31                 q1.push({it.second + distance[u], v});
32             }
33         }
34     }
35     for(int i = 1; i <= n; i ++){
36         cout << distance[i] << " ";
37     }
38     return 0;
39 }
```

反向索引堆

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 //heap
4 int MAXN = 100001;
5 int MAXM = 200001;
6 vector<int> head(MAXN);
7 vector<int> next1(MAXM);
8 vector<int> to(MAXM);
9 vector<int> weight(MAXM);
10 int cnt = 1;
11 vector<int> heap(MAXN);
12 int sizeh = 0;
13 vector<int> where(MAXN, -1);
14 vector<int> distancel(MAXN, INT_MAX);
15 void swap1(int i, int j) {
16     swap(heap[i], heap[j]);
17     where[heap[i]] = i;
18     where[heap[j]] = j;
19 }
20 void heapInsert(int i) {
21     while (i > 0 && distancel[heap[i]] < distancel[heap[(i - 1) / 2]]){
22         swap1(i, (i - 1) / 2);
23         i = (i - 1) / 2;
24     }
25 }
26 void heapify(int i) {
27     int l = i * 2 + 1;
```

```

28     int max_mark;
29     int mark;
30     while (l < sizeh) {
31         max_mark = l + 1 < sizeh && distance1[heap[l + 1]] < distance1[heap[l]]
? l + 1 : l;
32         mark =distance1[ heap[i]] <distance1[ heap[max_mark]] ? i : max_mark;
33         if (mark == i) break;
34         else {
35             swap1(i, mark);
36             i = mark;
37             l = i * 2 + 1;
38         }
39     }
40 }
41 void solve2() {
42     int n, m, s;
43     cin >> n >> m >> s;
44     for (int i = 0, x, y, w; i < m; i++) {
45         cin >> x >> y >> w;
46         next1[cnt] = head[x];
47         to[cnt] = y;
48         weight[cnt] = w;
49         head[x] = cnt++;
50     }
51     heap[sizeh++] = s;
52     distance1[s] = 0;
53     while (sizeh) {
54         int u = heap[0];
55         swap1(0, --sizeh);
56         heapify(0);
57         where[u] = -2;
58         for (int ei = head[u]; ei > 0; ei = next1[ei]) {
59             int v = to[ei];
60             int w = weight[ei];
61             if (where[v] == -1) {
62                 distance1[v] = w + distance1[u];
63                 where[v] = sizeh;
64                 heap[sizeh++] = v;
65                 heapInsert(where[v]);
66             } else if (where[v] >= 0){
67                 distance1[v] = min(distance1[v], distance1[u] + w);
68                 heapInsert(where[v]);
69             }
70         }
71     }
72     for (int i = 1; i <= n; i++) {
73         cout << distance1[i];
74         if (i != n) cout << " ";
75     }
76     cout << endl;
77 }
```

求最短奇偶路径

该思想可拓展为某一个节点有n种状态，故每条边增加一个变量为路的数量，一般设置为t = 1,而跑DJ算法时，可以在优先队列后面再加一个属性表示状态type， $type = (type + t) \% n$ 。

当再增加一个属性表示节点的类别，相同节点间可以以某个代价相互传送时，则可以抽象为新增了一个祖先节点，该祖先与所有该类别的节点相互通连，节点到祖先的t=1，祖先到节点的t=0。

```
1 // 假设边权为1, 若不为1则w+1变成w+weight[ei]
2 // 初始化奇数和偶数路径长度的距离数组
3 vector<int> dis_o(n + 1, 1e18), dis_e(n + 1, 1e18); // dis_o: 最短奇数路径, dis_e: 最短偶数路径
4 dis_e[1] = 0; // 起点(节点1)的偶数路径长度为0
5 // 使用最小堆优先队列存储{距离, 节点}对
6 priority_queue<pair<int,int>, vector<pair<int,int>>, greater<>> q1;
7 q1.push({0, 1}); // 从节点1开始, 距离为0
8 // Dijkstra算法
9 while (!q1.empty()) {
10     auto [w, u] = q1.top(); // w: 当前距离, u: 当前节点
11     q1.pop();
12     // 遍历节点u的所有邻居
13     for (int ei = head[u]; ei; ei = next[ei]) {
14         int v = to[ei]; // 邻居节点
15         if (w & 1) { // 如果到u的路径长度为奇数
16             if (dis_e[v] > w + 1) { // 尝试更新v的偶数路径
17                 dis_e[v] = w + 1;
18                 q1.push({w + 1, v});
19             }
20         } else { // 如果到u的路径长度为偶数
21             if (dis_o[v] > w + 1) { // 尝试更新v的奇数路径
22                 dis_o[v] = w + 1;
23                 q1.push({w + 1, v});
24             }
25         }
26     }
27 }
```

```
1 /*迷宫有n个房间，通过m条单向道路连接，迷宫的起点是1号房间，终点是n号房间。每条道路都有一个守路人，需要支付对应数量的金币才能通过，这个迷宫还存在着两个神秘的规则：
2 1.只有当走到终点时经过的路的数量是3的倍数时她才能顺利逃离迷宫。
3 2.每个房间都有一个类别属性，她可以花费x个金币在相同类别的属性之间瞬移（注意每次瞬移也算作走过了
一条路），特殊地她可以在同个房间内进行瞬移
4 如果怎样都不能走出迷宫的话输出"-1"*/
5 #include<bits/stdc++.h>
6 using namespace std;
7 #define int long long
8 #define fr first
9 #define sc second
10 constexpr int inf = 1e18;
11 signed main(){
12     cin.tie(0)->iostream::sync_with_stdio(0);
13     int t = 1;
14     cin >> t;
15     while(t --){
16         int n, m, k;
```

```

17     cin >> n >> m >> k;
18     vector<vector<int>> adj(n << 1 | 1, vector<array<int, 3>>());
19     vector<int> sta(n + 1);
20     for(int i = 1; i <= n; i ++){
21         cin >> sta[i];
22         adj[i].push_back({sta[i] + n, k, 1});
23         adj[sta[i] + n].push_back({i, 0, 0});
24     }
25     for(int i = 1, u, v, w; i <= m; i ++){
26         cin >> u >> v >> w;
27         adj[u].push_back({v, w, 1});
28     }
29     vector<array<int, 3>> dis(n << 1 | 1, {inf, inf, inf});
30     priority_queue<array<int, 3>, vector<array<int, 3>>, greater<> q1;
31     array<int, 3> t1;
32     int u, val, in, in1;
33     dis[1][0] = 0;
34     q1.push({0, 1, 0});
35     while(!q1.empty()){
36         t1 = q1.top();
37         q1.pop();
38         val = t1[0], u = t1[1], in = t1[2];
39         for(auto [v, w, gap] : adj[u]){
40             in1 = (in + gap) % 3;
41             if(dis[v][in1] > val + w){
42                 dis[v][in1] = val + w;
43                 q1.push({dis[v][in1], v, in1});
44             }
45         }
46     }
47     if(dis[n][0] == inf){
48         cout << "-1\n";
49     }else{
50         cout << dis[n][0] << "\n";
51     }
52 }
53 return 0;
54 }
```

Floyd算法

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int INF = numeric_limits<int>::max();
4 void floydWarshallTemplate() {
5     int n, m;
6     cin >> n >> m;
7     // 初始化邻接矩阵，默认不可达时权值为 INF
8     vector<vector<int>> dist(n, vector<int>(n, INF));
9     // 自己到自己距离为 0
10    for (int i = 0; i < n; i++) {
11        dist[i][i] = 0;
12    }
13    // 读入边，假设输入为无向图，若为有向图则只更新一边
14    for (int i = 0; i < m; i++) {
15        int u, v, w;
```

```

16     cin >> u >> v >> w;
17     u--; v--; // 若输入顶点编号从 1 开始则转换为从 0 开始
18     // 对于多重边取权值最小的一条
19     dist[u][v] = min(dist[u][v], w);
20     dist[v][u] = min(dist[v][u], w);
21 }
22 // Floyd-Warshall 算法：枚举中转点 k、起点 i 和终点 j
23 for (int k = 0; k < n; k++) {
24     for (int i = 0; i < n; i++) {
25         // 如果 i->k 不可达则无需更新
26         if (dist[i][k] == INF) continue;
27         for (int j = 0; j < n; j++) {
28             // 若路径 i->k 和 k->j 均可达，更新 i->j 的最短路径
29             if (dist[k][j] < INF)
30                 dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
31         }
32     }
33 }
34 // 输出结果：若两点不可达则输出 0，否则输出最短路径长度
35 for (int i = 0; i < n; i++) {
36     for (int j = 0; j < n; j++) {
37         cout << (dist[i][j] == INF ? 0 : dist[i][j]) << " ";
38     }
39     cout << "\n";
40 }
41 }

```

SPFA (Shortest Path Faster Algorithm) 算法

SPFA 算法是对 Bellman-Ford 算法的一种改进，主要用于在含有负权边的图中求最短路径。它利用队列来维护“待更新”的节点，从而提高更新效率。

根据最短路径理论，在没有负权回路的图中，从起点到任一节点的最短路径最多只需要经过 $n-1$ 条边。所以若某节点的路径更新次数超过 $n-1$ ，就能确定有负权回路存在。

`visited`：标记当前节点是否在队列中，防止重复入队

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 #define pb push_back
5 #define sc second
6 #define fr first
7 const int M = 6001;
8 vector<int> head(M/2);
9 vector<int> next1(M);
10 vector<int> to(M);
11 vector<int> weight(M);
12 int cnt;
13 void addedge(int x,int y,int w){
14     next1[cnt] = head[x];
15     to[cnt] = y;
16     weight[cnt] = w;
17     head[x] = cnt++;

```

```

18 }
19 void solve(void) {
20     int n, m;
21     cin >> n >> m;
22     cnt=1;
23     fill(head.begin(),head.end(),0);
24     for (int i = 0, x, y, w; i < m; i++) {
25         cin >> x >> y >> w;
26         addedge(x,y,w);
27         if (w >= 0) {
28             addedge(y,x,w);
29         }
30     }
31     queue<int> heap;
32     vector<int> distance(n + 1, INT_MAX);
33     vector<int> upcnt(n + 1, 0);
34     vector<int> visited(n + 1, 0);
35     heap.push(1);
36     distance[1] = 0;
37     upcnt[1]++;
38     visited[1] = 1;
39     while (!heap.empty()) {
40         int u = heap.front();
41         heap.pop();
42         visited[u] = 0;
43         for (int ei = head[u]; ei > 0; ei = next1[ei]) {
44             int v = to[ei];
45             int w = weight[ei];
46             if (w + distance[u] < distance[v]) {
47                 distance[v] = w + distance[u];
48                 if (visited[v]==0) {
49                     if ((++upcnt[v]) > n-1) {
50                         //存在负权回路
51                         cout << "YES" << endl;
52                         return;
53                     }
54                     visited[v] = 1;
55                     heap.push(v);
56                 }
57             }
58         }
59     }
60     cout << "NO" << endl;
61 }

```

最小生成树

树上的一条无回路的连通所有节点的一条权值最小的路径

Kruskal算法

```

1 // Kruskal 算法：利用并查集求最小生成树
2 int kruskal() {
3     int n, m;

```

```

4     cin >> n >> m;
5     vector<tuple<int,int,int>> edges; // (权值, 点u, 点v)
6     for (int i = 0; i < m; i++) {
7         int u, v, w;
8         cin >> u >> v >> w;
9         edges.push_back({w, u, v});
10    }
11    sort(edges.begin(), edges.end());
12    vector<int> parent(n + 1);
13    for (int i = 1; i <= n; i++) parent[i] = i;
14    auto find = [&](auto && f, int x) -> int{
15        if(parent[x] != x){
16            parent[x] = f(f, parent[x]);
17        }
18        return parent[x];
19    };
20    int sum = 0;
21    for (auto [w, u, v] : edges) {
22        int pu = find(find, u), pv = find(find, v);
23        if (pu != pv) {
24            parent[pu] = pv;
25            sum += w;
26        }
27    }
28    return sum;
29}
30

```

Prim算法

```

1 // Prim 算法
2 int prim() {
3     int n, m;
4     cin >> n >> m;
5     vector<vector<pair<int,int>>> graph(n + 1);
6     for (int i = 0; i < m; i++) {
7         int u, v, w;
8         cin >> u >> v >> w;
9         graph[u].push_back({v, w});
10        graph[v].push_back({u, w});
11    }
12    vector<bool> visited(n + 1, false);
13    // 优先队列: pair(权值, 节点)
14    priority_queue<pair<int,int>, vector<pair<int,int>>,
greater<pair<int,int>>> pq;
15    visited[1] = true;
16    for (auto edge : graph[1]) pq.push({edge.second, edge.first});
17    int sum = 0;
18    while (!pq.empty()) {
19        auto [w, u] = pq.top();
20        pq.pop();
21        if (visited[u]) continue;
22        visited[u] = true;
23        sum += w;
24        for (auto edge : graph[u])
25            if (!visited[edge.first])

```

```

26             pq.push({edge.second, edge.first});
27     }
28     return sum;
29 }

```

二叉树

二叉树的序列化与反序列化及先序构造

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 struct TreeNode{
4     int val;
5     TreeNode * left, * right;
6     TreeNode (int x){
7         val = x;
8     }
9 };
10 //先序的序列化与反序列化
11 class Codec {
12 public:
13     string serialize(TreeNode* root) {
14         string r;
15         serialize_h(root,r);
16         return r;
17     }
18     TreeNode* deserialize(string data) {
19         vector<string> vals = split(data,',');
20         int cnt=0;
21         return deserialize_h(vals,cnt);
22     }
23 private:
24     void serialize_h(TreeNode* root,string& r){
25         if(root==nullptr){
26             r+="#,";
27         }else{
28             r+=to_string(root->val)+",";
29             serialize_h(root->left,r);
30             serialize_h(root->right,r);
31         }
32     }
33     TreeNode* deserialize_h(const vector<string>& vals,int& cnt){
34         if(cnt >= vals.size() || vals[cnt]==="#" ){
35             cnt++;
36             return nullptr;
37         }
38         TreeNode* node = new TreeNode(stoi(vals[cnt++]));
39         node->left=deserialize_h(vals,cnt);
40         node->right=deserialize_h(vals,cnt);
41         return node;
42     }
43     vector<string> split(const string&s,char delimiter){
44         vector<string> tokens;
45         string token;
46         istringstream tokenStream(s);

```

```

47     while (getline(tokenStream, token, delimiter)) {
48         tokens.push_back(token);
49     }
50     return tokens;
51 }
52 //按层的序列化与反序列化
53 string serialize1(TreeNode* root) {
54     if (root==nullptr) return "";
55     string result;
56     queue<TreeNode*> q;
57     q.push(root);
58     while (!q.empty()) {
59         TreeNode* node = q.front();
60         q.pop();
61         if (node!=nullptr) {
62             result += to_string(node->val)+",";
63             q.push(node->left);
64             q.push(node->right);
65         } else result+="#,";
66     }
67     return result;
68 }
69 TreeNode* deserialize1(string data) {
70     if (data.empty()) return nullptr;
71     stringstream ss(data);
72     string item;
73     getline(ss, item, ',');
74     TreeNode* root = new TreeNode(stoi(item));
75     queue<TreeNode*> q;
76     q.push(root);
77     while (!q.empty()) {
78         TreeNode* node = q.front();
79         q.pop();
80         if (getline(ss, item, ',')) {
81             if (item=="#") {
82                 node->left=nullptr;
83             } else{
84                 node->left = new TreeNode(stoi(item));
85                 q.push(node->left);
86             }
87             if (getline(ss, item, ',')) {
88                 if (item=="#") {
89                     node->right=nullptr;
90                 } else{
91                     node->right=new TreeNode(stoi(item));
92                     q.push(node->right);
93                 }
94             }
95         }
96     }
97 }

```

利用先序和中序构造二叉树

```
1 //先序和中序构造二叉树
2 class Solution {
3 public:
4     TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
5         if(preorder.empty() || inorder.empty() || preorder.size() != inorder.size()) return nullptr;
6         unordered_map<int,int> map;
7         for(int i=0;i<inorder.size();i++) map[inorder[i]]=i;
8         return
9             build(preorder,0,inorder.size()-1,inorder,0,inorder.size()-1,map);
10    }
11 private:
12     TreeNode* build(const vector<int>& preorder,int l1,int r1,const
13     vector<int>& inorder,int l2,int r2,
14     unordered_map<int,int>& map) {
15         if(l1>r1) return nullptr;
16         TreeNode* root = new TreeNode(preorder[l1]);
17         if(l1==r1) return root;
18         int k=map[preorder[l1]];
19         root->left = build(preorder,l1+1,l1+k-1,inorder,l2,k-1,map);
20         root->right = build(preorder,l1+k-1+1,r1,inorder,k+1,r2,map);
21         return root;
22     }
23 };
```

二叉树中LCA算法

```
1 //LCA问题
2 //求二叉树两个节点的最近公共祖先
3 class Solution {
4 public:
5     TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
6         if(root==nullptr || root==p || root==q) {
7             return root;
8         }
9         TreeNode* l = lowestCommonAncestor(root->left,p,q);
10        TreeNode* r = lowestCommonAncestor(root->right,p,q);
11        if(l!=nullptr && r!=nullptr) {
12            //左右两边都已经找到
13            return root;
14        }
15        if(l==nullptr && r==nullptr) {
16            //左右两边都没有找到
17            return nullptr;
18        }
19        //只有一边找到了，返回找到的一边
20        return l==nullptr?r:l;
21    }
22 };
23 //求线索二叉树两个节点的最近公共祖先
24 class Solution {
25 public:
26     TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
27         // root从上到下
```

```

28     // 如果先遇到了p，说明p是答案
29     // 如果先遇到了q，说明q是答案
30     // 如果root在p~q的值之间，不用管p和q谁大谁小，只要root在中间，那么此时的root就是答
案
31     // 如果root在p~q的值的左侧，那么root往右移动
32     // 如果root在p~q的值的右侧，那么root往左移动
33     while (root->val != p->val && root->val != q->val) {
34         if (root->val < max(p->val, q->val) && root->val > min(p->val, q-
>val)) {
35             break;
36         }
37         root = root->val < min(q->val, p->val) ? root->right : root->left;
38     }
39 }
40 };

```

Morris遍历

Morris遍历是二叉树遍历的进阶算法，相比递归（ $O(h)$ 空间）和非递归栈实现（ $O(h)$ 空间），它巧妙利用树中空闲指针（叶节点左右为空），将空间复杂度极限优化至 $O(1)$ ，时间复杂度仍为 $O(N)$ ，实现高效无栈遍历。

实现原则（以当前节点cur为例）：

1. 若cur无左子树，直接向右移动： `cur = cur->right;`
2. 若cur有左子树，找到左子树最右节点mostRight：
 - 若`mostRight->right == null`（首次访问）：设`mostRight->right = cur`，然后向左移动：`cur = cur-
 >left;`
 - 若`mostRight->right == cur`（二次访问）：恢复`mostRight->right = null`，然后向右移动：`cur = cur-
 >right。`

遵循此机制，即完成Morris遍历框架。

实质洞察：通过临时“线索”指针（右指针线程化），确保无左子树节点访问1次，有左子树节点访问2次（首次入左、二次出左），从而模拟递归路径而不耗额外空间。适用于前/中/后序遍历及衍生问题（如BST校验、最小深度、LCA）。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct TreeNode {
4     int val;
5     TreeNode* left;
6     TreeNode* right;
7     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
8 };
9 // ===== 通用Morris框架 =====
10 void morrisFrame(TreeNode* head, function<void(TreeNode*)> process) {
11     if (!head) return;
12     TreeNode* cur = head;
13     TreeNode* mostRight = nullptr;
14     while (cur) {

```

```

15     mostRight = cur->left;
16     if (mostRight) {
17         while (mostRight->right && mostRight->right != cur) {
18             mostRight = mostRight->right;
19         }
20         if (!mostRight->right) { // 第一次
21             mostRight->right = cur;
22             cur = cur->left;
23             continue;
24         } else { // 第二次
25             mostRight->right = nullptr;
26         }
27     }
28     process(cur); // 自定义处理
29     cur = cur->right;
30 }
31 }
32
33 // ===== 1. 前序遍历 (LeetCode 144) =====
34 vector<int> preorderTraversal(TreeNode* head) {
35     vector<int> ans;
36     TreeNode* cur = head;
37     TreeNode* mostRight = nullptr;
38     while (cur) {
39         mostRight = cur->left;
40         if (mostRight) {
41             while (mostRight->right && mostRight->right != cur) mostRight =
42             mostRight->right;
43             if (!mostRight->right) {
44                 ans.push_back(cur->val); // 访问
45                 mostRight->right = cur;
46                 cur = cur->left; continue;
47             } else {
48                 mostRight->right = nullptr;
49             }
50         } else {
51             ans.push_back(cur->val); // 访问
52         }
53         cur = cur->right;
54     }
55     return ans;
56 }
57
58 // ===== 2. 中序遍历 (LeetCode 94) =====
59 vector<int> inorderTraversal(TreeNode* head) {
60     vector<int> ans;
61     morrisFrame(head, [&] (TreeNode* node) { ans.push_back(node->val); });
62     return ans;
63 }
64
65 // ===== 3. 后序遍历 (LeetCode 145) =====
66 void reverseRightEdge(TreeNode* from, vector<int>& ans) {
67     if (!from) return;
68     TreeNode* pre = nullptr;
69     TreeNode* next = nullptr;
70     TreeNode* tail = from;

```

```

70 // 第一遍翻转
71 while (tail) {
72     next = tail->right;
73     tail->right = pre;
74     pre = tail;
75     tail = next;
76 }
77 // pre 现在是原尾部，链已反转
78 // 收集：从pre遍历right（反转链），得到逆序
79 TreeNode* cur = pre;
80 while (cur) {
81     ans.push_back(cur->val);
82     cur = cur->right;
83 }
84 // 恢复：从pre（原尾）再翻转一次
85 tail = pre;
86 pre = nullptr;
87 while (tail) {
88     next = tail->right;
89     tail->right = pre;
90     pre = tail;
91     tail = next;
92 }
93 // 现在链恢复，原头是pre
94 }

95
96 vector<int> postorderTraversal(TreeNode* head) {
97     vector<int> ans;
98     if (!head) return ans;
99     TreeNode* cur = head;
100    TreeNode* mostRight = nullptr;
101    while (cur) {
102        mostRight = cur->left;
103        if (mostRight) {
104            while (mostRight->right && mostRight->right != cur) mostRight =
mostRight->right;
105            if (!mostRight->right) {
106                mostRight->right = cur;
107                cur = cur->left; continue;
108            } else {
109                mostRight->right = nullptr;
110                reverseRightEdge(cur->left, ans); // 左边界逆序
111            }
112        }
113        cur = cur->right;
114    }
115    reverseRightEdge(head, ans); // 根右边界逆序
116    return ans;
117 }

118
119 // ===== 4. 检查BST (LeetCode 98) =====
120 bool isValidBST(TreeNode* head) {
121     if (!head) return true;
122     TreeNode* cur = head;
123     TreeNode* mostRight = nullptr;
124     TreeNode* pre = nullptr;

```

```

125     bool valid = true;
126     while (cur) {
127         mostRight = cur->left;
128         if (mostRight) {
129             while (mostRight->right && mostRight->right != cur) mostRight =
130             mostRight->right;
131             if (!mostRight->right) {
132                 mostRight->right = cur;
133                 cur = cur->left; continue;
134             } else {
135                 mostRight->right = nullptr;
136             }
137             if (pre && pre->val >= cur->val) valid = false;
138             pre = cur;
139             cur = cur->right;
140         }
141         return valid;
142     }
143
144 // ===== 5. 最小深度 (LeetCode 111) =====
145 int minDepth(TreeNode* head) {
146     if (!head) return 0;
147     TreeNode* cur = head;
148     TreeNode* mostRight = nullptr;
149     int preLevel = 0;
150     int rightLen = 0;
151     int ans = INT_MAX;
152     while (cur) {
153         mostRight = cur->left;
154         if (mostRight) {
155             rightLen = 1;
156             while (mostRight->right && mostRight->right != cur) {
157                 rightLen++;
158                 mostRight = mostRight->right;
159             }
160             if (!mostRight->right) {
161                 preLevel++;
162                 mostRight->right = cur;
163                 cur = cur->left; continue;
164             } else {
165                 if (!mostRight->left) ans = min(ans, preLevel); // 左无子
166                 preLevel -= rightLen;
167                 mostRight->right = nullptr;
168             }
169         } else {
170             preLevel++;
171         }
172         cur = cur->right;
173     }
174     // 最右叶
175     rightLen = 1; cur = head;
176     while (cur->right) { rightLen++; cur = cur->right; }
177     if (!cur->left) ans = min(ans, rightLen);
178     return ans == INT_MAX ? 0 : ans;
179 }
```

```

180
181 // ===== 6. LCA (LeetCode 236) =====
182 TreeNode* preOrderFind(TreeNode* head, TreeNode* o1, TreeNode* o2) { // 先序找
183     第一个o1/o2
184     if (!head) return nullptr;
185     TreeNode* cur = head;
186     TreeNode* mostRight = nullptr;
187     TreeNode* res = nullptr;
188     while (cur) {
189         mostRight = cur->left;
190         if (mostRight) {
191             while (mostRight->right && mostRight->right != cur) mostRight =
192             mostRight->right;
193             if (!mostRight->right) {
194                 if (!res && (cur == o1 || cur == o2)) res = cur;
195                 mostRight->right = cur;
196                 cur = cur->left; continue;
197             } else {
198                 mostRight->right = nullptr;
199             }
200         } else {
201             if (!res && (cur == o1 || cur == o2)) res = cur;
202         }
203         cur = cur->right;
204     }
205     return res;
206 }
207
208 bool rightCheck(TreeNode* head, TreeNode* target) { // 右边界找target
209     while (head) {
210         if (head == target) return true;
211         head = head->right;
212     }
213     return false;
214 }
215
216 TreeNode* lowestCommonAncestor(TreeNode* head, TreeNode* o1, TreeNode* o2) {
217     if (preOrderFind(o1->left, o1, o2) || preOrderFind(o1->right, o1, o2))
218     return o1;
219     if (preOrderFind(o2->left, o1, o2) || preOrderFind(o2->right, o1, o2))
220     return o2;
221     TreeNode* left = preOrderFind(head, o1, o2);
222     if (!left) return head; // 如果left为空, head是LCA
223     TreeNode* cur = head;
224     TreeNode* mostRight = nullptr;
225     TreeNode* lca = nullptr;
226     while (cur) {
227         mostRight = cur->left;
228         if (mostRight) {
229             while (mostRight->right && mostRight->right != cur) mostRight =
230             mostRight->right;
231             if (!mostRight->right) {
232                 mostRight->right = cur;
233                 cur = cur->left; continue;
234             } else {
235                 mostRight->right = nullptr;
236             }
237         }
238     }
239     return lca;
240 }
```

```

231         if (!lca) {
232             if (rightCheck(cur->left, left)) {
233                 if (preOrderFind(left->right, o1, o2)) lca = left;
234                 left = cur;
235             }
236         }
237     }
238     cur = cur->right;
239 }
240
241 return lca ? lca : left;
242 }
243
244 // ===== 示例测试 =====
245 int main() {
246     // 建树: LeetCode 236 样例
247     TreeNode* root = new TreeNode(3);
248     root->left = new TreeNode(5);
249     root->left->left = new TreeNode(6);
250     root->left->right = new TreeNode(2);
251     root->left->right->left = new TreeNode(7);
252     root->left->right->right = new TreeNode(4);
253     root->right = new TreeNode(1);
254     root->right->left = new TreeNode(8);
255     root->right->right = new TreeNode(0);
256     // 测试
257     cout << "Preorder: ";
258     for (int v : preorderTraversal(root)) cout << v << " ";
259     cout << endl;
260
261     cout << "Inorder: ";
262     for (int v : inorderTraversal(root)) cout << v << " ";
263     cout << endl;
264
265     cout << "Postorder: ";
266     for (int v : postorderTraversal(root)) cout << v << " ";
267     cout << endl;
268
269     cout << "Is BST: " << (isValidBST(root) ? "Yes" : "No") << endl;
270     cout << "Min Depth: " << minDepth(root) << endl;
271
272     TreeNode* n1 = root->left->right->left; // 7
273     TreeNode* n2 = root->right->left; // 8
274     cout << "LCA(7,8): " << lowestCommonAncestor(root, n1, n2)->val << endl;
275
276     return 0;
277 }

```

树状数组

一维

单点修改区间查询

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 constexpr int N = 5e5 + 1;
4 vector<int> v1(N);
5 int n, m;
6 int low(int x) {
7     return x & -x;
8 }
9 void add(int i, int x) {
10     while(i <= n) {
11         v1[i] += x;
12         i += low(i);
13     }
14 }
15 int get(int i) {
16     int ans = 0;
17     while(i) {
18         ans += v1[i];
19         i -= low(i);
20     }
21     return ans;
22 }
23 int range(int i, int j) {
24     return get(j) - get(i - 1);
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     cout.tie(nullptr);
30     cin >> n >> m;
31     for(int i = 1, x; i <= n; i++) {
32         cin >> x;
33         add(i, x);
34     }
35     for(int i = 0, op, x, y; i < m; i++) {
36         cin >> op >> x >> y;
37         if(op == 1) {
38             add(x, y);
39         } else {
40             cout << range(x, y) << "\n";
41         }
42     }
43     return 0;
44 }

```

区间修改单点查询

在单点基础上，利用一维差分，最后求从0到i的累加和即为i点的值

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 constexpr int N = 5e5 + 1;
4 vector<long long> v1(N, 0);
5 int n, m;

```

```

6 int low(int x) {
7     return x & -x;
8 }
9 void add(int i, int x) {
10    while(i <= n) {
11        v1[i] += x;
12        i += low(i);
13    }
14 }
15 long long get(int i) {
16     long long ans = 0;
17     while(i) {
18         ans += v1[i];
19         i -= low(i);
20     }
21     return ans;
22 }
23 long long range(int i, int j) {
24     return get(j) - get(i - 1);
25 }
26 int main() {
27     ios::sync_with_stdio(false);
28     cin.tie(nullptr);
29     cout.tie(nullptr);
30     cin >> n >> m;
31     for(int i = 1, x; i <= n; i++) {
32         cin >> x;
33         add(i, x);
34         add(i + 1, -x);
35     }
36     for(int i = 0, op, x, y, z; i < m; i++) {
37         cin >> op;
38         if(op == 1) {
39             cin >> x >> y >> z;
40             add(x, z);
41             add(y + 1, -z);
42         } else {
43             cin >> x;
44             cout << range(1, x) << "\n";
45         }
46     }
47     return 0;
48 }

```

范围修改范围查询

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 constexpr int N = 1e5 + 1;
4 vector<long long> v1(N);
5 vector<long long> v2(N);
6 int n, m;
7 int low(int x) {
8     return x & -x;
9 }
10 void add(int i, int x, vector<long long> & v) {

```

```

11     while(i <= n) {
12         v[i] += x;
13         i += low(i);
14     }
15 }
16 long long get(int i, vector<long long> & v) {
17     long long ans = 0;
18     while(i) {
19         ans += v[i];
20         i -= low(i);
21     }
22     return ans;
23 }
24 long long range(int i, int j) {
25     return j * get(j, v1) - get(j, v2) - ((i - 1) * get(i - 1, v1) - get(i - 1,
v2));
26 }
27 void add(int x, int y, long long w) {
28     add(x, w, v1);
29     add(y + 1, -w, v1);
30     add(x, (x - 1) * w, v2);
31     add(y + 1, -y * w, v2);
32 }
33 int main() {
34     ios::sync_with_stdio(false);
35     cin.tie(nullptr);
36     cout.tie(nullptr);
37     cin >> n >> m;
38     for(int i = 1, x; i <= n; i++) {
39         cin >> x;
40         add(i, i, x);
41     }
42     for(int i = 0, op, x, y, z; i < m; i++) {
43         cin >> op;
44         if(op == 1) {
45             cin >> x >> y >> z;
46             add(x, y, z);
47         } else{
48             cin >> x >> y;
49             cout << range(x, y) << "\n";
50         }
51     }
52     return 0;
53 }

```

二维

单点修改范围查询

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int low(int i) {
4     return i & -i;
5 }
6 void add(vector<vector<int>> & bit, int x, int y, int w, int n, int m) {

```

```

7     for (int i = x; i <= n; i += low(i)) {
8         for (int j = y; j <= m; j += low(j)) {
9             bit[i][j] += w;
10        }
11    }
12 }
13 int query(vector<vector<int>> &bit, int x, int y) {
14     int ans = 0;
15     for (int i = x; i > 0; i -= low(i)) {
16         for (int j = y; j > 0; j -= low(j)) {
17             ans += bit[i][j];
18         }
19     }
20     return ans;
21 }
22 int get(vector<vector<int>> &bit, int x1, int y1, int x2, int y2) {
23     if (x1 > x2 || y1 > y2) return 0;
24     return query(bit, x2, y2) - query(bit, x1 - 1, y2) - query(bit, x2, y1 - 1)
25 + query(bit, x1 - 1, y1 - 1);
26 }
27 void solve() {
28     int n, m;
29     cin >> n >> m;
30     vector<vector<int>> v1(n + 1, vector<int>(m + 1, 0)); // 原始数组
31     vector<vector<int>> bit(n + 1, vector<int>(m + 1, 0)); // 树状数组
32     for (int i = 1; i <= n; i++) {
33         for (int j = 1; j <= m; j++) {
34             cin >> v1[i][j];
35             add(bit, i, j, v1[i][j], n, m);
36         }
37     }
38     int q;
39     cin >> q;
40     int type;
41     int x, y, w;
42     int diff;
43     int x1, y1, x2, y2;
44     while (q--) {
45         cin >> type;
46         if (type == 1) {
47             cin >> x >> y >> w;
48             diff = w - v1[x][y];
49             v1[x][y] = w;
50             add(bit, x, y, diff, n, m);
51         } else if (type == 2) {
52             cin >> x1 >> y1 >> x2 >> y2;
53             cout << get(bit, x1, y1, x2, y2) << "\n";
54         }
55     }
56 signed main() {
57     ios::sync_with_stdio(false);
58     cin.tie(nullptr);
59     int t = 1;
60     // cin >> t;
61     while (t--) {

```

```

62     solve();
63 }
64 return 0;
65 }
```

范围查询范围修改

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 2050;
4 int bit1[N][N];
5 int bit2[N][N];
6 int bit3[N][N];
7 int bit4[N][N];
8 int n, m;
9 int a, b, c, d, w;
10 int v1, v2, v3, v4;
11 int low(int x) {
12     return x & -x;
13 }
14 void add(int x, int y, int v) {
15     v1 = v;
16     v2 = x * v;
17     v3 = y * v;
18     v4 = x * y * v;
19     for(int i = x; i <= n; i += low(i)) {
20         for(int j = y; j <= m; j += low(j)) {
21             bit1[i][j] += v1;
22             bit2[i][j] += v2;
23             bit3[i][j] += v3;
24             bit4[i][j] += v4;
25         }
26     }
27 }
28 int get(int x, int y) {
29     int ans = 0;
30     for(int i = x; i; i -= low(i)) {
31         for(int j = y; j; j -= low(j)) {
32             ans += (x + 1) * (y + 1) * bit1[i][j] - (y + 1) * bit2[i][j] - (x +
1) * bit3[i][j] + bit4[i][j];
33         }
34     }
35     return ans;
36 }
37 void add(int x1, int y1, int x2, int y2, int v) {
38     add(x1, y1, v);
39     add(x2 + 1, y2 + 1, v);
40     add(x1, y2 + 1, -v);
41     add(x2 + 1, y1, -v);
42 }
43 int get(int x1, int y1, int x2, int y2) {
44     return get(x2, y2) + get(x1 - 1, y1 - 1) - get(x1 - 1, y2) - get(x2, y1 -
1);
45 }
46 void solve(char C) {
47     if(C == 'L') {
```

```

48     cin >> a >> b >> c >> d >> w;
49     add(a, b, c, d, w);
50 }else if(C != 'X') {
51     cin >> a >> b >> c >> d;
52     cout << get(a, b, c, d) << "\n";
53 }else{
54     cin >> n >> m;
55 }
56 }
57 signed main() {
58     ios::sync_with_stdio(false);
59     cin.tie(nullptr);
60     char C;
61     while (cin >> C) {
62         solve(C);
63     }
64     return 0;
65 }
```

线段树

范围增加范围查询累加和

```

1 // https://www.luogu.com.cn/problem/P3372
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main(){
8     cin.tie(0) -> ios::sync_with_stdio(0);
9     int t = 1;
10    // cin >> t;
11    while (t--) {
12        int n, q;
13        cin >> n >> q;
14        vector<int> v1(n + 1), sum((n << 2) + 1), add((n << 2) + 1);
15        for(int i = 1; i <= n; i ++){
16            cin >> v1[i];
17        }
18        auto build = [&](auto && f, int i, int l, int r) -> void{
19            if(l == r){
20                sum[i] = v1[l];
21            }else{
22                int mid = (l + r) >> 1;
23                f(f, i << 1, l, mid);
24                f(f, i << 1 | 1, mid + 1, r);
25                sum[i] = sum[i << 1] + sum[i << 1 | 1];
26            }
27            add[i] = 0;
28        };
29        auto up = [&](int i){
30            sum[i] = sum[i << 1] + sum[i << 1 | 1];
31        };
32        auto lazy = [&](int i, int val, int len){
```

```

33         sum[i] += val * len;
34         add[i] += val;
35     };
36     auto down = [&](int i, int lenl, int lenr){
37         if(add[i] == 0){
38             return ;
39         }
40         lazy(i << 1, add[i], lenl);
41         lazy(i << 1 | 1, add[i], lenr);
42         add[i] = 0;
43     };
44     auto add1 = [&](auto && f,int i, int val, int basel, int baser, int l,
int r) -> void{
45         if(basel <= l && baser >= r){
46             lazy(i, val, r - l + 1);
47         }else{
48             int mid = (l + r) >> 1;
49             down(i, mid - 1 + 1, r - mid);
50             if(basel <= mid){
51                 f(f, i << 1, val, basel, baser, l, mid);
52             }
53             if(baser >= mid + 1){
54                 f(f, i << 1 | 1, val, basel ,baser, mid + 1, r);
55             }
56             up(i);
57         }
58     };
59     build(build, 1, 1, n);
60     auto query = [&](auto && f, int i, int basel, int baser, int l, int r)
-> int {
61         if(basel <= l && baser >= r){
62             return sum[i];
63         }else{
64             int mid = (l + r) >> 1;
65             down(i, mid - 1 + 1, r - mid);
66             int ans = 0;
67             if(basel <= mid){
68                 ans += f(f, i << 1, basel, baser, l, mid);
69             }
70             if(baser >= mid + 1){
71                 ans += f(f, i << 1 | 1, basel, baser, mid + 1, r);
72             }
73             return ans;
74         }
75     };
76     for(int i = 0; i < q; i ++){
77         int op;
78         cin >> op;
79         if(op == 1){
80             int l, r, val;
81             cin >> l >> r >> val;
82             add1(add1, 1, val, l , r, 1, n);
83         }else{
84             int l, r;
85             cin >> l >> r;
86             cout << query(query, 1, l, r, 1, n) << "\n";

```

```
87         }
88     }
89 }
90 return 0;
91 }
```

有优先级调整的多范围修改任务

```
1 // https://www.luogu.com.cn/problem/P1253
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main(){
8     cin.tie(0) -> ios::sync_with_stdio(0);
9     int t = 1;
10    // cin >> t;
11    while (t--) {
12        int n, q;
13        cin >> n >> q;
14        vector<int> v1(n + 1), change((n << 2)), max1((n << 2)), add((n << 2));
15        vector<bool> has((n << 2));
16        for(int i = 1; i <= n; i++) {
17            cin >> v1[i];
18        }
19        auto biuld = [&](auto && f, int i, int l, int r) -> void {
20            if(l == r){
21                max1[i] = v1[l];
22            }else{
23                int mid = (l + r) >> 1;
24                f(f, i << 1, l, mid);
25                f(f, i << 1 | 1, mid + 1, r);
26                max1[i] = max(max1[i << 1], max1[i << 1 | 1]);
27            }
28            has[i] = false;
29            add[i] = 0;
30        };
31        biuld(biuld, 1, 1, n);
32        auto up = [&](int i) {
33            max1[i] = max(max1[i << 1], max1[i << 1 | 1]);
34        };
35        auto lazyadd = [&](int i, int val) {
36            max1[i] += val;
37            add[i] += val;
38        };
39        auto lazyset = [&](int i, int val) {
40            max1[i] = val;
41            change[i] = val;
42            add[i] = 0;
43            has[i] = true;
44        };
45        auto down = [&](int i) {
46            if(has[i]){
47                lazyset(i << 1, change[i]);
```

```

48         lazyset(i << 1 | 1, change[i]);
49         has[i] = false;
50     }
51     if(add[i]) {
52         lazyadd(i << 1, add[i]);
53         lazyadd(i << 1 | 1, add[i]);
54         add[i] = 0;
55     }
56 }
57 auto add1 = [&](auto && f, int i, int val, int basel, int baser, int
l, int r) -> void {
58     if(basel <= l && baser >= r){
59         lazyadd(i, val);
60     }else{
61         int mid = (l + r) >> 1;
62         down(i);
63         if(basel <= mid){
64             f(f, i << 1, val, basel, baser, l, mid);
65         }
66         if(baser > mid){
67             f(f, i << 1 | 1, val, basel, baser, mid + 1, r);
68         }
69         up(i);
70     }
71 };
72 auto set = [&](auto && f, int i, int val, int basel, int baser, int l,
int r) -> void {
73     if(basel <= l && baser >= r){
74         lazyset(i, val);
75     }else{
76         int mid = (l + r) >> 1;
77         down(i);
78         if(basel <= mid){
79             f(f, i << 1, val, basel, baser, l, mid);
80         }
81         if(baser > mid){
82             f(f, i << 1 | 1, val, basel, baser, mid + 1, r);
83         }
84         up(i);
85     }
86 };
87 auto query = [&](auto && f, int i, int basel, int baser, int l, int r)
-> int{
88     if(basel <= l && baser >= r){
89         return max1[i];
90     }else{
91         int mid = (l + r) >> 1;
92         down(i);
93         int ans = -1e18;
94         if(basel <= mid){
95             ans = max(ans, f(f, i << 1, basel, baser, l, mid));
96         }
97         if(baser > mid){
98             ans = max(ans, f(f, i << 1 | 1, basel, baser, mid + 1,
r));
99         }

```

```

100         return ans;
101     }
102 }
103 for(int i = 0; i < q; i ++){
104     int op;
105     cin >> op;
106     int l, r;
107     cin >> l >> r;
108     if(op == 2){
109         int x;
110         cin >> x;
111         add1(add1, 1, x, l, r, 1, n);
112     }else if(op == 1){
113         int x;
114         cin >> x;
115         set(set, 1, x, l, r, 1, n);
116     }else{
117         cout << query(query, 1, l, r, 1, n) << "\n";
118     }
119 }
120 }
121 return 0;
122 }
```

离散化

离散化之后要注意间隙，如覆盖问题，要在离散化之后的两个差值不为一的数据中间添加一个数

区间最值和历史最值

```

1 // https://www.luogu.com.cn/problem/P6242
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main(){
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    int t = 1;
11    // cin >> t;
12    while(t--){
13        int n, m;
14        cin >> n >> m;
15        vector<int> max1(n<<2),      // 区间当前最大值
16                          sec(n<<2),      // 区间严格次大值
17                          cnt(n<<2),      // 区间等于最大值的元素个数
18                          sum(n<<2),      // 区间元素和
19                          his(n<<2),      // 区间历史最大值（曾经出现过的最大值）
20                          add1(n<<2),      // 懒标记 A：对“当前最大值元素”做加法
21                          add2(n<<2),      // 懒标记 B：对“非最大值元素”做加法
22                          adds1(n<<2),      // 历史最大值懒标记 A
23                          adds2(n<<2);      // 历史最大值懒标记 B
24        auto up = [&](int i){
25            sum[i] = sum[i<<1] + sum[i<<1|1];
26        }
27    }
28 }
```

```

26     his[i] = max(his[i<<1], his[i<<1|1]);
27     max1[i] = max(max1[i<<1], max1[i<<1|1]);
28     // 合并 cnt 与 sec
29     if(max1[i<<1] == max1[i<<1|1]){
30         cnt[i] = cnt[i<<1] + cnt[i<<1|1];
31         sec[i] = max(sec[i<<1], sec[i<<1|1]);
32     } else if(max1[i<<1] > max1[i<<1|1]){
33         cnt[i] = cnt[i<<1];
34         sec[i] = max(sec[i<<1], max1[i<<1|1]);
35     } else {
36         cnt[i] = cnt[i<<1|1];
37         sec[i] = max(sec[i<<1|1], max1[i<<1]);
38     }
39 }
40 auto build = [&](auto&& self, int i, int l, int r) -> void {
41     if(l == r){
42         int x;
43         cin >> x;
44         max1[i] = sum[i] = his[i] = x; // 当前值、区间和值、历史最大均为 x
45         cnt[i] = 1; // 只有一个元素
46         sec[i] = LLONG_MIN / 2; // 次大值设为 -INF (用很小的数)
47     } else {
48         int mid = (l + r) >> 1;
49         self(self, i<<1, l, mid);
50         self(self, i<<1|1, mid+1, r);
51         up(i);
52     }
53     // 懒标记初始为 0
54     add1[i] = add2[i] = adds1[i] = adds2[i] = 0;
55 };
56 build(build, 1, 1, n);
57 // 核心懒标记下发函数
58 // val1: 加到“当前最大值元素”上的增量
59 // val2: 加到“非最大值元素”上的增量
60 // val3/val4: 用于更新历史最大值的增量
61 // len: 本区间长度
62 auto lazy = [&](int i, int val1, int val2, int val3, int val4, int
len) {
63     // 更新历史最大值
64     his[i] = max(his[i], max1[i] + val3);
65     // 更新子树的历史懒标记
66     adds1[i] = max(adds1[i], add1[i] + val3);
67     adds2[i] = max(adds2[i], add2[i] + val4);
68     // 区间和更新
69     sum[i] += val1 * cnt[i] + val2 * (len - cnt[i]);
70     // 更新“当前最大值”与“非最大值”懒标记
71     add1[i] += val1;
72     add2[i] += val2;
73     // 更新当前最大值
74     max1[i] += val1;
75     // 次大值也需加 val2 (若不存在则保持 -INF)
76     if(sec[i] > LLONG_MIN/2) sec[i] += val2;
77 };
78 // 下推函数: 将 i 节点的懒标记传递给左右子节点
79 auto down = [&](int i, int lenL, int lenR) {
80     // 左右子节点新的“当前最大”值

```

```

81     int tL = max1[i<<1], tR = max1[i<<1|1];
82     // 父节点最高值
83     int mx = max(max1[i<<1], max1[i<<1|1]);
84     // 左子树
85     if(tL == mx) {
86         // 左子树中有父区间最大值元素
87         lazy(i<<1, add1[i], add2[i], adds1[i], adds2[i], lenL);
88     } else {
89         // 左子树中只有“非最大值”元素
90         lazy(i<<1, add2[i], add2[i], adds2[i], adds2[i], lenL);
91     }
92     // 右子树 (同理)
93     if(tR == mx) {
94         lazy(i<<1|1, add1[i], add2[i], adds1[i], adds2[i], lenR);
95     } else {
96         lazy(i<<1|1, add2[i], add2[i], adds2[i], adds2[i], lenR);
97     }
98     // 清空父节点懒标记
99     add1[i] = add2[i] = adds1[i] = adds2[i] = 0;
100 }
101 // 区间加法 [bl,br] 全加 val
102 auto addRange = [&](auto&& self, int i, int bl, int br, int val, int
1, int r) -> void {
103     if(bl <= l && r <= br) {
104         // 统一打懒标记, val1=val2=val3=val4=val
105         lazy(i, val, val, val, val, r-l+1);
106     } else {
107         int mid = (l + r) >> 1;
108         down(i, mid-l+1, r-mid);
109         if(bl <= mid)      self(self, i<<1, bl, br, val, l, mid);
110         if(br > mid)      self(self, i<<1|1, bl, br, val, mid+1, r);
111         up(i);
112     }
113 };
114 // 区间 chmin: 把区间内所有大于 val 的值降到 val
115 auto chminRange = [&](auto&& self, int i, int bl, int br, int val, int
1, int r) -> void {
116     if(max1[i] <= val) return; // 本区间最大值已 ≤ val, 无需操作
117     if(bl <= l && r <= br && sec[i] < val) {
118         // 全区间最大值都大于 val, 且次大值 < val
119         // 只需把“最大值元素”降到 val
120         lazy(i, val-max1[i], 0, val-max1[i], 0, r-l+1);
121     } else {
122         int mid = (l + r) >> 1;
123         down(i, mid-l+1, r-mid);
124         if(bl <= mid)      self(self, i<<1, bl, br, val, l, mid);
125         if(br > mid)      self(self, i<<1|1, bl, br, val, mid+1, r);
126         up(i);
127     }
128 };
129 // 区间求和
130 auto querySum = [&](auto&& self, int i, int bl, int br, int l, int r)
-> int {
131     if(bl <= l && r <= br) {
132         return sum[i];
133     } else {

```

```

134         int mid = (l + r) >> 1;
135         down(i, mid-1+1, r-mid);
136         int res = 0;
137         if(bl <= mid)      res += self(self, i<<1, bl, br, l, mid);
138         if(br > mid)      res += self(self, i<<1|1, bl, br, mid+1, r);
139         return res;
140     }
141 }
142 // 区间最大值
143 auto queryMax = [&] (auto&& self, int i, int bl, int br, int l, int r)
-> int {
144     if(bl <= l && r <= br) {
145         return max1[i];
146     } else {
147         int mid = (l + r) >> 1;
148         down(i, mid-1+1, r-mid);
149         int res = LLONG_MIN/2;
150         if(bl <= mid)      res = max(res, self(self, i<<1, bl, br, l,
mid));
151         if(br > mid)      res = max(res, self(self, i<<1|1, bl, br,
mid+1, r));
152         return res;
153     }
154 }
155 // 区间历史最大值
156 auto queryHis = [&] (auto&& self, int i, int bl, int br, int l, int r)
-> int {
157     if(bl <= l && r <= br) {
158         return his[i];
159     } else {
160         int mid = (l + r) >> 1;
161         down(i, mid-1+1, r-mid);
162         int res = LLONG_MIN/2;
163         if(bl <= mid)      res = max(res, self(self, i<<1, bl, br, l,
mid));
164         if(br > mid)      res = max(res, self(self, i<<1|1, bl, br,
mid+1, r));
165         return res;
166     }
167 }
168 // 处理 m 次操作
169 while(m--) {
170     int op, l, r;
171     cin >> op >> l >> r;
172     if(op == 1) {
173         int x; cin >> x;
174         addRange(addRange, 1, l, r, x, 1, n);
175     } else if(op == 2) {
176         int x; cin >> x;
177         chminRange(chminRange, 1, l, r, x, 1, n);
178     } else if(op == 3) {
179         cout << querySum(querySum, 1, l, r, 1, n) << "\n";
180     } else if(op == 4) {
181         cout << queryMax(queryMax, 1, l, r, 1, n) << "\n";
182     } else if(op == 5) {
183         cout << queryHis(queryHis, 1, l, r, 1, n) << "\n";

```

```

184         }
185     }
186 }
187     return 0;
188 }
```

结合摩尔投票

经典摩尔投票，求数组中出现次数大于 $\frac{n}{2}$ 的数

```

1 class Solution {
2 public:
3     int majorityElement(vector<int>& nums) {
4         int now = -1, cnt = 0;
5         for(auto x : nums){
6             if(!cnt){
7                 now = x;
8                 cnt = 1;
9             }else{
10                 if(now == x){
11                     cnt++;
12                 }else{
13                     cnt--;
14                 }
15             }
16         }
17         return now;
18     }
19 };
```

结合线段树求区间海王数

```

1 // https://leetcode.cn/problems/online-majority-element-in-subarray/
2 #define fr first
3 #define sc second
4 class MajorityChecker {
5 public:
6     vector<int> val, cnt, v1;
7     vector<pair<int,int>> v2;
8     int n;
9     void up(int i) {
10         if(!cnt[i << 1] && !cnt[i << 1 | 1]){
11             cnt[i] = 0;
12         }else if(!cnt[i << 1]){
13             cnt[i] = cnt[i << 1 | 1];
14             val[i] = val[i << 1 | 1];
15         }else if(!cnt[i << 1 | 1]){
16             cnt[i] = cnt[i << 1];
17             val[i] = val[i << 1];
18         }else{
19             if(val[i << 1] == val[i << 1 | 1]){
20                 val[i] = val[i << 1];
21                 cnt[i] = cnt[i << 1] + cnt[i << 1 | 1];
22             }else{
23                 if(cnt[i << 1] >= cnt[i << 1 | 1]){


```

```

24         val[i] = val[i << 1];
25         cnt[i] = cnt[i << 1] - cnt[i << 1 | 1];
26     }else{
27         val[i] = val[i << 1 | 1];
28         cnt[i] = cnt[i << 1 | 1] - cnt[i << 1];
29     }
30 }
31 }
32 }
33 void build(int i, int l, int r) {
34     if(l == r){
35         cnt[i] = 1;
36         val[i] = v1[l];
37     }else{
38         int mid = (l + r) >> 1;
39         build(i << 1, l, mid);
40         build(i << 1 | 1, mid + 1, r);
41         up(i);
42     }
43 }
44 pair<int,int> get(int i, int bl, int br, int l, int r){
45     if(bl <= l && br >= r){
46         return {val[i], cnt[i]};
47     }else{
48         int mid = (l + r) >> 1;
49         pair<int,int> left = {0, 0}, right = {0, 0}, ans;
50         if(bl <= mid){
51             left = get(i << 1, bl, br, l, mid);
52         }
53         if(br > mid){
54             right = get(i << 1 | 1, bl, br, mid + 1, r);
55         }
56         if(!left.sc && !right.sc){
57             ans.sc = 0;
58         }else if(!left.sc){
59             ans.fr = right.fr;
60             ans.sc = right.sc;
61         }else if(!right.sc){
62             ans.fr = left.fr;
63             ans.sc = left.sc;
64         }else{
65             if(left.fr == right.fr){
66                 ans.fr = left.fr;
67                 ans.sc = left.sc + right.sc;
68             }else{
69                 if(left.sc >= right.sc){
70                     ans.fr = left.fr;
71                     ans.sc = left.sc - right.sc;
72                 }else{
73                     ans.fr = right.fr;
74                     ans.sc = right.sc - left.sc;
75                 }
76             }
77         }
78         return ans;
79     }
}

```

```

80     }
81     int find(int x, int in) {
82         int l = 0, r = n - 1;
83         int mid, ans = -1;
84         while(l <= r) {
85             mid = (l + r) >> 1;
86             if(v2[mid].fr > x) {
87                 r = mid - 1;
88             }else if(v2[mid].fr < x) {
89                 l = mid + 1;
90             }else{
91
92                 if(v2[mid].sc >= in){
93                     ans = mid;
94                     r = mid - 1;
95                 }else{
96                     l = mid + 1;
97                 }
98             }
99         }
100        return ans;
101    };
102    int find2(int x, int in) {
103        int l = 0, r = n - 1;
104        int mid, ans = -1;
105        while(l <= r) {
106            mid = (l + r) >> 1;
107            if(v2[mid].fr > x) {
108                r = mid - 1;
109            }else if(v2[mid].fr < x) {
110                l = mid + 1;
111            }else{
112
113                if(v2[mid].sc <= in){
114                    ans = mid;
115                    l = mid + 1;
116                }else{
117                    r = mid - 1;
118                }
119            }
120        }
121        return ans;
122    };
123    MajorityChecker(vector<int>& arr) {
124        v1.push_back(0);
125        for(auto x : arr){
126            v1.push_back(x);
127        }
128        n = arr.size();
129        val.resize(n << 2);
130        cnt.resize(n << 2);
131        build(1, 1, n);
132        for(int i = 1; i <= n; i ++){
133            v2.push_back({v1[i], i});
134        }
135        sort(v2.begin(), v2.end(), [] (pair<int,int> a, pair<int,int> b) {

```

```

136         if(a.fr == b.fr) {
137             return a.sc < b.sc;
138         }else{
139             return a.fr < b.fr;
140         }
141     });
142     // for(int i = 0; i < n; i ++){
143     //     cout << v2[i].fr << " " << v2[i].sc << "\n";
144     // }
145 }
146 int query(int left, int right, int threshold) {
147     left++, right++;
148     pair<int,int> temp = get(1, left, right, 1, n);
149     if(temp.sc == 0){
150         return -1;
151     }
152     // cout << temp.fr << " " << temp.sc << " ";
153     int l = find(temp.fr, left), r = find2(temp.fr, right);
154     // cout << "L: " << left << " R: " << right << " l1: " << l << " r1:
155     // " << r << "\n";
156     return (r - l + 1) >= threshold ? temp.fr : -1;
157 }
158 };

```

求一个数组中某一个区间出现了某个数的次数

参考上面摩尔投票结合线段树的例子，将一个数组按照数的大小和出现顺序升序排序，然后用不同逻辑的二分查找得到相对位置

查询最值以及出现位置

```

1 // https://ac.nowcoder.com/acm/contest/108307/F
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 constexpr int N = 1e6;
8 vector<int> v1(N + 1), r(N + 1), p(N + 1);
9 vector<int> W(N << 2 | 1), add1(N << 2 | 1), P(N << 2 | 1), L(N + 1);
10 signed main() {
11     cin.tie(nullptr)->iostream::sync_with_stdio(false);
12     int t = 1;
13     cin >> t;
14     auto build = [&](auto && f, int i, int l, int r) -> void {
15         add1[i] = W[i] = 0;
16         P[i] = r;
17         if(l != r) {
18             int mid = (l + r) >> 1;
19             f(f, i << 1, l, mid);
20             f(f, i << 1 | 1, mid + 1, r);
21         }
22     };
23     auto up = [&](int i) -> void {
24         if(W[i << 1] >= W[i << 1 | 1]) W[i] = W[i << 1], P[i] = P[i << 1];

```

```

25         else W[i] = W[i << 1 | 1], P[i] = P[i << 1 | 1];
26     };
27     auto lazy = [&](int i, int w) -> void {
28         W[i] += w, add1[i] += w;
29     };
30     auto down = [&](int i) -> void {
31         lazy(i << 1, add1[i]), lazy(i << 1 | 1, add1[i]);
32         add1[i] = 0;
33     };
34     auto add = [&](auto && f, int i, int l, int r, int L, int R, int w) -> void
{
35         if(l >= L && r <= R) return lazy(i, w);
36         down(i);
37         int mid = (l + r) >> 1;
38         if(L <= mid) f(f, i << 1, l, mid, L, R, w);
39         if(R > mid) f(f, i << 1 | 1, mid + 1, r, L, R, w);
40         up(i);
41     };
42     while(t --) {
43         int n;
44         cin >> n;
45         build(build, 1, 1, n);
46         fill(r.begin(), r.begin() + 1 + n, n + 1);
47         for(int i = 1; i <= n; i++) {
48             cin >> v1[i];
49             if(p[v1[i]]) r[p[v1[i]]] = i;
50             p[v1[i]] = i;
51         }
52         // for(int i = 1; i <= n; i++) cout << r[i] << " " << p[v1[i]] <<
53         "\n";
54         int ans = 0, ans_l = 2, ans_r = 3;
55         for(int i = 1; i <= n; i++) {
56             // cout << i << " ";
57             if(L[v1[i]]) add(add, 1, 1, n, L[v1[i]], p[v1[i]], -1);
58             L[v1[i]] = r[i] + 1;
59             // cout << L[v1[i]] << " " << p[v1[i]] << " ";
60             add(add, 1, 1, n, L[v1[i]], p[v1[i]], 1);
61             // cout << W[1] << "\n";
62             if(ans < W[1]) ans = W[1], ans_l = i + 1, ans_r = P[1];
63         }
64         cout << ans << "\n" << ans_l << " " << ans_r << "\n";
65         for(int i = 1; i <= n; i++) p[v1[i]] = L[v1[i]] = 0;
66     }
67 }

```

稀疏表ST

Sparse Table (稀疏表) 是一种高效的数据结构，主要用于 **静态数组的区间最值查询 (RMQ)**。

- 适用于：**查询频繁，修改不需要或很少**
- 查询时间复杂度：**O(1)**
- 预处理时间复杂度：**O(n log n)**

支持的运算

Sparse Table 支持 **幂等操作 (idempotent operations)** :

可以求 `min()`、`max()`、`gcd()`

不可求 `lcm()` (不满足幂等性)、`sum()` (不满足幂等性)

核心构建流程 (以最大值为例)

1. 定义

- `st[i][j]` 表示从 **下标 i 开始，长度为 2^j 的区间最大值**

2. 初始化

```
1 for (int i = 1; i <= n; i++) {  
2     st[i][0] = arr[i];  
3 }
```

3. 状态转移

```
1 for (int j = 1; (1 << j) <= n; j++) {  
2     for (int i = 1; i + (1 << j) - 1 <= n; i++) {  
3         st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);  
4     }  
5 }
```

预处理 `log2` 表

为什么要提前预处理 `log2[]` ?

查询时需计算区间长度对应的 `log2(r - l + 1)`，预处理可避免浮点误差和时间浪费

```
1 log2[0] = -1;  
2 for (int i = 1; i <= n; i++) {  
3     log2[i] = log2[i >> 1] + 1;  
4 }
```

区间查询 (以最大值为例)

```
1 int k = log2[r - l + 1];  
2 int res = max(st[l][k], st[r - (1 << k) + 1][k]);
```

这个查询方式的核心是使用**两个长度为 2^k 的区间覆盖整个 $[l, r]$ 区间。**

Luogu P4155: 环形区间覆盖跳跃优化

目标: 给定一系列区间，在环上查找覆盖长度为 m 所需的最少跳跃次数 (倍增思想)

关键用法:

- `st[i][j]` 表示从第 i 个区间开始，跳 2^j 步可覆盖到的位置

构建：

```
1 | st[i][0] = rightmost reachable from i
2 | st[i][j] = st[st[i][j - 1]][j - 1];
```

查询：

```
1 | for (int j = logn; j >= 0; j--) {
2 |     if (v1[st[cur][j]][2] < goal) {
3 |         cur = st[cur][j];
4 |         ans += 1 << j;
5 |     }
6 | }
```

利用 Sparse Table 进行跳跃优化（倍增思想）

每次跳跃代表一段区间，用于计算覆盖长度最少的路径

模板封装（最值查询）

```
1 | const int N = 1e5 + 5, LOG = 20;
2 | int st[N][LOG], log2[N];
3 |
4 | void build_st(const vector<int>& a) {
5 |     int n = a.size() - 1;
6 |     log2[0] = -1;
7 |     for (int i = 1; i <= n; i++) {
8 |         st[i][0] = a[i];
9 |         log2[i] = log2[i >> 1] + 1;
10 |    }
11 |    for (int j = 1; (1 << j) <= n; j++) {
12 |        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
13 |            st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
14 |        }
15 |    }
16 | }
17 |
18 | int query_max(int l, int r) {
19 |     int k = log2[r - l + 1];
20 |     return max(st[l][k], st[r - (1 << k) + 1][k]);
21 | }
```

LCA

最近祖先问题，用于快速查询树中两个节点的最近公共祖先。

采用deep数组和st表

deep数组用于记录深度，st表用于快速查询父亲及以上节点

```
1 | // https://www.luogu.com.cn/problem/P3379
```

```

2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main() {
8     ios::sync_with_stdio(0);
9     cin.tie(0), cout.tie(0);
10    int t = 1;
11    // cin >> t;
12    while(t --) {
13        int n, m, root;
14        cin >> n >> m >> root;
15        vector adj(n + 1, vector<int>());
16        for(int i = 1; i < n; i++) {
17            int u, v;
18            cin >> u >> v;
19            adj[u].push_back(v);
20            adj[v].push_back(u);
21        }
22        // 构建LCA结构
23        int n1 = log2(n);
24        vector<int> deep(n + 1);
25        vector st(n + 1, vector<int>(n1 + 1));
26        auto dfs = [&](auto && f, int u, int last, int d) -> void{
27            deep[u] = d;
28            st[u][0] = last;
29            for(int i = 1; i <= n1; i++) {
30                st[u][i] = st[st[u][i - 1]][i - 1];
31            }
32            for(auto v : adj[u]) {
33                if(v != last) {
34                    f(f, v, u, d + 1);
35                }
36            }
37        };
38        dfs(dfs, root, root, 1);
39
40        for(int i = 0; i < m; i++) {
41            int u, v;
42            cin >> u >> v;
43            // 查询公共祖先部分部分
44            // 先统一高度
45            if(deep[u] < deep[v]) {
46                swap(u, v);
47            }
48            for(int j = n1; j >= 0; j--) {
49                if(deep[st[u][j]] >= deep[v]) {
50                    u = st[u][j];
51                }
52            }
53            // 特判
54            if(u == v) {
55                cout << v << "\n";
56            } else {
57                for(int j = n1; j >= 0; j--) {

```

```

58             if(st[u][j] != st[v][j]){
59                 u = st[u][j];
60                 v = st[v][j];
61             }
62         }
63         cout << st[u][0] << "\n";
64     }
65     //
66 }
67 }
68 return 0;
69 }

```

Tarjan

利用并查集来一次dfs就构建好lca信息

```

1 // https://www.luogu.com.cn/problem/P3379
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main(){
8     ios::sync_with_stdio(0);
9     cin.tie(0), cout.tie(0);
10    int t = 1;
11    // cin >> t;
12    while(t --){
13        int n, m, root;
14        cin >> n >> m >> root;
15        vector adj(n + 1, vector<int>());
16        for(int i = 1, u, v; i < n; i ++){
17            cin >> u >> v;
18            adj[u].push_back(v);
19            adj[v].push_back(u);
20        }
21        vector adj2(m + 1, vector<pair<int, int>>());
22        for(int i = 1, u, v; i <= m; i ++){
23            cin >> u >> v;
24            adj2[u].push_back({v, i});
25            adj2[v].push_back({u, i});
26        }
27        vector<int> fa(n + 1), ans(m + 1);
28        vector<bool> vi(n + 1, false);
29        for(int i = 0; i <= n; i ++){
30            fa[i] = i;
31        }
32        auto find = [&](auto && f, int x) -> int {
33            if(x != fa[x]){
34                fa[x] = f(f, fa[x]);
35            }
36            return fa[x];
37        };
38        auto tarjin = [&](auto && f, int u, int last) -> void {

```

```

39     vi[u] = true;
40     for(auto v : adj[u]){
41         if(v != last){
42             f(f, v, u);
43         }
44     }
45     for(auto [v, n1] : adj2[u]){
46         if(vi[v]){
47             ans[n1] = find(find, v);
48         }
49     }
50     fa[u] = last;
51 };
52 tarjin(tarjin, root, -1);
53 for(int i = 1; i <= m; i ++){
54     cout << ans[i] << "\n";
55 }
56 }
57 return 0;
58 }
```

练习题

树的重心

如果在树中选择某个节点并删除，这棵树将分为若干棵子树，使所有子树最大值最小的节点被称为整个树的重心。

性质：

- 树的重心如果不唯一，则至多有两个，且这两个重心相邻。
- 以树的重心为根时，所有子树的大小都不超过整棵树大小的一半。
- 树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么到它们的距离和一样。
- 把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。
- 在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n;
4 const int N = 5e4 + 1;
5 int head[N], next1[N << 1], to[N << 1];
6 int size1[N], max1[N];
7 int cnt1 = 1;
8 void add (int u, int v){
9     next1[cnt1] = head[u];
10    to[cnt1] = v;
11    head[u] = cnt1++;
12 };
13 void dfs(int u, int last){
14     size1[u] = 1;
15     max1[u] = 0;
```

```

16     for(int ei = head[u], v; ei; ei = next1[ei]){
17         v = to[ei];
18         if(v != last){
19             dfs(v, u);
20             max1[u] = max(max1[u], size1[v]);
21             size1[u] += size1[v];
22         }
23     }
24     max1[u] = max(max1[u], n - size1[u]);
25 }
26 signed main(){
27     ios::sync_with_stdio(0);
28     cin.tie(0);
29     int t = 1;
30     // cin >> t;
31     while(t --){
32         cin >> n;
33         fill(head, head + 1 + n, 0);
34         for(int i = 0; i < n - 1; i ++){
35             int u, v;
36             cin >> u >> v;
37             add(u, v);
38             add(v, u);
39         }
40         dfs(1, -1);
41         int ans[2];
42         int m = 0;
43         for(int i = 1; i <= n; i ++){
44             // 依照树的重心的定义
45             if(max1[i] <= n >> 1){
46                 ans[m ++] = i;
47             }
48         }
49         for(int i = 0; i < m; i ++){
50             cout << ans[i] << " ";
51         }
52     }
53     return 0;
54 }
```

当节点有值时，重心的依据是节点权而不是边权。

树的直径

树上任意两点之间最常的简单路径为树的直径，一棵树可以有多条直径，长度相等

两次DFS

适用与无负边树

首先从任意节点 y 开始进行第一次 DFS，到达距离其最远的节点，记为 z ，然后再从 z 开始做第二次 DFS，到达距离 z 最远的节点，记为 z' ，则 $\delta(z, z')$ 即为树的直径。

显然，如果第一次 DFS 到达的节点 z 是直径的一端，那么第二次 DFS 到达的节点 z' 一定是直径的一端。我们只需证明在任意情况下， z 必为直径的一端。

定理：在一棵树上，从任意节点 y 开始进行一次 DFS，到达的距离其最远的节点 z 必为直径的一端。

```
1 int n;
2 cin >> n;
3 vector<int> head(n + 1, 0), next(n << 1 | 1), to(n << 1 | 1), weight(n << 1 | 1);
4 int cnt1 = 1;
5 // 两次DFS
6 vector<int> dis(n + 1, 0), fa(n + 1), len(n + 1, 0);
7 int st, en;
8 auto dfs = [&] (auto && f, int u, int last, int val) -> void {
9     dis[u] = val;
10    fa[u] = last;
11    for(int ei = head[u], v, w; ei; ei = next[ei]){
12        v = to[ei];
13        w = weight[ei];
14        if(v != last){
15            len[v] = w;
16            f(f, v, u, val + w);
17        }
18    }
19};
20 dfs(dfs, 1, -1, 0);
21 for(int i = 1, max1 = 0; i <= n; i ++){
22     if(max1 < dis[i]){
23         max1 = dis[i];
24         dis[i] = len[i] = 0;
25         st = i;
26     }
27 }
28 dfs(dfs, st, -1, 0);
29 for(int i = 1, max1 = 0; i <= n; i ++){
30     if(max1 < dis[i]){
31         max1 = dis[i];
32         en = i;
33     }
34 }
35 int sum1 = dis[en]; // 直径
```

树形DP

可用于有负边树

我们记录当1为树的根时，每个节点作为子树的根向下，所能延伸的最长路径长度 d_1 与次长路径（与最长路径无公共边） d_2 ，那么直径就是对于每一个点，该点的 $d_1 + d_2$ 能取到的最大值

```
1 vector<int> head1(n + 1, 0), next1(n << 1 | 1), to1(n << 1 | 1), weight(n << 1 | 1);
2 cnt1 = 1;
3 vector<int> ans(n + 1), v1(n + 1);
4 auto dfs3 = [&] (auto && f, int u, int last) -> void {
```

```

5     ans[u] = 0, v1[u] = 0;
6     for(int ei = head1[u], v, w; ei; ei = next1[ei]){
7         v = to1[ei], w = weight[ei];
8         if(v != last){
9             f(f, v, u);
10            ans[u] = max(ans[u], v1[u] + v1[v] + w);
11            v1[u] = max(v1[u], v1[v] + w);
12        }
13    }
14 }
15 dfs3(dfs3, 1, 0);
16 int max2 = 0;
17 for(int i = 1; i <= n; i ++){
18     max2 = max(max2, ans[i]);
19 }

```

树上差分

在树上对多条路径上的点或者边进行修改时，不能直接便利操作，而是利用差分思想积累信息，最后累加

- 对于点，路径两端++,两端LCA和LCA的父亲--
- 对于边，路径两端++,其值可理解为该点到其父亲的边，两端LCA-=2

点差分

```

1 // https://www.luogu.com.cn/problem/P3128
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 constexpr int N = 5e4, K = 1e5;
8 vector<int> head(N + 1, 0), next1(N << 1 | 1), to(N << 1 | 1);
9 vector<int> head2(K + 1, 0), next2(K << 1 | 1), to2(K << 1 | 1),
10 vector<int> fa_lca(K + 1), lca(K + 1), has(N + 1), fa(N + 1);
11 int cnt1 = 1, cnt2 = 1;
12 signed main(){
13     cin.tie(nullptr)->iostream::sync_with_stdio(false);
14     int t = 1;
15     // cin >> t;
16     while(t --){
17         int n, q;
18         cin >> n >> q;
19         auto add = [&](int u, int v, int i, int f) -> void {
20             if(!f) {
21                 next1[cnt1] = head[u];
22                 to[cnt1] = v;
23                 head[u] = cnt1++;
24             } else {
25                 next2[cnt2] = head2[u];
26                 to2[cnt2] = v;
27             }
28         };
29         if(q == 1) {
30             add(n, q, 1, 1);
31         } else {
32             add(n, q, 1, 0);
33             add(q, n, 0, 1);
34         }
35     }
36 }

```

```

27         in[cnt2] = i;
28         head2[u] = cnt2++;
29     }
30 }
31 for(int i = 1, u, v; i < n; i++) {
32     cin >> u >> v;
33     add(u, v, 0, 0);
34     add(v, u, 0, 0);
35 }
36 iota(fa_lca.begin() + 1, fa_lca.begin() + 1 + q, 1);
37 auto find = [&](auto && f, int x) -> int {
38     if(x != fa_lca[x]) fa_lca[x] = f(f, fa_lca[x]);
39     return fa_lca[x];
40 };
41 vector<pair<int,int>> edges(q);
42 for(int i = 0, u, v; i < q; i++) {
43     cin >> u >> v;
44     edges[i] = {u, v};
45     add(u, v, i, 1);
46     add(v, u, i, 1);
47 }
48 auto tarjan = [&](auto && f, int u, int last) -> void {
49     has[u] = 1;
50     for(int ei = head[u], v; ei; ei = next1[ei]) {
51         v = to[ei];
52         if(v != last) f(f, v, u);
53     }
54     for(int ei = head2[u], v; ei; ei = next2[ei]) {
55         v = to2[ei];
56         // 注意lca的编号是in[ei]而不是v或者ei
57         if(has[v]) lca[in[ei]] = find(find, v);
58     }
59     fa[u] = last;
60     fa_lca[u] = last;
61 };
62 // 跑lca
63 tarjan(tarjan, 1, 0);
64 // 差分
65 vector<int> ans(n + 1, 0);
66 for(int i = 0; i < q; i++) {
67     ans[edges[i].fr]++, ans[edges[i].sc]++;
68     ans[lca[i]]--, ans[fa[lca[i]]]--;
69 }
70 auto dfs = [&](auto && f, int u, int last) -> void {
71     for(int ei = head[u], v; ei; ei = next1[ei]) {
72         v = to[ei];
73         if(v != last) {
74             f(f, v, u);
75             ans[u] += ans[v];
76         }
77     }
78 };
79 // 跑差分
80 dfs(dfs, 1, 0);
81 int max1 = 0;
82 for(int i = 1; i <= n; i++) {

```

```

83         // cout << i << " " << ans[i] << "\n";
84         max1 = max(max1, ans[i]);
85     }
86     cout << max1 << "\n";
87 }
88 return 0;
89 }
```

边差分

Tarjan

```

1 // https://www.luogu.com.cn/problem/P2680
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #define fr first
6 #define sc second
7
8 signed main() {
9     cin.tie(nullptr)->iostream::sync_with_stdio(false);
10    int t = 1; // 题目只有一组数据
11    while (t--) {
12        int n, m;
13        cin >> n >> m;
14
15        // 树的邻接表存储
16        vector<int> head(n + 1, 0), next1(n << 1 | 1), to(n << 1 | 1), wei(n
<< 1 | 1);
17        // 查询图 (用于 Tarjan 离线 LCA)
18        vector<int> head2(n + 1, 0), next2(m << 1 | 1), to2(m << 1 | 1), in(m
<< 1 | 1);
19
20        // 各种辅助数组
21        vector<int> len(n + 1), has(n + 1, 0), fa_lca(n + 1), lca(m), cost(m);
22        int cnt1 = 1, cnt2 = 1;
23
24        // 加边 (树边)
25        auto add = [&](int u, int v, int w) {
26            next1[cnt1] = head[u];
27            to[cnt1] = v;
28            wei[cnt1] = w;
29            head[u] = cnt1++;
30        };
31
32        // 加边 (查询边)
33        auto add2 = [&](int u, int v, int i) {
34            next2[cnt2] = head2[u];
35            to2[cnt2] = v;
36            in[cnt2] = i;
37            head2[u] = cnt2++;
38        };
39
40        // 读入树
41        for (int i = 1, u, v, w; i < n; i++) {
```

```

42         cin >> u >> v >> w;
43         add(u, v, w);
44         add(v, u, w);
45     }
46
47     int max1 = 0; // 最大路径长度
48     vector<array<int, 2>> edges(m);
49
50     // 读入运输任务
51     for (int i = 0; i < m; i++) {
52         cin >> edges[i][0] >> edges[i][1];
53         add2(edges[i][0], edges[i][1], i);
54         add2(edges[i][1], edges[i][0], i);
55     }
56
57     // 并查集 find
58     auto find = [&](auto &&f, int x) -> int {
59         if (x != fa_lca[x]) fa_lca[x] = f(f, fa_lca[x]);
60         return fa_lca[x];
61     };
62
63     iota(fa_lca.begin(), fa_lca.end(), 0); // 初始化并查集
64
65     // Tarjan 离线 LCA
66     auto tarjan = [&](auto &&f, int u, int last, int w) -> void {
67         len[u] = w; // 从根到 u 的距离
68         has[u] = 1; // 标记已访问
69
70         // 遍历树的子节点
71         for (int ei = head[u], v; ei; ei = next1[ei]) {
72             v = to[ei];
73             if (v != last) f(f, v, u, w + wei[ei]);
74         }
75
76         // 遍历与 u 有关的查询
77         for (int ei = head2[u], v; ei; ei = next2[ei]) {
78             v = to2[ei];
79             if (has[v]) {
80                 lca[in[ei]] = find(find, v); // 找到 LCA
81                 // 路径长度 = len[u] + len[v] - 2 * len[lca]
82                 cost[in[ei]] = len[v] + len[u] - 2 * len[lca[in[ei]]];
83                 max1 = max(max1, cost[in[ei]]);
84             }
85         }
86
87         fa_lca[u] = last; // 回溯并查集父节点
88     };
89
90     tarjan(tarjan, 1, 0, 0); // 从根节点 1 开始
91
92     // 二分答案
93     int l = 0, r = max1, ans = 0, mid;
94     vector<int> num(n + 1);
95     int cnt, need;
96
97     // 检查函数 (判断是否能在限制 x 下完成任务)

```

```

98     auto cf = [&] (auto &&f, int u, int last, int w) -> bool {
99         for (int ei = head[u], v; ei; ei = next1[ei]) {
100             v = to[ei];
101             if (v != last) {
102                 if (f(f, v, u, wei[ei])) return 1;
103                 num[u] += num[v];
104             }
105         }
106         return num[u] == cnt && w >= need; // 判断是否找到关键边
107     };
108
109     auto F = [&] (int x) -> bool {
110         cnt = 0;
111         need = max1 - x;
112         fill(num.begin(), num.end(), 0);
113
114         // 标记需要减少的路径
115         for (int i = 0; i < m; i++) {
116             if (cost[i] > x) {
117                 num[edges[i][0]]++;
118                 num[edges[i][1]]++;
119                 num[lca[i]] -= 2;
120                 cnt++;
121             }
122         }
123         return !cnt || cf(cf, 1, 0, 0);
124     };
125
126     while (l <= r) {
127         mid = (l + r) >> 1;
128         if (F(mid)) ans = mid, r = mid - 1;
129         else l = mid + 1;
130     }
131     cout << ans << "\n";
132 }
133
134 }
```

| ST

```

1 // https://www.luogu.com.cn/problem/P2680
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #define fr first
6 #define sc second
7
8 signed main() {
9     cin.tie(nullptr)->ios_base::sync_with_stdio(false);
10    int t = 1;
11    while (t--) {
12        int n, m;
13        cin >> n >> m;
14
15        // 树的邻接表
```

```

16     vector<int> head(n + 1, 0), next1(n << 1 | 1), to(n << 1 | 1), wei(n
17     << 1 | 1);
18     int cnt1 = 1;
19
20     auto add = [&](int u, int v, int w) -> void {
21         next1[cnt1] = head[u];
22         to[cnt1] = v;
23         wei[cnt1] = w;
24         head[u] = cnt1++;
25     };
26
27     // 读入树
28     for (int i = 1, u, v, w; i < n; i++) {
29         cin >> u >> v >> w;
30         add(u, v, w);
31         add(v, u, w);
32     }
33
34     int n1 = log2(n);
35     vector<int> deep(n + 1, 0); // 深度
36     vector st_lca(n + 1, vector<int>(n1 + 1)); // 倍增父节点表
37     vector st_len(n + 1, vector<int>(n1 + 1)); // 对应的边权和
38
39     // 预处理 DFS
40     auto dfs = [&](auto &f, int u, int last, int d, int w) -> void {
41         deep[u] = d;
42         st_lca[u][0] = last;
43         st_len[u][0] = w;
44
45         // 倍增表更新
46         for (int i = 1; i <= n1; i++) {
47             st_lca[u][i] = st_lca[st_lca[u][i - 1]][i - 1];
48             st_len[u][i] = st_len[u][i - 1] + st_len[st_lca[u][i - 1]][i - 1];
49         }
50
51         for (int ei = head[u], v; ei; ei = next1[ei]) {
52             v = to[ei];
53             if (v != last) f(f, v, u, d + 1, wei[ei]);
54         }
55     };
56
57     dfs(dfs, 1, 0, 1, 0);
58
59     vector<array<int, 4>> edges(m);
60     int max1 = 0;
61
62     // LCA 查询
63     auto lca = [&](int u, int v, int il) -> void {
64         if (deep[u] < deep[v]) swap(u, v);
65         int len = 0;
66
67         // 先让 u 和 v 到同一深度
68         for (int i = n1; i >= 0; i--)
69             if (deep[st_lca[u][i]] >= deep[v])
70                 len += st_len[u][i], u = st_lca[u][i];

```

```

70
71     if (u == v) goto t1;
72
73     // 同时向上跳
74     for (int i = n1; i >= 0; i--) {
75         if (st_lca[u][i] != st_lca[v][i]) {
76             len += st_len[u][i] + st_len[v][i];
77             u = st_lca[u][i];
78             v = st_lca[v][i];
79         }
80     }
81
82     // 最后一步
83     len += st_len[u][0] + st_len[v][0];
84     u = st_lca[u][0];
85
86     t1:;
87     max1 = max(max1, len);
88     edges[i1][2] = len;
89     edges[i1][3] = u; // 存储 LCA
90 };
91
92     // 读入任务并计算路径长度
93     for (int i = 0; i < m; i++) {
94         cin >> edges[i][0] >> edges[i][1];
95         lca(edges[i][0], edges[i][1], i);
96     }
97
98     // 二分答案
99     int l = 0, r = max1, ans = 0, mid;
100    vector<int> num(n + 1);
101    int cnt, need;
102
103    auto cf = [&](auto &&f, int u, int last, int w) -> bool {
104        for (int ei = head[u], v; ei; ei = next1[ei]) {
105            v = to[ei];
106            if (v != last) {
107                if (f(f, v, u, wei[ei])) return 1;
108                num[u] += num[v];
109            }
110        }
111        return num[u] == cnt && w >= need;
112    };
113
114    auto F = [&](int x) -> bool {
115        cnt = 0;
116        need = max1 - x;
117        fill(num.begin(), num.end(), 0);
118
119        for (int i = 0; i < m; i++) {
120            if (edges[i][2] > x) {
121                num[edges[i][0]]++;
122                num[edges[i][1]]++;
123                num[edges[i][3]] -= 2;
124                cnt++;
125            }

```

```

126         }
127         return !cnt || cf(cf, 1, 0, 0);
128     };
129
130     while (l <= r) {
131         mid = (l + r) >> 1;
132         if (F(mid)) ans = mid, r = mid - 1;
133         else l = mid + 1;
134     }
135
136     cout << ans << "\n";
137 }
138 return 0;
139 }
140

```

树上换根

在树上做DP时，很多时候与根有关，如果把根从父亲换到儿子，答案可能按照同样的计算方法通过可推导公式进行变换，通常是dfs一次先算出某一个节点的值并且同时构造出必要信息，第二次dfs分为直接计算答案和构造前后缀信息。通常的时间复杂度为O(n)

增量换根公式（轻量）

仅根据第一次dfs时构造的变量，通过第二次遍历按节点传递答案并利用变量进行计算

```

1 // https://codeforces.com/problemset/problem/1187/E
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 constexpr int N = 2e5;
6 vector<int> head(N + 1), next1(N << 1 | 1), to(N << 1 | 1), S(N + 1);
7 int cnt1;
8 signed main() {
9     cin.tie(nullptr) -> ios_base::sync_with_stdio(false);
10    int t = 1;
11    // 建图函数（链式前向星存储）
12    auto add = [&](int u, int v) {
13        next1[cnt1] = head[u];
14        to[cnt1] = v;
15        head[u] = cnt1++;
16    };
17    while (t--) {
18        int n;
19        cin >> n;
20        cnt1 = 1;
21        fill(head.begin(), head.begin() + 1 + n, 0);
22        // 输入边
23        for (int i = 1, u, v; i < n; i++) cin >> u >> v, add(u, v), add(v, u);
24        int val1 = 0; // 以 1 为根时所有子树大小之和
25        // 第一次 DFS：计算子树大小和 val1
26        auto dfs = [&](auto&& f, int u, int fa) -> void {
27            S[u] = 1;

```

```

28         for (int ei = head[u], v = to[ei]; ei; ei = next1[ei], v = to[ei])
29     {
30         if (v != fa) f(f, v, u), S[u] += S[v];
31         val1 += S[u];
32     };
33     dfs(dfs, 1, 0);
34     // 第二次 DFS: 换根, 动态维护答案
35     auto dfs2 = [&](auto&& f, int u, int fa, int val) -> int {
36         int val2 = val; // 当前答案
37         for (int ei = head[u], v = to[ei]; ei; ei = next1[ei], v = to[ei])
38     {
39         if (v != fa) {
40             // 换根公式: val + n - 2 * S[v]
41             val2 = max(val2, f(f, v, u, val + n - (S[v] << 1)));
42         }
43         return val2;
44     };
45     cout << dfs2(dfs2, 1, 0, val1) << "\n";
46 }
47 return 0;
48 }
```

前后缀合并

当每个点的 DP 来自“合并所有子树的贡献”，且这个合并满足**结合律**（如 max/sum/min 的组合），就可以用**前缀/后缀**把“除某个儿子外的其余贡献”在 O(1) 拿到，再作为“父信息”下推给儿子，实现“从父带来的贡献 + 自己子树贡献”的重组。

```

1 // https://www.luogu.com.cn/problem/P6419
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 constexpr int N = 5e5;
6 // 链式前向星存图
7 vector<int> head(N + 1), next1(N << 1 | 1), to(N << 1 | 1), wei(N << 1 | 1);
8 int cnt1;
9 // DP 数组
10 vector<int> sum_in(N + 1), sum_out(N + 1);
11 vector<int> max_in(N + 1), max_in2(N + 1), cho(N + 1), max_out(N + 1), has(N + 1);
12 signed main() {
13     cin.tie(nullptr)->iostream::sync_with_stdio(false);
14     int t = 1;
15     // cin >> t;
16     auto add = [&](int u, int v, int w) -> void {
17         next1[cnt1] = head[u];
18         to[cnt1] = v;
19         wei[cnt1] = w;
20         head[u] = cnt1++;
21     };
22     while (t--) {
23         int n, k;
```

```

24     cin >> n >> k;
25     cnt1 = 1;
26     fill(head.begin(), head.begin() + 1 + n, 0);
27     // 输入边
28     for (int i = 1, u, v, w; i < n; i++) cin >> u >> v >> w, add(u, v, w),
29     add(v, u, w);
30     // 输入特殊节点
31     for (int i = 0, x; i < k; i++) cin >> x, has[x] = 1;
32     // 第一次 DFS: 自底向上
33     auto dfs = [&](auto&& f, int u, int fa) -> void {
34         max_in[u] = max_in2[u] = sum_in[u] = 0;
35         for (int ei = head[u], v = to[ei]; ei; ei = next1[ei], v = to[ei])
36     {
37             if (v != fa) {
38                 f(f, v, u);
39                 // 累计子树代价
40                 sum_in[u] += sum_in[v] + (has[v] || sum_in[v]) * (wei[ei]
41                 << 1);
42                 // 更新最长路径
43                 int w = max_in[v] + (has[v] || sum_in[v]) * wei[ei];
44                 if (w > max_in[u]) {
45                     cho[u] = v;
46                     max_in2[u] = max_in[u];
47                     max_in[u] = w;
48                 } else if (w > max_in2[u]) max_in2[u] = w;
49             }
50         }
51     };
52     dfs(dfs, 1, 0);
53     // 第二次 DFS: 自顶向下
54     auto dfs2 = [&](auto&& f, int u, int fa) -> void {
55         for (int ei = head[u], v = to[ei]; ei; ei = next1[ei], v = to[ei])
56     {
57             if (v != fa) {
58                 max_out[v] = max_out[u];
59                 if (cho[u] == v) // 如果走的是重儿子, 用次优的 max_in2
60                     max_out[v] = max(max_out[v] + wei[ei], max_in2[u] +
61                     wei[ei]);
62                 else
63                     max_out[v] = max(max_out[v] + wei[ei], max_in[u] +
64                     wei[ei]);
65                 // sum_out 的计算
66                 sum_out[v] = (sum_in[u] - sum_in[v] + sum_out[u])
67                 + (sum_in[v] == 0 && !has[v]) * wei[ei] * 2;
68                 // 特殊情况: 该边唯一连接所有特殊点时, 去掉重复
69                 if (sum_in[u] == sum_in[v] + (wei[ei] << 1) && !sum_out[u])
70                     max_out[v] = sum_out[v] = 0;
71                 f(f, v, u);
72             }
73         }
74     };
75     dfs2(dfs2, 1, 0);
76     // 输出答案
77     for (int i = 1; i <= n; i++) {
78         cout << sum_in[i] + sum_out[i] - max(max_in[i], max_out[i]) <<
79         "\n";

```

```
73         }
74     }
75     return 0;
76 }
```

AVL树

AVL树（英语：Adelson-Velsky and Landis tree）是[计算机科学](#)中最早被发明的[自平衡二叉查找树](#)。在AVL树中，任一节点对应的两棵子树的最大高度差为1，因此它也被称为[高度平衡树](#)。查找、插入和删除在平均和最坏情况下的[时间复杂度](#)都是 $O(\log n)$ 。增加和删除元素的操作则可能需要借由一次或多次[树旋转](#)，以实现树的重新平衡。AVL树得名于它的发明者[格奥尔吉·阿杰尔松-韦利斯基](#)和[叶夫根尼·兰迪斯](#)，他们在1962年的论文《An algorithm for the organization of information》中公开了这一数据结构。

一种有序表数据结构

定义

- 平衡二叉搜索树，左右子树高度差 ≤ 1
 - 插入、删除、查询操作复杂度： $O(\log n)$
-

节点信息

- **key**: 值
 - **count**: 重复值频次
 - **height**: 子树高度
 - **size**: 子树大小（统计排名、k-th）
 - **left/right**: 左右子节点
-

核心函数

- **up(i)** : 更新 size & height
 - **maintain(i)** : 检测平衡并旋转
 - **rotateL / rotateR** : 左旋 / 右旋
-

插入

1. 按 BST 规则递归
 2. 命中相等 $\rightarrow \text{count}++$
 3. 回溯时 $\text{up} + \text{maintain}$
-

删除

1. count > 1 → count--
 2. 无子 → 直接删
 3. 单子 → 用唯一子替代
 4. 双子 → 找右子树最左节点替代，删掉它
 5. 回溯 up + maintain
-

查询

- rank(x): 小于 x 的元素数 + 1
 - index(k): 第 k 小，靠 size 判断走向
 - pre(x): 小于 x 的最大值
 - suf(x): 大于 x 的最小值
-

旋转类型

- LL: 右旋
 - RR: 左旋
 - LR: 先左旋子树，再右旋
 - RL: 先右旋子树，再左旋
-

```
1 // https://www.luogu.com.cn/problem/P3369
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 constexpr int N = 1e5;
8 // 节点信息
9 vector<int> key(N + 1), height(N + 1), left1(N + 1), right1(N + 1), size1(N +
10 1), count1(N + 1);
11 int head, cnt1; // head: 根节点编号, cnt1: 节点总数
12 signed main(){
13     cin.tie(nullptr)->iostream::sync_with_stdio(false);
14     int t = 1;
15     // 更新节点信息: 子树大小 + 高度
16     auto up = [&](int i) {
17         size1[i] = size1[left1[i]] + size1[right1[i]] + count1[i];
18         height[i] = max(height[left1[i]], height[right1[i]]) + 1;
19     };
20     // 左旋
21     auto rotate_l = [&](int i) -> int {
22         int r = right1[i];
23         right1[i] = left1[r];
24         left1[r] = i;
25         up(i);
26         up(r);
27     };
28 }
```

```

26         return r;
27     };
28     // 右旋
29     auto rotate_r = [&](int i) -> int {
30         int l = left1[i];
31         left1[i] = right1[l];
32         right1[l] = i;
33         up(i);
34         up(l);
35         return l;
36     };
37     // 保持平衡: 检查并旋转
38     auto maintain = [&](int i) -> int {
39         int ln = height[left1[i]], rn = height[right1[i]];
40         if(ln - rn > 1) { // 左子树更高
41             if(height[left1[left1[i]]] >= height[right1[left1[i]]])
42                 i = rotate_r(i); // LL
43             else
44                 left1[i] = rotate_l(left1[i]), i = rotate_r(i); // LR
45         } else if(rn - ln > 1) { // 右子树更高
46             if(height[right1[right1[i]]] >= height[left1[right1[i]]])
47                 i = rotate_l(i); // RR
48             else
49                 right1[i] = rotate_r(right1[i]), i = rotate_l(i); // RL
50         }
51         return i;
52     };
53     // 插入节点 x
54     auto add = [&](auto && f, int i, int x) -> int {
55         if(i == 0) { // 新建节点
56             key[++cnt1] = x;
57             count1[cnt1] = size1[cnt1] = height[cnt1] = 1;
58             left1[cnt1] = right1[cnt1] = 0;
59             return cnt1;
60         }
61         if(x == key[i]) count1[i]++;
62         else if(x < key[i]) left1[i] = f(f, left1[i], x);
63         else right1[i] = f(f, right1[i], x);
64         up(i);
65         return maintain(i);
66     };
67     // 删除子树中的最左节点
68     auto removel = [&](auto && f, int i, int x) -> int {
69         if(i == x) return right1[i];
70         else {
71             left1[i] = f(f, left1[i], x);
72             up(i);
73             return maintain(i);
74         }
75     };
76     // 删除值为 x 的节点 (若存在)
77     auto remove = [&](auto && f, int i, int x) -> int {
78         if(x < key[i]) left1[i] = f(f, left1[i], x);
79         else if(x > key[i]) right1[i] = f(f, right1[i], x);
80         else {
81             count1[i] --; // 多次出现, 减频次

```

```

82         if(!count1[i]) { // 频次为 0 -> 删除节点
83             if(!left1[i] && !right1[i]) return 0;           // 无子
84             else if(!left1[i] && right1[i]) i = right1[i]; // 只有右子
85             else if(left1[i] && !right1[i]) i = left1[i]; // 只有左子
86             else { // 两个子
87                 int l = right1[i];
88                 while(left1[l]) l = left1[l]; // 找右子树最左节点
89                 right1[i] = remove1(remove1, right1[i], l);
90                 right1[l] = right1[i];
91                 left1[l] = left1[i];
92                 i = l;
93             }
94         }
95     }
96     up(i);
97     return maintain(i);
98 };
99 // 统计小于 x 的元素个数
100 auto small = [&](auto && f, int i, int x) -> int {
101     if(i == 0) return 0;
102     if(key[i] >= x) return f(f, left1[i], x);
103     else return count1[i] + f(f, right1[i], x) + size1[left1[i]];
104 };
105 // 查询第 x 小的数
106 auto index = [&](auto && f, int i, int x) -> int {
107     if(size1[left1[i]] >= x) return f(f, left1[i], x);
108     else if(size1[left1[i]] + count1[i] < x)
109         return f(f, right1[i], x - size1[left1[i]] - count1[i]);
110     else return key[i];
111 };
112 // 查询前驱: 小于 x 的最大值
113 auto pre = [&](auto && f, int i, int x) -> int {
114     if(i == 0) return INT_MIN;
115     if(key[i] >= x) return f(f, left1[i], x);
116     else return max(key[i], f(f, right1[i], x));
117 };
118 // 查询后继: 大于 x 的最小值
119 auto suf = [&](auto && f, int i, int x) -> int {
120     if(i == 0) return INT_MAX;
121     if(key[i] <= x) return f(f, right1[i], x);
122     else return min(key[i], f(f, left1[i], x));
123 };
124 while(t--) {
125     int n;
126     cin >> n;
127     cnt1 = head = 0;
128     for(int i = 0, op, x; i < n; i++) {
129         cin >> op >> x;
130         if(op == 1) head = add(add, head, x);           // 插入 x
131         else if(op == 2) {                             // 删除 x
132             if(small(small, head, x) != small(small, head, x + 1))
133                 head = remove(remove, head, x);
134         }
135         else if(op == 3) cout << small(small, head, x) + 1 << "\n"; // 查询

```

排名

```

136         else if(op == 4) cout << index(index, head, x) << "\n";      // 查询
第 k 小
137         else if(op == 5) cout << pre(pre, head, x) << "\n";          // 前驱
138         else cout << suf(suf, head, x) << "\n";                      // 后继
139     }
140 }
141 return 0;
142 }
```

笛卡尔树

笛卡尔树是一种二叉树，每一个节点由一个键值二元组 构成。要求 满足二叉搜索树的性质，而 满足堆的性质。如果笛卡尔树的 键值确定，且 互不相同， 也互不相同， 那么这棵笛卡尔树的结构是唯一的

```

1 // https://ac.nowcoder.com/acm/contest/108306/G
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 constexpr int mod = 998244353;
8 constexpr int N = 1e6;
// 单调栈建树相关
9 int head, tot;
10 vector<int> L(N + 1), R(N + 1), S(N + 1), H(N + 1), st(N + 1);
11 // 组合数预处理
12 vector<int> fac(N + 1), inv(N + 1);
13 signed main(){
14     cin.tie(nullptr)->iostream::sync_with_stdio(false);
15     int t = 1;
16     cin >> t;
17     // 预处理阶乘与逆元: fac[i] = i!, inv[i] = (i!)^{-1}
18     fac[0] = 1;
19     for(int i = 1; i <= N; i++) fac[i] = fac[i - 1] * i % mod;
20     // 快速幂
21     auto power = [&](int x, int n) -> int {
22         int ans = 1;
23         while(n) {
24             if(n & 1) ans = ans * x % mod;
25             x = x * x % mod;
26             n >>= 1;
27         }
28     };
29     return ans;
30 };
31 // 费马小定理求逆阶乘
32 inv[N] = power(fac[N], mod - 2);
33 for(int i = N - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % mod;
34 while(t --){
35     int n;
36     cin >> n;
37     // 初始化左右儿子指针
38     fill(L.begin(), L.begin() + 1 + n, 0);
39     fill(R.begin(), R.begin() + 1 + n, 0);
40     vector<int> v1(n + 1);
```

```

41     int min1 = 1e18;
42     // 读入并记录全局最小值位置 (Cartesian Tree 的根)
43     for(int i = 1; i <= n; i++) {
44         cin >> v1[i];
45         if(v1[i] < min1) min1 = v1[i], head = i;
46     }
47     // 用单调递增栈构造“最小值 Cartesian Tree”
48     // 规则 (经典做法) :
49     // - 弹出所有比当前值大的栈顶;
50     // - 栈顶 (若存在) 作为当前结点的左/右关系的一端;
51     // - 维护 L、R 为每个结点的左右儿子。
52     tot = 0;
53     for(int i = 1, now; i <= n; i++) {
54         now = tot;
55         while(tot && v1[i] < v1[st[tot]]) tot--; // 保持单调 (严格递增)
56         if(tot) R[st[tot]] = i; // 栈顶的右儿子是当前 i
57         if(now > tot) L[i] = st[tot + 1]; // 被弹出的第一个成为 i 的
左儿子
58         st[++tot] = i;
59     };
60     // 组合数 C(k, n) = nCk, 这里参数名是 (up, down), 实际返回的是 C(up, down)
61     auto C = [&](int up, int down) -> int {
62         return fac[down] * inv[up] % mod * inv[down - up] % mod;
63     };
64     // DFS:
65     // - 先递归儿子, 得到子树规模 S[*]
66     // - 本结点贡献: C(d, d+S[i]-1) (等价于 C(d+S[i]-1, d))
67     // - 返回子树总答案之和
68     auto dfs = [&](auto && f, int i, int d) -> int {
69         int ans = 0;
70         if(!L[i] && !R[i]) {
71             S[i] = 1; // 叶子
72         } else {
73             if(L[i]) ans = (ans + f(f, L[i], d + 1)) % mod;
74             if(R[i]) ans = (ans + f(f, R[i], d + 1)) % mod;
75             S[i] = S[L[i]] + S[R[i]] + 1; // 子树规模
76         }
77         ans = (ans + C(d, d + S[i] - 1)) % mod; // 加上当前结点的组合贡献
78         return ans;
79     };
80     // 最后 +1 一般用于把“空集/空方案”等计入答案 (具体取决于题意)
81     cout << (dfs(dfs, head, 1) + 1) % mod << "\n";
82 }
83 return 0;
84 }
```

并查集

对数据进行分类

```

1 // 递归找祖先
2 int find(int x) {
3     if(x != fa[x]) fa[x] = find(fa[x]);
4     return fa[x];
5 }
6 void U(int x, int y) {
7     fa[find(x)] = find(y);
8     find(x);
9 }

```

最后使用前记得统一根，即便利一遍时操作find

常见二分

加速查询

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 vector<int> v1 = {0, 1, 2, 3, 4, 5};
4 // 最小大于等于
5 int F1(int x) {
6     int r = v1.size() - 1;
7     int l = 0;
8     int ans, mid;
9     while(l <= r) {
10         mid = (l + r) >> 1;
11         if(x >= v1[mid]) {
12             ans = mid;
13             l = mid - 1;
14         }
15         else r = mid - 1;
16     }
17     return r;
18 }
19 // 最小大于
20 int F2(int x) {
21     int r = v1.size() - 1;
22     int l = 0;
23     int ans, mid;
24     while(l <= r) {
25         mid = (l + r) >> 1;
26         if(x > v1[mid]) {
27             ans = mid;
28             l = mid - 1;
29         }
30         else r = mid - 1;
31     }
32     return r;
33 }
34 // 最长递增子序列
35 int F3(int x, vector<int> & v) {
36     int r = v1.size() - 1;
37     int l = 0;
38     int mid;

```

```

39     int ans = -1;
40     while(l <= r) {
41         mid = (l + r) >> 1;
42         if(x > v[mid]) l = mid + 1;
43         else{
44             r = mid - 1;
45             ans = mid;
46         }
47     }
48     return ans;
49 }
50 // 最长递减子序列
51 int F2(int x, vector<int> & v) {
52     int r = v1.size() - 1;
53     int l = 0;
54     int mid;
55     int ans = -1;
56     while(l <= r) {
57         mid = (l + r) >> 1;
58         if(x < v[mid]) l = mid + 1;
59         else{
60             r = mid - 1;
61             ans = mid;
62         }
63     }
64     return ans;
65 }

```

三分

```

1 // F视具体情况
2 ll ternary_search(ll l, ll r) {
3     while (r - l > 3) {
4         ll m1 = l + (r - l) / 3;
5         ll m2 = r - (r - l) / 3;
6         if (F(m1) <= F(m2)) {
7             r = m2;
8         } else {
9             l = m1;
10        }
11    }
12    ll best = 1e9;
13    // 最后暴力求解
14    for (ll i = l; i <= r; i++) {
15        if (F(i) < F(best)) {
16            best = i;
17        }
18    }
19    return best;
20 }

```

博奕论（sg 定理）

所有相同子游戏最终 sg 异或结果为0则先手必败，不为零则先手必胜， $_{\text{sg}}$ 为从某一节点开始，到最大限度往前便利后的第一个没有出现的自然数。

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 // 计算集合中缺失的最小非负整数 (mex函数)
4 int mex(const set<int>& s) {
5     int m = 0;
6     while (s.count(m)) m++;
7     return m;
8 }
9 int main() {
10     int n;
11     cout << "请输入石子堆的石子数量：";
12     cin >> n;
13     // 允许的操作：每次可以取 1、2 或 3 个石子
14     vector<int> moves = {1, 2, 3};
15     // 动态规划数组，sg[i] 表示当前石子数量为 i 时的 SG 值
16     vector<int> sg(n+1, 0);
17     sg[0] = 0; // 没有石子时，SG值为0
18     // 计算每个状态的 SG 值
19     for (int i = 1; i <= n; i++) {
20         set<int> nextSG;
21         // 对每一种允许的走法，计算下一状态的 SG 值
22         for (int move : moves) {
23             if (i - move >= 0) {
24                 nextSG.insert(sg[i - move]);
25             }
26         }
27         sg[i] = mex(nextSG);
28     }
29     cout << "状态 " << n << " 的 SG 值为：" << sg[n] << "\n";
30     if(sg[n] != 0)
31         cout << "先手必胜\n";
32     else
33         cout << "后手必胜\n";
34     return 0;
35 }
```

乘积最大子数组

保留乘积最小的结果以准备触底反弹

```
1 vector<int> v1;
2 int min1 = 1;
3 int max1 = 1;
4 int ans1 = INT_MIN;
5 int pre1 = 1;
6 int pre2 = 1;
7 for(int i = 0; i < v1.size(); i ++){
8     max1 = max({v1[i], pre1 * v1[i], pre2 * v1[i]});
```

```

9     min1 = min({v1[i], pre1 * v1[i], pre2 * v1[i]}); 
10    pre1 = max1;
11    pre2 = min1;
12    ans1 = max(ans1, max1);
13 }
14 cout << ans1 << "\n";

```

栈

单调栈

求某一侧第一个大于或者小于其的数

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 // 求右边第一个比其大的数
5 void solve(vector<int> & v1, int n) {
6     vector<int> ans(n + 1);
7     int size=v1.size();
8     stack<int> s1;
9     for(int i=0;i<size;i++) {
10         while(!s1.empty() && v1[i] > v1[s1.top()]) {
11             ans[s1.top()] = i+1;
12             s1.pop();
13         }
14         s1.push(i);
15     }
16     for(int i=0;i<size;i++) {
17         cout<<ans[i]<<' ';
18     }cout<<endl;
19 }
20 int main() {
21     ios::sync_with_stdio(false);
22     cin.tie(0);
23     cout.tie(0);
24     int n;
25     cin>>n;
26     vector<int> v1;
27     for(int i=0;i<n;i++) {
28         cin>>v1[i];
29     }
30     solve(v1, n);
31     return 0;
32 }

```

最大频率栈

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 class FreqStack {
4 private:
5     unordered_map<int,vector<int>> s1;

```

```
6     unordered_map<int,int> t1;
7     int max1;
8 public:
9     FreqStack() {
10         max1=0;
11     }
12     void push(int val) {
13         t1[val]++;
14         s1[t1[val]].push_back(val);
15         max1=t1[val]>max1?t1[val]:max1;
16     }
17     /*
18     int pop() 删除并返回堆栈中出现频率最高的元素。
19     如果出现频率最高的元素不只一个，则移除并返回最接近栈顶的元素。
20     */
21     int pop() {
22         int ans=s1[max1].back();
23         s1[t1[ans]--].pop_back();
24         if(t1[ans]==0){
25             t1.erase(ans);
26         }
27         if(s1[max1].empty()){
28             max1--;
29             s1.erase(max1+1);
30         }
31     }
32 };
```

堆

动态堆寻找中位数

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 constexpr int N = 1e5;
4 int n; // 操作次数
5 string op; // 输入的操作命令
6 int size1 = 0; // 当前栈的大小
7 int minsize = 0; // 小根堆中元素的数量
8 int maxsize = 0; // 大根堆中元素的数量
9 vector<int> stack1(N); // 用于模拟栈，存储所有插入的数字
10 priority_queue<int, vector<int>, greater<int>> minheap; // 小根堆，用于存储大于等于中位数的数
11 priority_queue<int> maxheap; // 大根堆，用于存储小于等于中位数的数
12 // 用于延迟删除的映射：记录需要从堆中删除但还未实际删除的数字及其次数
13 unordered_map<int, int> delayed;
14 template <typename T>
15 void prune(T &heap) {
16     // 当堆不为空时，检查堆顶元素是否在延迟删除映射中
17     while (!heap.empty()) {
18         int num = heap.top();
19         if (delayed.count(num)) { // 如果堆顶元素需要被删除
20             delayed[num]--; // 延迟删除计数减一
21             if (delayed[num] == 0)
22                 delayed.erase(num); // 如果次数为0则从映射中删除
23         }
24     }
25 }
```

```

23         heap.pop(); // 弹出堆顶元素
24     } else {
25         break; // 堆顶元素未被延迟删除，退出循环
26     }
27 }
28 }
29 void balance() {
30     // 如果大根堆的元素比小根堆多超过1个，则将大根堆的堆顶移动到小根堆中
31     if (maxsize > minsize + 1) {
32         minheap.push(maxheap.top());
33         maxheap.pop();
34         minsize++;
35         maxsize--;
36         prune(maxheap); // 清理大根堆堆顶可能存在的延迟删除元素
37     }
38     // 如果大根堆的元素数目小于小根堆，则将小根堆的堆顶移动到大根堆中
39     else if (maxsize < minsize) {
40         maxheap.push(minheap.top());
41         minheap.pop();
42         maxsize++;
43         minsize--;
44         prune(minheap); // 清理小根堆堆顶可能存在的延迟删除元素
45     }
46 }
47 void erase(int num) {
48     // 标记该数字需要延迟删除
49     delayed[num]++;
50     if (num <= maxheap.top()) {
51         maxsize--;
52         // 如果待删除数字恰好是大根堆堆顶，则立即清理
53         if (num == maxheap.top()) {
54             prune(maxheap);
55         }
56     } else {
57         minsize--;
58         // 如果待删除数字恰好是小根堆堆顶，则立即清理
59         if (num == minheap.top())
60             prune(minheap);
61     }
62     balance();
63 }
64 void solve(void) {
65     cin >> n;
66     for (int i = 0; i < n; i++) {
67         cin >> op;
68         // peek
69         if (op[1] == 'e') {
70             if (size1 == 0)
71                 cout << "Invalid" << "\n";
72             else
73                 // 中位数为大根堆的堆顶
74                 cout << maxheap.top() << "\n";
75         }
76         // pop
77         else if (op[1] == 'o') {
78             if (size1 == 0)

```

```

79         cout << "Invalid" << "\n";
80     else {
81         // 输出栈顶元素
82         cout << stack1[size1 - 1] << "\n";
83         // 删除栈顶元素，并在堆中延迟删除对应数字
84         erase(stack1[--size1]);
85     }
86 }
87 // push
88 else {
89     cin >> stack1[size1++];
90     // 如果大根堆为空，则直接插入到大根堆中
91     if (maxsize == 0) {
92         maxheap.push(stack1[size1 - 1]);
93         maxsize++;
94         continue;
95     }
96     // 根据新插入数字与当前中位数比较，决定放入哪个堆
97     if (stack1[size1 - 1] > maxheap.top()) {
98         minheap.push(stack1[size1 - 1]);
99         minsize++;
100    } else {
101        maxheap.push(stack1[size1 - 1]);
102        maxsize++;
103    }
104    // 保持两个堆的平衡
105    balance();
106 }
107 }
108 }
109 int main() {
110     cin.tie(0) -> ios::sync_with_stdio(0);
111     int t;
112     while (t--) {
113         solve();
114     }
115     return 0;
116 }
```

大小根堆

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // 向堆中插入新元素，使其满足大顶堆性质
4 void heapInsert(int arr[], int i) {
5     while(arr[i] > arr[(i-1)/2]) {
6         swap(arr[i], arr[(i-1)/2]);
7         i = (i-1)/2;
8     }
9 }
10 // 堆化：从下标 i 开始向下调整，使子树满足大顶堆性质，size 为堆的大小
11 void heapify(int arr[], int i, int size) {
12     int l = i * 2 + 1;
13     int maxIdx; // 保存左右孩子中较大值的下标
14     int idx; // 保存父节点与较大孩子中较大者的下标
15     while(l < size) {
```

```

16     // 在左右孩子中选择较大者
17     maxIdx = (l + 1 < size && arr[l+1] > arr[l]) ? l + 1 : l;
18     // 选择父节点与较大孩子中较大者
19     idx = (arr[i] > arr[maxIdx]) ? i : maxIdx;
20     if(idx == i) break;
21     else {
22         swap(arr[i], arr[idx]);
23         i = idx;
24         l = i * 2 + 1;
25     }
26 }
27 }
28 int heapPop(int arr[], int &size) {
29     if(size <= 0) {
30         return -1;
31     }
32     int top = arr[0];           // 保存堆顶元素
33     swap(arr[0], arr[size - 1]); // 将堆顶与最后一个元素交换
34     size--;                   // 删除最后一个元素
35     heapify(arr, 0, size);    // 从顶端重新调整堆
36     return top;
37 }

```

扫描线

利用扫描思想平移某一个轴时，加工另一个轴的信息来一边扫过去就能解决问题

```

1 // https://www.luogu.com.cn/problem/P5490
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main(){
8     cin.tie(0) -> ios::sync_with_stdio(0);
9     int t = 1;
10    // cin >> t;
11    while(t --){
12        int n;
13        cin >> n;
14        vector<int> v2 = {-2};
15        vector<array<int, 4>> v1;
16        for(int i = 0; i < n; i ++){
17            int x1, y1, x2, y2;
18            cin >> x1 >> y1 >> x2 >> y2;
19            v2.push_back(y1);
20            v2.push_back(y2);
21            v1.push_back({x1, y1, y2, 1});
22            v1.push_back({x2, y1, y2, -1});
23        }
24        sort(v2.begin() + 1, v2.end());
25        int m = 1;
26        for(int i = 2; i < v2.size(); i ++){
27            if(v2[i] != v2[m]){
28                v2[++m] = v2[i];
29            }
30        }
31    }
32 }

```

```

29         }
30     }
31     v2.push_back(v2[m]);
32     auto find = [&](int x) -> int {
33         int l = 1, r = m;
34         int mid, ans = -1;
35         while(l <= r) {
36             mid = (l + r) >> 1;
37             if(v2[mid] >= x) {
38                 ans = mid;
39                 r = mid - 1;
40             } else {
41                 l = mid + 1;
42             }
43         }
44         return ans;
45     };
46     vector<int> sum1(m << 4), cnt1(m << 4), cover(m << 4);
47     auto build = [&](auto && f, int i, int l, int r) -> void {
48         if(l < r) {
49             int mid = (l + r) >> 1;
50             f(f, i << 1, l, mid);
51             f(f, i << 1 | 1, mid + 1, r);
52         }
53         cover[i] = v2[r + 1] - v2[l];
54         sum1[i] = cnt1[i] = 0;
55     };
56     build(build, 1, 1, m);
57     auto up = [&](int i) {
58         if(cnt1[i] > 0) {
59             sum1[i] = cover[i];
60         } else {
61             sum1[i] = sum1[i << 1] + sum1[i << 1 | 1];
62         }
63     };
64     auto set = [&](auto && f, int i, int bl, int br, int val, int l, int r)
-> void {
65         if(bl <= l && br >= r) {
66             cnt1[i] += val;
67         } else {
68             int mid = (l + r) >> 1;
69             if(bl <= mid) {
70                 f(f, i << 1, bl, br, val, l, mid);
71             }
72             if(br > mid) {
73                 f(f, i << 1 | 1, bl, br, val, mid + 1, r);
74             }
75         }
76         up(i);
77     };
78     sort(v1.begin(), v1.end());
79     int last = 0, ans = 0;
80     for(int i = 0; i < n << 1; i++) {
81         int now = v1[i][0];
82         ans += sum1[1] * (now - last);
83         // cout << sum1[1] << " " << now << " " << last << "\n";

```

```

84         last = now;
85         set(set, 1, find(v1[i][1]), find(v1[i][2]) - 1, v1[i][3], 1, m);
86     }
87     cout << ans << "\n";
88 }
89 return 0;
90 }
```

排序

归并分治

分而治之

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define M 500
4 int a1[M];
5 int temp[M];
6 void merge(int l,int m,int r){
7     int i=l;
8     int j=m+1;
9     int mark=l;
10    while(i<=m && j<=r) a1[mark++]=temp[i]<temp[j]?temp[i++]:temp[j++];
11    while(i<=m) a1[mark++]=temp[i++];
12    while(j<=r) a1[mark++]=temp[j++];
13    for(int k=l;k<=r;k++) temp[k]=a1[k];
14 }
15 void guibing(int l,int r){
16     if(l>=r) return ;
17     int m=(l+r)/2;
18     guibing(l,m);
19     guibing(m+1,r);
20     merge(l,m,r);
21 }
22 int main(){
23     int n;
24     cin>>n;
25     for(int i=0;i<n;i++){
26         cin>>a1[i];
27         temp[i]=a1[i];
28     }
29     guibing(0,n-1);
30     for(int i=0;i<n;i++) cout<<a1[i]<<" ";
31     return 0;
32 }
```

随机快排 & 选择第 k 小大的数

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define M 500
4 // 经典
```

```

5 int partition1(int a1[],int l,int mid,int r){
6     int i=l;
7     int j=r;
8     int temp=a1[mid];
9     swap(a1[mid],a1[l]);
10    while(i<j){
11        while(j>i && a1[j]>temp) {
12            j--;
13        }
14        a1[i]=a1[j];
15        while(i<j && a1[i]<=temp) {
16            i++;
17        }
18        a1[j]=a1[i];
19    }
20    a1[i]=temp;
21    return i;
22 }
23 void qs1(int a1[],int l,int r){
24     if(l>=r) return ;
25     int x=l+(rand()% (r-l+1));
26     int mid=partition1(a1,l,x,r);
27     qs1(a1,l,mid-1);
28     qs1(a1,mid+1,r);
29 }
30 // 优化
31 int first;
32 int second;
33 void partition2(int a1[],int l,int mid,int r){
34     first=l;
35     second=r;
36     int temp=a1[mid];
37     int i=l;
38     while(i<=second) {
39         if(a1[i]==temp) {
40             i++;
41         } else if(a1[i]<temp) {
42             swap(a1[first++],a1[i++]);
43         } else{
44             swap(a1[i],a1[second--]);
45         }
46     }
47 }
48 void qs2(int a1[],int l,int r){
49     if(l>=r) return ;
50     int x=l+(rand()% (r-l+1));
51     partition2(a1,l,x,r);
52     int left=first;
53     int right=second;
54     qs2(a1,l,left-1);
55     qs2(a1,right+1,r);
56 }
57 // 随机选择第 tar 小的元素 (0-based)
58 // 要求第k大的元素等同于求n - k小的元素
59 int randomizedSelect(int a1[], int k, int n) {
60     int l = 0, r = n - 1;

```

```

61     while (l <= r) {
62         int x = l + rand() % (r - l + 1); // 随机选取枢纽值
63         partition2(a1, l, x, r); // 三路划分，结果通过 first 和 second 表示
64         if (k < first) {
65             r = first - 1; // 目标在左侧
66         } else if (k > second) {
67             l = second + 1; // 目标在右侧
68         } else {
69             return a1[k]; // 命中：tar 位于等于区域内
70         }
71     }
72     // 若未找到（理论上不会出现），返回 -1 表示错误
73     return -1;
74 }
```

阶乘与逆元

乘以逆元等于除以某一个数再取模>

| 1!,2!,3!...n!的逆元

```

1 #define int long long
2 constexpr int mod = 998244353;
3 auto power = [&](int x, int n) -> int {
4     int ans = 1;
5     while(n) {
6         if(n & 1) {
7             ans = ans * x % mod;
8         }
9         x = x * x % mod;
10        n >= 1;
11    }
12    return ans;
13};
14 vector<int> fac(n + 1); // 阶乘
15 fac[0] = 1;
16 for(int i = 1; i <= n; i++) {
17     fac[i] = fac[i - 1] * i % mod;
18 }
19 vector<int> inv(n + 1); // 逆元
20 inv[n] = power(fac[n], mod - 2);
21 for(int i = n - 1; i >= 0; i--) {
22     inv[i] = inv[i + 1] * (i + 1) % mod;
23 }
24 auto C = [&](int n, int m) -> int { // 组合数
25     if(m > n) {
26         return 0;
27     }
28     return fac[n] * inv[m] % mod * inv[n - m] % mod;
29 };
```

线性递推逆元

1,2,3....n的逆元

```
1 // p 必须为质数
2 inv2[1] = 1;
3 for(int i = 2; i <= n; i++) {
4     inv2[i] = (p - p / i) * inv2[p % i] % p;
5 }
```

矩阵快速幂

加速矩阵计算

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 class Solution {
4 public:
5     vector<vector<long long>> multiply(const vector<vector<long long>> &a, const
6     vector<vector<long long>> &b) {
7         int n = a.size(), p = a[0].size(), m = b[0].size();
8         vector<vector<long long>> ans(n, vector<long long>(m, 0));
9         for (int i = 0; i < n; i++) {
10             for (int j = 0; j < m; j++) {
11                 for (int k = 0; k < p; k++) {
12                     ans[i][j] += a[i][k] * b[k][j];
13                 }
14             }
15         }
16         return ans;
17     }
18     vector<vector<long long>> power(vector<vector<long long>> v, int cnt) {
19         int n = v.size();
20         vector<vector<long long>> ans(n, vector<long long>(n, 0));
21         for (int i = 0; i < n; i++) {
22             ans[i][i] = 1;
23         }
24         while (cnt) {
25             if (cnt & 1) {
26                 ans = multiply(ans, v);
27             }
28             v = multiply(v, v);
29             cnt >= 1;
30         }
31         return ans;
32     }
33     int tribonacci(int n) {
34         if(n == 0) return 0;
35         if(n == 1 || n == 2) return 1;
36         vector<vector<long long>> start = {{1, 1, 0}};
37         vector<vector<long long>> base = {{1, 1, 0}, {1, 0, 1}, {1, 0, 0}};
38         return multiply(start, power(base, n - 2))[0][0];
39     }
40 }
```

```
39 };
```

扩展欧几里得算法

得到某一个 $ax + by = d$ 的各种参数, d 为最大公约数, x 和 y 为贝祖系数 (裴蜀定理的算法实现)

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int x, y, px, py;
4 int d;
5 void gcd1(int a, int b) {
6     if(!b) {
7         x = 1;
8         y = 0;
9         d = a;
10    }else{
11        gcd1(b, a % b);
12        px = x;
13        py = y;
14        x = py;
15        y = px - py * (a / b);
16    }
17 }
18 int main() {
19     ios::sync_with_stdio(false); cin.tie(nullptr);
20     int a, b;
21     cin >> a >> b;
22     gcd1(a, b);
23     cout << (x % b + b) % b << "\n";
24     return 0;
25 }
26 }
```

GCD

一个数组的GCD前缀不同种类是 \log 级别的, 可通过此题进行进一步了解和体会[2024ICPC Kunming](#).

对任意 $a \geq b$, 有 $\gcd(a, b) = \gcd(a - b, b)$;

GCD 本质

求 $\gcd(a^b, c^d)$.

```
1 // https://ac.nowcoder.com/acm/contest/108304/J
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 constexpr int mod = 998244353;
8 signed main() {
```

```

9  cin.tie(nullptr) ->ios_base::sync_with_stdio(false);
10 int t = 1;
11 cin >> t;
12 while(t --) {
13     int a, b, c, d;
14     cin >> a >> b >> c >> d;
15     auto power = [&](int x, int n) -> int {
16         int ans = 1;
17         x %= mod;
18         while(n) {
19             if(n & 1) ans = ans * x % mod;
20             x = x * x % mod;
21             n >= 1;
22         }
23         return ans;
24     };
25     int ans = 1;
26     while(1) {
27         if(b > d) swap(a, c), swap(b, d);
28         int temp = gcd(a, c);
29         ans = ans * power(temp, b) % mod;
30         a = a / temp;
31         //      c = temp;
32         d -= b;
33         if(gcd(a, c) == 1) break;
34     }
35     cout << ans << "\n";
36 }
37 return 0;
38 }
```

Stein GCD算法

Stein算法是一种计算两个非负整数的最大公约数的算法。它比传统的欧几里得算法在某些情况下更高效,

因为它避免了使用除法操作。该算法利用了以下几条规则:

$\text{GCD}(0, b) = b$: 如果一个数是0, 另一个数就是最大公约数。

$\text{GCD}(a, 0) = a$: 同上, 如果一个数是0, 另一个数就是最大公约数。

如果a和b都是偶数, 那么 $\text{GCD}(a, b) = 2 * \text{GCD}(a/2, b/2)$ 。

如果a是偶数且b是奇数, 那么 $\text{GCD}(a, b) = \text{GCD}(a/2, b)$ 。

如果a是奇数且b是偶数, 那么 $\text{GCD}(a, b) = \text{GCD}(a, b/2)$ 。

如果a和b都是奇数, 并且 $a \geq b$, 那么 $\text{GCD}(a, b) = \text{GCD}((a - b)/2, b)$ 。

如果a和b都是奇数, 并且 $a < b$, 那么 $\text{GCD}(a, b) = \text{GCD}((b - a)/2, a)$ 。

通过不断应用这些规则, 最终可以将两个数变为0, 从而找到它们的最大公约数。

```

1 unsigned int steinGCD(unsigned int a, unsigned int b) {
2     if (a == 0) return b;
3     if (b == 0) return a;
4     // 找到a和b的共同因子2的数量
5     unsigned int shift = 0;
6     while (((a | b) & 1) == 0) {
7         a >>= 1;
8         b >>= 1;
9         shift++;
```

```

10 }
11 // 使a成为奇数
12 while ((a & 1) == 0) {
13     a >>= 1;
14 }
15 do {
16     // 使b成为奇数
17     while ((b & 1) == 0) {
18         b >>= 1;
19     }
20     // 保证a <= b
21     if (a > b) {
22         swap(a, b);
23     }
24     // b - a是偶数，所以我们可以继续右移
25     b = (b - a);
26 } while (b != 0);
27 // 恢复共同因子2的数量
28 return a << shift;
29 }

```

Least Recently Used (LRU) 算法

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define M 1010
5 // Least Recently Used (LRU) 缓存实现
6 class LRU {
7 public:
8     // 定义双向链表节点
9     class DoubleNode {
10    public:
11         int key;      // 节点对应的 key
12         int val;      // 节点对应的 value
13         DoubleNode* next; // 后继节点指针
14         DoubleNode* last; // 前驱节点指针
15         DoubleNode(int k, int v): key(k), val(v), last(nullptr), next(nullptr)
16     {} 
17 };
18     // 定义双向链表，用于维护访问顺序
19     class DoubleList {
20     private:
21         DoubleNode* head; // 链表头部（最久未使用）
22         DoubleNode* tail; // 链表尾部（最近使用）
23     public:
24         DoubleList(): head(nullptr), tail(nullptr) {}
25         // 在链表尾部添加新节点（最新使用的节点）
26         void addNode(DoubleNode* newNode) {
27             if (!newNode) return;
28             if (!head) {
29                 head = tail = newNode;
30             } else {
31                 tail->next = newNode;
32                 newNode->last = tail;
33                 tail = newNode;
34             }
35         }
36     };
37 };
38 
```

```

33         }
34     }
35     // 将已有节点移动到链表尾部（表示最近使用）
36     void moveNodeToTail(DoubleNode* node) {
37         if (tail == node) return; // 节点已在尾部，无需移动
38         if (head == node) { // 节点位于头部
39             head = node->next;
40             if (head) head->last = nullptr;
41         } else { // 节点在中间
42             node->last->next = node->next;
43             if (node->next)
44                 node->next->last = node->last;
45         }
46         // 将节点移动到尾部
47         node->last = tail;
48         node->next = nullptr;
49         if (tail) tail->next = node;
50         tail = node;
51     }
52     // 移除链表头部节点（最久未使用的节点）并返回该节点
53     DoubleNode* removeHead() {
54         if (!head) return nullptr;
55         DoubleNode* ans = head;
56         if (head == tail) {
57             head = tail = nullptr;
58         } else {
59             head = head->next;
60             if (head) head->last = nullptr;
61         }
62         return ans;
63     }
64     // 辅助函数：清空链表，释放所有节点
65     void clear() {
66         DoubleNode* cur = head;
67         while (cur) {
68             DoubleNode* next = cur->next;
69             delete cur;
70             cur = next;
71         }
72         head = tail = nullptr;
73     }
74     // 提供链表头部访问，用于析构时遍历
75     DoubleNode* getHead() {
76         return head;
77     }
78 };
79 private:
80     unordered_map<int, DoubleNode*> keyNodeMap; // 快速定位 key 对应的节点
81     DoubleList nodeList; // 维护节点顺序的双向链表
82     int cap; // 缓存容量
83 public:
84     // 构造函数
85     LRU(int c) : cap(c) {}
86     // 析构函数，释放所有申请的内存
87     ~LRU() {
88         // 遍历双向链表释放节点

```

```
89     DoubleNode* cur = nodeList.getHead();
90     while(cur) {
91         DoubleNode* next = cur->next;
92         delete cur;
93         cur = next;
94     }
95     keyNodeMap.clear();
96 }
97 /**
98 * @brief 获取 key 对应的值，并将该节点移到尾部表示最近使用
99 *
100 * @param key 缓存中的键
101 * @return int 若 key 存在则返回对应值，否则返回 -1
102 */
103 int get(int key) {
104     if (keyNodeMap.count(key)) {
105         DoubleNode* ans = keyNodeMap[key];
106         nodeList.moveNodeToTail(ans); // 更新最近使用
107         return ans->val;
108     }
109     return -1;
110 }
111 /**
112 * @brief 向缓存中插入或更新一个 key-value 对
113 *
114 * 若 key 存在，则更新其值并将节点移到尾部；
115 * 若 key 不存在，则插入新节点，若缓存已满，则移除最久未使用的节点。
116 *
117 * @param key 缓存中的键
118 * @param val1 缓存中的值
119 */
120 void put(int key, int val1) {
121     if (keyNodeMap.count(key)) {
122         // key 存在，更新值并移到尾部
123         DoubleNode* node = keyNodeMap[key];
124         node->val = val1;
125         nodeList.moveNodeToTail(node);
126     } else {
127         // 如果缓存已满，删除最久未使用的节点
128         if (keyNodeMap.size() == cap) {
129             DoubleNode* removed = nodeList.removeHead();
130             if(removed) {
131                 keyNodeMap.erase(removed->key);
132                 delete removed; // 释放内存
133             }
134         }
135         // 插入新节点
136         DoubleNode* newNode = new DoubleNode(key, val1);
137         keyNodeMap[key] = newNode;
138         nodeList.addNode(newNode);
139     }
140 }
141 };
142 int main() {
143     ios::sync_with_stdio(false);
144     cin.tie(0);
```

```

145 // 测试用例：创建容量为3的LRU缓存
146 LRU cache(3);
147 // 插入数据
148 cache.put(1, 100);
149 cache.put(2, 200);
150 cache.put(3, 300);
151 // 访问 key 1，将其更新为最近使用
152 cout << "get(1): " << cache.get(1) << "\n"; // 输出 100
153 // 插入新数据，当缓存满时会淘汰最久未使用的 key (此时 key 2 被淘汰)
154 cache.put(4, 400);
155 // 尝试获取被淘汰的 key 2
156 cout << "get(2): " << cache.get(2) << "\n"; // 输出 -1，表示不存在
157 // 继续获取其他 key 的值
158 cout << "get(3): " << cache.get(3) << "\n"; // 输出 300
159 cout << "get(4): " << cache.get(4) << "\n"; // 输出 400
160 return 0;
161 }

```

二进制位

位图

判断某一个数是否存在

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 void add(int arr[],int num){
4     arr[num/32] |= 1<<(num%32);
5 }
6 void remove(int arr[],int num){
7     arr[num/32] &= ~(1<<(num%32));
8 }
9 void reverse(int arr[],int num){
10    arr[num/32] ^= 1<<(num%32);
11 }
12 bool contains(int arr[],int num){
13     return arr[num/32] & (1<<(num%32));
14 }
15 int main(){
16     int n;
17     cin>>n;
18     int set[(n + 32 - 1) / 32]; // 表示存储n个数需要的大小
19     return 0;
20 }

```

位运算实现四则运算

加速运算

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int add(int n, int m) {

```

```

4     int ans = n;
5     while(m != 0) {
6         // 无进位相加
7         ans = n ^ m;
8         // 计算进位，注意进位要左移一位
9         m = (n & m) << 1;
10        n = ans;
11    }
12    return ans;
13}
14 int neg(int n) {
15     // 对 n 取反后加 1 得到 -n
16     return add(~n, 1);
17}
18 int minus1(int n, int m) {
19     return add(n, neg(m));
20}
21 int mul(int n, int m) {
22     int ans = 0;
23     while(m != 0) {
24         if((m & 1) == 1) {
25             ans = add(ans, n);
26         }
27         n <= 1;
28         m >= 1;
29     }
30     return ans;
31}
32 int div1(int n, int m) {
33     // 转为正数处理，保存符号信息
34     int a = n < 0 ? neg(n) : n;
35     int b = m < 0 ? neg(m) : m;
36     int ans = 0;
37     // 从最高位开始遍历
38     for(int i = 30; i >= 0; i = minus1(i, 1)) {
39         // 判断 a 右移 i 位后是否大于等于 b
40         if((a >> i) >= b) {
41             // 减去 b 左移 i 位
42             a = minus1(a, (b << i));
43             // 将结果对应位设为1
44             ans |= (1 << i);
45         }
46     }
47     // 如果 n 与 m 符号不同，则结果取负
48     return (n < 0) ^ (m < 0) ? neg(ans) : ans;
49}
50 // 完整版
51 int div2(int a, int b) {
52     if(a == INT_MIN && b == INT_MIN) return 1;
53     if(a != INT_MIN && b != INT_MIN) return div1(a, b);
54     if(b == INT_MIN) return 0; // 除数为 INT_MIN，其它数除以它的商为0
55     if(b == neg(1)) return INT_MAX; // 防止 INT_MIN / -1 溢出
56     // 近似处理：调整 a 的值
57     a = add(a, b > 0 ? b : neg(b));
58     int ans = div1(a, b);
59     // 根据除数的正负决定偏移

```

```

60     int offset = b > 0 ? neg(1) : 1;
61     return add(ans, offset);
62 }

```

位运算骚操作

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //判断一个数是不是2的幂
4 int jurge1(int n){
5     return n>0 && n == (n&(-n));
6 }
7 //判断一个数是不是3的幂
8 #define M 1162261467
9 int jurge2(int n){
10    return n>0 && M%n==0;
11 }
12 //求大于等于n的最小2次幂
13 int F1(int n){
14     if(n<=0) return 1;
15     n--;
16     n |= n>>1;
17     n |= n>>2;
18     n |= n>>4;
19     n |= n>>8;
20     n |= n>>16;
21     return n+1;
22 }
23 //逆序二进制状态
24 int F2(int n){
25     n= ((n&0xaaaaaaaa)>>1) | ((n&0x55555555)<<1);
26     n= ((n&0xcccccccc)>>2) | ((n&0x33333333)<<2);
27     n= ((n&0xf0f0f0f0)>>4) | ((n&0x0f0f0f0f)<<4);
28     n= ((n&0xff00ff00)>>8) | ((n&0x00ff00ff)<<8);
29     n = (n>>16) | (n<<16);
30     return n;
31 }
32 //返回n的二进制中有几个1
33 int F3(int n){
34     n = (n&0x55555555)+((n>>1)&0x55555555);
35     n = (n&0x33333333)+((n>>2)&0x33333333);
36     n = (n&0x0f0f0f0f)+((n>>4)&0x0f0f0f0f);
37     n = (n&0x00ff00ff)+((n>>8)&0x00ff00ff);
38     n = (n&0x0000ffff)+((n>>16)&0x0000ffff);
39     return n;
40 }
41 //求x到y的所有数&的结果
42 int F4(int x,int y){
43     while(y>x){
44         y -= y & (-y);
45     }return y;
46 }

```

Brain Kernighan算法

异或技巧

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define M 500
4 int a1[M];
5 int n;
6 // 以下两个全局变量用于功能1：两个出现次数为奇数次的数字
7 int ero1 = 0;
8 int ero2 = 0;
9 void solve(int a[], int size) {
10     ero1 = 0;
11     ero2 = 0;
12     // 计算所有数字的异或结果
13     for (int i = 0; i < size; i++) {
14         ero1 ^= a[i];
15     }
16     // 找到最右侧的1
17     int rightone = ero1 & (-ero1);
18     // 根据该位是否为1将数字分组，并对其中一组异或
19     for (int i = 0; i < size; i++) {
20         if ((a[i] & rightone) == 0) {
21             ero2 ^= a[i];
22         }
23     }
24     // 另一个奇数次数字
25     ero1 = ero1 ^ ero2;
26 }
27 // 以下全局变量用于功能2：求数组中出现次数少于 cnt 次的数字
28 int ans = 0;
29 int bits[32]; // 用于统计各二进制位1的个数
30 void solve2(int size, int cnt, int a[]) {
31     ans = 0;
32     memset(bits, 0, sizeof(bits));
33     // 统计每一位的1的个数
34     for (int i = 0; i < size; i++) {
35         for (int j = 0; j < 32; j++) {
36             // 等价于：if(a[i] & (1 << j)) bits[j]++;
37             bits[j] += (a[i] >> j) & 1;
38         }
39     }
40     // 对每一位判断是否为目标数字的1
41     for (int j = 0; j < 32; j++) {
42         if (bits[j] % cnt) {
43             ans |= (1 << j);
44         }
45     }
46 }
```

链表基本操作

返回两个无环链表相交的第一个节点

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 struct LinkNode{
4     int val;
5     LinkNode *next;
6     LinkNode(int _val) : val(_val), next(nullptr) {}
7 };
8 //返回两个无环链表相交的第一个节点
9 LinkNode *converge(LinkNode *&L1, LinkNode *&L2) {
10     LinkNode *p1=L1->next;
11     LinkNode *p2=L2->next;
12     int cnt=0;
13     if(L1==nullptr || L2==nullptr) return nullptr;
14     while(p1!=nullptr){
15         cnt++;
16         p1=p1->next;
17     }
18     while(p2!=nullptr){
19         cnt--;
20         p2=p2->next;
21     }
22     p1=L1;
23     p2=L2;
24     if(cnt>=0){
25         while(cnt--) p1=p1->next;
26     }
27     else{
28         cnt=abs(cnt);
29         while(cnt--) p2=p2->next;
30     }
31     while(p1!=p2 && p1!=nullptr && p2!=nullptr){
32         p1=p1->next;
33         p2=p2->next;
34     }
35     return p1;
36 }
```

每k个节点一组翻转链表

```
1 //每k个节点一组翻转链表
2 void reverse(int k, LinkNode *&L) {
3     if(k<=1 || L==nullptr || L->next==nullptr) return;
4     LinkNode *p1=L->next;
5     int cnt=0;
6     while(p1!=nullptr){
7         cnt++;
8         p1=p1->next;
9     }
10    if(cnt<k) return;
11    p1=L;
```

```

12     LinkNode *pre;
13     LinkNode *p;
14     LinkNode *pp;
15     LinkNode *p0=L;
16     LinkNode *last=p0->next;
17     for(int i=0;i<k;i++) p1=p1->next;
18     p=p0->next;
19     pp=p->next;
20     p->next=p1->next;
21     pre=p;
22     for(int i=1;i<k;i++) {
23         p=pp;
24         pp=p->next;
25         p->next=pre;
26         pre=p;
27     }
28     p0->next=p1;
29     for(int j=cnt-k;j>=k;j-=k) {
30         p0=last;
31         p1=last;
32         for(int i=0;i<k;i++) p1=p1->next;
33         p=p0->next;
34         last=p;
35         pp=p->next;
36         p->next=p1->next;
37         pre=p;
38         for(int i=1;i<k;i++) {
39             p=pp;
40             pp=p->next;
41             p->next=pre;
42             pre=p;
43         }
44         p0->next=p1;
45     }
46 }

```

复制带随机指针的链表

```

1 //复制带随机指针的链表
2 class NodeR{
3 public:
4     int val;
5     NodeR *next;
6     NodeR *random;
7     NodeR(int x){
8         val=x;
9         next=nullptr;
10        random=nullptr;
11    }
12 };
13 NodeR * copy(NodeR *&L1){
14     if(L1==nullptr) return nullptr;
15     NodeR* cur=L1->next;
16     //复制节点并插入原节点之后
17     while(cur!=nullptr){
18         NodeR* next=new NodeR(cur->val);

```

```

19         next->next=cur->next;
20         cur->next=next;
21         cur=next->next;
22     }
23     //更新节点的随机指针
24     cur=L1->next;
25     while(cur!=nullptr){
26         if(cur->random!=nullptr){
27             cur->next->random=cur->random;
28         }
29         cur=cur->next->next;
30     }
31     //分离链表
32     cur=L1->next;
33     NodeR* L2=new NodeR(0);
34     NodeR* temp=L2;
35     while(cur!=nullptr){
36         temp->next=cur->next;
37         cur->next=cur->next->next;
38         cur=cur->next;
39         temp=temp->next;
40     }
41     return L2;
42 }
```

判断是否是回文结构（快慢指针）

```

1 //判断链表是否是回文结构（快慢指针）
2 bool judge_HuiWen(LinkNode *L){
3     if(L==nullptr || L->next==nullptr) return true;
4     //快慢指针找中点
5     LinkNode* f=L;
6     LinkNode* s=L;
7     while(f!=nullptr && f->next!=nullptr){
8         f=f->next->next;
9         s=s->next;
10    }
11    //翻转
12    LinkNode* pre=s;
13    LinkNode* cur=s->next;
14    LinkNode* next=nullptr;
15    pre->next=nullptr;
16    while(cur!=nullptr){
17        next=cur->next;
18        cur->next=pre;
19        pre=cur;
20        cur=next;
21    }
22    //判断回文
23    bool ans=true;
24    LinkNode* sta1=L->next;
25    LinkNode* sta2=pre;
26    while(sta1!=nullptr && sta2!=nullptr){
27        if(sta1->val!=sta2->val){
28            ans=false;
29            break;
30        }
31    }
32    return ans;
33 }
```

```

30         }
31         sta1=sta1->next;
32         sta2=sta2->next;
33     }
34     //还原
35     cur=pre->next;
36     pre->next=nullptr;
37     next=nullptr;
38     while(cur!=nullptr) {
39         next=cur->next;
40         cur->next=pre;
41         pre=cur;
42         cur=next;
43     }
44     return ans;
45 }
```

返回第一个入环节点

```

1 //返回链表的第一个入环节点
2 LinkNode* cycle(LinkNode *L) {
3     if (L == nullptr || L->next == nullptr) {
4         return nullptr;
5     }
6     LinkNode* f=L;
7     LinkNode* s=L;
8     while(f!=nullptr && f->next!=nullptr) {
9         s=s->next;
10        f=f->next->next;
11        if(s==f) {
12            f=L;
13            while(f!=s) {
14                f=f->next;
15                s=s->next;
16            }
17            return f;
18        }
19    }return nullptr;
20 }
```

稳定排序

```

1 //在链表上排序，时间复杂度O (n*logn) ，额外空间复杂度 (1) ，排序有稳定性
2 LinkNode* createNode(int data) {
3     return new LinkNode(data);
4 }
5 // 在链表尾部添加节点
6 void appendNode(LinkNode*& head, int data) {
7     LinkNode* newNode = createNode(data);
8     LinkNode* temp = head;
9     while (temp->next) {
10         temp = temp->next;
11     }
12     temp->next = newNode;
13 }
14 LinkNode* start;
```

```

15 LinkNode* end1;
16 LinkNode* FindEnd(LinkNode* l,int len) {
17     while(l->next!=nullptr && --len!=0) {
18         l=l->next;
19     }return l;
20 }
21 void merge(LinkNode* l1, LinkNode* r1, LinkNode* l2, LinkNode* r2) {
22     LinkNode* pre;
23     if (l1->val <= l2->val) {
24         start = l1;
25         pre = l1;
26         l1 = l1->next;
27     } else {
28         start = l2;
29         pre = l2;
30         l2 = l2->next;
31     }
32     while (l1 != nullptr && l2 != nullptr) {
33         if (l1->val <= l2->val) {
34             pre->next = l1;
35             pre = l1;
36             l1 = l1->next;
37         } else {
38             pre->next = l2;
39             pre = l2;
40             l2 = l2->next;
41         }
42     }
43     if (l1 != nullptr) {
44         pre->next = l1;
45         end1 = r1;
46     } else {
47         pre->next = l2;
48         end1 = r2;
49     }
50 }
51 void sort(LinkNode*& L){
52     if (L == nullptr || L->next == nullptr) {
53         return;
54     }
55     int len=0;
56     LinkNode* temp=L;
57     while(temp!=nullptr) {
58         len++;
59         temp=temp->next;
60     }
61     LinkNode* l1;
62     LinkNode* r1;
63     LinkNode* l2;
64     LinkNode* r2;
65     LinkNode* lastTeamend;
66     LinkNode* next;
67     for(int i=1;i<len;i<<=1) {
68         l1=L;
69         r1=FindEnd(l1,i);
70         l2=r1->next;

```

```

71     r2=FindEnd(l2,i);
72     next = r2->next;
73     r1->next = nullptr;
74     r2->next = nullptr;
75     merge(l1,r1,l2,r2);
76     L=start;
77     lastTeamend=end1;
78     while(next!=nullptr) {
79         l1=next;
80         r1=FindEnd(l1,i);
81         l2 = r1->next;
82         if(l2==nullptr) {
83             lastTeamend->next = l1;
84             break;
85         }
86         r2 = FindEnd(l2, i);
87         next = r2->next;
88         r1->next = nullptr;
89         r2->next = nullptr;
90         merge(l1,r1,l2,r2);
91         lastTeamend->next=start;
92         lastTeamend=end1;
93     }
94 }
95 }
```

合并n个有序链表

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define M 500
4 struct ListNode{
5     int val;
6     ListNode * next;
7 };
8 void Create(ListNode *&q) {
9     q=(ListNode *)malloc(sizeof(ListNode));
10    q->next=nullptr;
11 }
12 struct cmp{
13     bool operator()(ListNode* a,ListNode* b) {
14         return a->val > b->val;
15     }
16 };
17
18 ListNode *merge(vector<ListNode*>& v1) {
19     priority_queue<ListNode*,vector<ListNode*>,cmp> pq1;
20     for(ListNode* it : v1) if(it) pq1.push(it);
21     ListNode* ans;
22     Create(ans);
23     ListNode* temp;
24     ListNode* pre=ans;
25     while(!pq1.empty()) {
26         temp=pq1.top();
27         pq1.pop();
28         pre->next=temp;
```

```

29         pre=temp;
30         if(temp->next) pq1.push(temp->next);
31     }pre->next=nullptr;
32     return ans;
33 }
34
35 int main() {
36     int n;
37     cin>>n;
38     int cnt;
39     ListNode* pre;
40     int temp1;
41     vector<ListNode*> v1;
42     while(n--) {
43         cin>>cnt;
44         ListNode * l=(ListNode*)malloc(sizeof(ListNode));
45         l->next=nullptr;
46         pre=l;
47         for(int i=0;i<cnt;i++) {
48             cin>>temp1;
49             ListNode *temp=(ListNode*)malloc(sizeof(ListNode));
50             temp->val=temp1;
51             temp->next=pre->next;
52             pre->next=temp;
53             pre=temp;
54         }v1.push_back(l->next);
55     }
56     ListNode * final;
57     Create(final);
58     final=merge(v1);
59     pre=final->next;
60     while(pre) {
61         cout<<pre->val<<" ";
62         pre=pre->next;
63     }cout<<endl;
64     return 0;
65 }

```

莫队算法

离线加速处理查询

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct Query {
4     int l, r, id, blk;
5 };
6 int main() {
7     ios::sync_with_stdio(false);
8     cin.tie(nullptr);
9     int n, q;
10    cin >> n >> q;
11    vector<int> original(n + 1), ans(q);
12    for (int i = 1; i <= n; i++) {
13        cin >> original[i];

```

```

14 }
15 // 离散化
16 vector<int> temp = original;
17 sort(temp.begin() + 1, temp.end());
18 int m = 1;
19 for(int i = 2; i <= n; i++) {
20     if(temp[m] != temp[i]) {
21         temp[++m] = temp[i];
22     }
23 }
24 auto find = [&](int x) -> int {
25     int l = 1, r = m;
26     int mid, ans;
27     while(l <= r) {
28         mid = (l + r) >> 1;
29         if(temp[mid] <= x) {
30             ans = mid;
31             l = mid + 1;
32         } else {
33             r = mid - 1;
34         }
35     }
36     return ans;
37 };
38 // 莫队函数
39 int now_ans = 0;
40 vector<int> fre(n + 1, 0);
41 auto add = [&](int x) {
42     int &f = fre[find(x)];
43     if (f == 1) {
44         now_ans++;
45     } else if (f == 2) {
46         now_ans--;
47     }
48     f++;
49 };
50 auto remove = [&](int x) {
51     int &f = fre[find(x)];
52     if (f == 2) {
53         now_ans--;
54     } else if (f == 3) {
55         now_ans++;
56     }
57     f--;
58 };
59 // 窗口优化
60 int blk = (int)sqrt(n);
61 vector<Query> queries(q);
62 for (int i = 0; i < q; i++) {
63     cin >> queries[i].l >> queries[i].r;
64     queries[i].id = i;
65     queries[i].blk = queries[i].l / blk;
66 }
67 sort(queries.begin(), queries.end(), [&](const Query &A, const Query &B) {
68     if (A.blk == B.blk) return A.r < B.r;
69     return A.blk < B.blk;

```

```

70    });
71    int curL = 1, curR = 0;
72    for (auto &q : queries) {
73        while (curR < q.r) add(original[+curR]);
74        while (curR > q.r) remove(original[curR--]);
75        while (curL < q.l) remove(original[curL++]);
76        while (curL > q.l) add(original[--curL]);
77        ans[q.id] = now_ans;
78    }
79    for (int i = 0; i < q; i++) cout << ans[i] << "\n";
80    return 0;
81 }

```

KMP算法

字符匹配匹配加速，加速的点在于匹配失败后不一定直接返回起点重新匹配，而是跳到最大前缀处，如
`abab.....` 匹配失败时，会跳到2位置而不是0位置

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(){
4     string s1, s2;
5     cin >> s1 >> s2;
6     int n = int(s1.size());
7     int m = int(s2.size());
8     vector<int> next1(m + 1, 0);
9     int i1 = 2;
10    int mark1 = 0;
11    while(i1 <= m) {
12        if(s2[i1 - 1] == s2[mark1]){
13            next1[i1++] = ++ mark1;
14        }else if(mark1 != 0){
15            mark1 = next1[mark1];
16        }else{
17            next1[i1++] = 0;
18        }
19    }
20    int j1 = i1 = 0;
21    while(j1 < n) {
22        if(s1[j1] == s2[i1]){
23            j1++, i1++;
24        }else if(i1 != 0){
25            i1 = next1[i1];
26        }else{
27            j1++;
28        }
29        if(i1 == m) {
30            cout << j1 - i1 + 1 << "\n";
31            i1 = next1[i1];
32        }
33    }
34    for(int i = 1; i <= m; i ++){
35        cout << next1[i];
36        if(i != m) cout << " ";

```

```
37     }
38     return 0;
39 }
```

manacher算法

以某一节点为中心向两端扩展的最大回文串

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 #define fr first
5 #define sc second
6 string manacherss(string s){
7     string ss = "#";
8     for(int i = 0; i < s.size(); i ++){
9         ss += s[i];
10        ss += '#';
11    }
12    return ss;
13 }
14 void solve(){
15     string s;
16     cin >> s;
17     string ss = manacherss(s);
18     int n = ss.size();
19     vector<int> r1(n, 0); // 真实回文串的长度为r1[i] - 1;
20     int ans = 0;
21     for(int i = 0, r = 0, c = 0, len; i < n; i ++){
22         len = r > i ? min(r1[2 * c - i], r - i) : 1;
23         while(i - len >= 0 && i + len < n && ss[i - len] == ss[i + len]){
24             len++;
25         }
26         if(len + i > r){
27             r = len + i;
28             c = i;
29         }
30         r1[i] = len;
31         ans = max(ans, len);
32     }
33     cout << ans - 1 << "\n";
34 }
35 signed main() {
36     ios::sync_with_stdio(false);
37     cin.tie(nullptr);
38
39     int t = 1;
40     // cin >> t;
41     while (t--) {
42         solve();
43     }
44     return 0;
45 }
```

Z函数

对于一个长度为n的字符串s， 定义函数z[i]表示s和s[i, n - 1]的最长公共前缀（LCP）的长度则z被成为s的Z函数

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 // https://www.luogu.com.cn/problem/P5410
4 #define int long long
5 #define fr first
6 #define sc second
7 void Z1(vector<int> & z, string s, int n) {
8     z[0] = n;
9     for(int i = 1, c = 1, r = 1, len; i < n; i++) {
10         len = r > i ? min(z[i - c], r - i) : 0;
11         while(i + len < n && s[len] == s[i + len]) {
12             len++;
13         }
14         if(i + len > r) {
15             r = i + len;
16             c = i;
17         }
18         z[i] = len;
19     }
20 }
21 void E1(vector<int> & e, vector<int> & z, string s1, string s2, int n) {
22     for(int i = 0, r = 0, c = 0, len; i < n; i++) {
23         len = r > i ? min(z[i - c], r - i) : 0;
24         while(i + len < n && s1[i + len] == s2[len]) {
25             len++;
26         }
27         if(i + len > r) {
28             r = i + len;
29             c = i;
30         }
31         e[i] = len;
32     }
33 }
34 void solve() {
35     string s1, s2;
36     cin >> s1 >> s2;
37     // 求得该字符串的最长前缀
38     int n = s2.size();
39     vector<int> Z(n, 0);
40     Z1(Z, s2, n);
41     // 求该串以某一串为基准的最长前缀
42     int m = s1.size();
43     vector<int> E(m + 1);
44     E1(E, Z, s1, s2, m);
45 }
```

前缀树

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 //静态数组版本
4 const int M = 150001;
5 const int M2 = 28;
6 int cnt=1;
7 int pass1[M];
8 int endl1[M];
9 int t1[M][M2];
10 int op1;
11 string op2;
12 void build() {
13     cnt = 1;
14     memset(t1, 0, sizeof(t1));
15     memset(endl1, 0, sizeof(endl1));
16     memset(pass1, 0, sizeof(pass1));
17 }
18 void insert(const string& word) {
19     int cur=1;
20     pass1[cur]++;
21     for(int i=0;i<word.size();i++) {
22         int path = word[i] - 'a';
23         if(t1[cur][path]==0) {
24             t1[cur][path]=++cnt;
25         }
26         cur=t1[cur][path];
27         pass1[cur]++;
28     }endl1[cur]++;
29 }
30 int search(const string& word) {
31     int cur=1,int path;
32     for(int i=0;i<word.size();i++) {
33         path=word[i] - 'a';
34         if(t1[cur][path]==0) {
35             return 0;
36         }
37         cur=t1[cur][path];
38     }return endl1[cur];
39 }
40 void erase(const string& word) {
41     if(search(word)>0) {
42         int path;
43         int cur=1;
44         for(int i=0;i<word.size();i++) {
45             path = word[i] - 'a';
46             if(--pass1[t1[cur][path]]==0) {
47                 t1[cur][path]=0;
48                 // 之后会重新分配节点，所以遍历不到未减减的end
49                 return ;
50             }cur=t1[cur][path];
51         }
52         endl1[cur]--;
53     }
54 }
```

```

55 int prefixNumber(const string& pre) {
56     int cur=1;
57     int path;
58     for(int i=0;i<pre.size();i++) {
59         path = pre[i]-'a';
60         if(t1[cur][path]==0) {
61             return 0;
62         }
63         cur=t1[cur][path];
64     }return pass1[cur];
65 }
66 int main() {
67     string line;
68     while (getline(cin, line)) {
69         build();
70         int m = stoi(line);
71         for (int i = 0; i < m; ++i) {
72             getline(cin, line);
73             stringstream ss(line);
74             int op;
75             string word;
76             ss >> op >> word;
77             if (op == 1) {
78                 insert(word);
79             } else if (op == 2) {
80                 erase(word);
81             } else if (op == 3) {
82                 cout << (search(word) > 0 ? "YES" : "NO") << endl;
83             } else if (op == 4) {
84                 cout << prefixNumber(word) << endl;
85             }
86         }
87     }
88     return 0;
89 }

```

AC自动机

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 // https://www.luogu.com.cn/problem/P3796
4 const int N = 2e4;
5 int cnt1;
6 vector<int> fail1(N);
7 vector<vector<int>> tree1(N, vector<int>(26, 0));
8 int n;
9 vector<int> end1(N);
10 vector<int> times1(N);
11 vector<int> v1(N); // BFS队列实现
12 // 链式前向星
13 int cnt2;
14 vector<int> head(N);
15 vector<int> next1(N);
16 vector<int> to(N);
17 string s1;
18 // 构建前缀树

```

```

19 vector<string> F1() {
20     vector<string> s1(n + 1);
21     string templ;
22     for(int i = 1, x; i <= n; i++) {
23         cin >> templ;
24         s1[i] = templ;
25         x = 0;
26         for(int j = 0, y; j < templ.size(); j++) {
27             y = templ[j] - 'a';
28             if(tree1[x][y] == 0) {
29                 tree1[x][y] = ++cnt1;
30             }
31             x = tree1[x][y];
32         }
33         endl[i] = x;
34     }
35     return s1;
36 }
37 // 构建自动机
38 void F2() {
39     // BFS遍历
40     int l, r;
41     l = r = 0;
42     for(int i = 0; i < 26; i++) {
43         if(tree1[0][i] != 0) {
44             v1[r++] = tree1[0][i];
45         }
46     }
47     int u;
48     while(l < r) {
49         u = v1[l++];
50         for(int i = 0; i < 26; i++) {
51             if(tree1[u][i] == 0) {
52                 tree1[u][i] = tree1[fail1[u]][i];
53             } else {
54                 fail1[tree1[u][i]] = tree1[fail1[u]][i];
55                 v1[r++] = tree1[u][i];
56             }
57         }
58     }
59 }
60 // 反向建图
61 void add1(int u, int v) {
62     next1[++cnt2] = head[u];
63     head[u] = cnt2;
64     to[cnt2] = v;
65 }
66 // 统计次数
67 void F3(int x) {
68     for(int e = head[x]; e > 0; e = next1[e]) {
69         F3(to[e]);
70         times1[x] += times1[to[e]];
71     }
72 }
73 void build() {
74     for(int i = 0; i < N; i++) {

```

```

75     for(int j = 0; j < 26; j++) {
76         tree1[i][j] = 0;
77     }
78 }
79 cnt2 = cnt1 = 0;
80 fill(head.begin(), head.end(), 0);
81 fill(times1.begin(), times1.end(), 0);
82 fill(fail1.begin(), fail1.end(), 0);
83 }
84 void solve() {
85     build();
86     vector<string> ss = F1();
87     F2();
88     cin >> s1;
89     for(int i = 0, u = 0; i < s1.size(); i++) {
90         u = tree1[u][s1[i] - 'a'];
91         times1[u]++;
92     }
93     for(int i = 1; i <= cnt1; i++) {
94         add1(fail1[i], i);
95     }
96     F3(0);
97     int max1 = -1;
98     for(int i = 1; i <= n; i++) {
99         if(times1[end1[i]] > max1) {
100             max1 = times1[end1[i]];
101         }
102     }
103     cout << max1 << "\n";
104     for(int i = 1; i <= n; i++) {
105         if(times1[end1[i]] == max1) {
106             cout << ss[i] << "\n";
107         }
108     }
109 }
110 int main() {
111     ios::sync_with_stdio(false);
112     cin.tie(nullptr);
113     cin >> n;
114     while(n) {
115         solve();
116         cin >> n;
117     }
118     return 0;
119 }

```

字符串哈希

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // https://www.luogu.com.cn/problem/P3370
4 // 利用质数进制信息
5 #define int long long
6 #define fr first
7 #define sc second
8 int base = 39127;

```

```

9 int v(char c){
10    int ans;
11    if('0' <= c && '9' >= c) {
12        ans = c - '0' + 1;
13    }else if('a' <= c && 'z' >= c) {
14        ans = c - 'a' + 11;
15    }else{
16        ans = c - 'A' + 37;
17    }
18    return ans;
19 }
20 // 加速进制
21 int F(string s){
22    int ans = v(s[0]);
23    for(int i = 1; i < s.size(); i ++){
24        ans = ans * base + v(s[i]);
25    }
26    return ans;
27 }
28 void solve() {
29    int n;
30    cin >> n;
31    vector<int> v1(n);
32    string s;
33    for(int i = 0; i < n; i ++){
34        cin >> s;
35        v1[i] = F(s);
36    }
37    sort(v1.begin(), v1.end());
38    int ans = 1;
39    for(int i = 1; i < n; i ++){
40        if(v1[i] != v1[i - 1]){
41            ans++;
42        }
43    }
44    cout << ans << "\n";
45 }
46 // https://www.luogu.com.cn/record/204776127
47 // 要求一段区间内的hash值，用前缀和
48 int H(vector<int> & hash, vector<int> & power, int l, int r){
49    if(l > 0){
50        l = hash[l - 1] * power[r - l + 1];
51    }
52    r = hash[r];
53    return r - l;
54 }
55 void solve2(){
56    string s1, s2;
57    cin >> s1 >> s2;
58    int n = s1.size(), m = s2.size();
59    vector<int> hash1(n), hash2(m), power(max(n, m));
60    power[0] = 1;
61    for(int i = 1; i < power.size(); i ++){
62        power[i] = power[i - 1] * base;
63    }
64    hash1[0] = s1[0] - 'A' + 1;

```

```

65  for(int i = 1; i < n; i++) {
66      hash1[i] = hash1[i - 1] * base + s1[i] - 'A' + 1;
67  }
68  hash2[0] = s2[0] - 'A' + 1;
69  for(int i = 1; i < m; i++) {
70      hash2[i] = hash2[i - 1] * base + s2[i] - 'A' + 1;
71  }
72  int ans = 0;
73  int l1, l2, r1;
74  int l, r, len, mid;
75  int k;
76  for(int i = 0; i <= n - m; i++) {
77      l1 = i, r1 = i + m - 1;
78      k = 0;
79      l2 = 0;
80      while(l1 <= r1 && k <= 3) {
81          l = l1, r = r1;
82          len = 0;
83          while(l <= r) {
84              mid = (l + r) >> 1;
85              if(H(hash1, power, l1, mid) == H(hash2, power, l2, 12 + mid -
l1)) {
86                  len = mid - l1 + 1;
87                  l = mid + 1;
88              } else {
89                  r = mid - 1;
90              }
91          }
92          if(l1 + len <= r1) {
93              k++;
94              l1 += len + 1;
95              l2 += len + 1;
96          } else {
97              break;
98          }
99      }
100     if(k <= 3) {
101         ans++;
102     }
103 }
104 cout << ans << "\n";
105 }

```

异或哈希

题目大概描述：给定一个数字序列a，请你找出最长的连续子区间，使得这个子区间内所有数字的乘积是一个完全平方数。如果存在多个长度相同的子区间，选择最靠左的一个。

完全平方数可以差分为质因数乘积，且所有质因数的个数均为偶数。

故把每一个数的值改为落单质因数的哈希值，如果某一个区间的异或值为0，就说明该区间的质因数都是偶数，说明乘积是一个完全平方数，利用前缀和思想可以加速这一查找过程（当一个状态第一次出现时，记录位置，以后每次出现都说明中间与第一次出现的位置的间隔为一个完全平方数）。

```

1 using ull = unsigned long long;
2 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
3 constexpr int N = 1e6 + 1;
4 vector<int> spf(N); // 最小质因子
5 vector<ull> prime_hash(N);
6 auto init() = [&](){
7     iota(spf.begin(), spf.end(), 0);
8     for(int i = 2; i < N; i ++){
9         if(spf[i] == i){
10             prime_hash[i] = rng();
11             for(int j = i << 1; j < N; j += i){
12                 if(spf[j] == j){
13                     spf[j] = i;
14                 }
15             }
16         }
17     }
18 }
19 // 根据最小质因子求某一个数的hash值
20 auto get_hash = [&](int x) -> ull {
21     ull ans = 0;
22     while(x > 1){
23         int p = spf[x];
24         int cnt = 0;
25         while(x % p == 0){
26             x /= p;
27             cnt ^= 1;
28         }
29         if(cnt & 1){
30             ans ^= prime_hash[p];
31         }
32     }
33     return ans;
34 };
35 int t = 1;
36 cin >> t;
37 while(t --){
38     int n;
39     cin >> n;
40     vector<int> v1(n + 1);
41     vector<ull> pref(n + 1);
42     pref[0] = 0;
43     for(int i = 1; i <= n; i ++){
44         cin >> v1[i];
45         pref[i] = pref[i - 1] ^ get_hash(v1[i]);
46     }
47     unordered_map<ull,int> pos;
48     pos[0] = 0;
49     int ans = 0, l = -1, r = -1;
50     ull x;
51     for(int i = 1; i <= n; i ++){
52         x = pref[i];
53         if(pos.count(x)){
54             if(i - pos[x] > ans){
55                 ans = i - pos[x];
56                 l = pos[x] + 1, r = i;
57             }
58         }
59     }
60 }
```

```

57         }
58     }else{
59         pos[x] = i;
60     }
61 }
62 cout << l << " " << r << "\n";
63 }

```

质数/素数

质数判别

Miller Rabin的时间复杂度为 $O(\log n)^3$.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 // 普通判别方法
4 int p(int n) {
5     if (n <= 3) {
6         return n > 1;
7     }
8     if (n % 6 != 1 && n % 6 != 5) {
9         return false;
10    }
11    int sqrt1 = (int) sqrt(n);
12    for (int i = 5; i <= sqrt1; i += 6) {
13        if (n % i == 0 || n % (i + 2) == 0) {
14            return false;
15        }
16    }
17    return true;
18 }
19 // Miller Rabin Prime判别法
20 using ll = long long;
21 // 大数的时候用
22 //typedef __int128 ll;
23 vector<ll> primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
24 ll power(ll base, ll exp, ll mod) {
25     ll result = 1;
26     base %= mod;
27     while (exp > 0) {
28         if (exp & 1) result = (result * base) % mod;
29         base = (base * base) % mod;
30         exp >>= 1;
31     }
32     return result;
33 }
34 bool witness(ll a, ll n) {
35     ll u = n - 1;
36     int t = 0;
37     while ((u & 1) == 0) {
38         t++;
39         u >>= 1;
40     }

```

```

41     ll x = power(a, u, n);
42     if (x == 1 || x == n - 1) return false;
43     for (int i = 0; i < t - 1; i++) {
44         x = (x * x) % n;
45         if (x == n - 1) return false;
46     }
47     return true;
48 }
49 bool millerRabin(ll n) {
50     if (n < 2) return false;
51     if (n % 2 == 0) return n == 2;
52     for (ll p : primes) {
53         if (p >= n) break;
54         if (witness(p, n)) return false;
55     }
56     return true;
57 }

```

线性质数筛

```

1 // 质数筛
2 void F1(int x){
3     vector<int> v1;
4     for(int i = 2; i * i <= x; i ++){
5         if(x % i != 0){
6             v1.push_back(i);
7             while(x % i == 0) x /= i;
8         }
9     }
10    if(x > 1) v1.push_back(x);
11 }
12 // 求数值小于等于n的质数的个数,
13 // 质数筛-埃氏筛
14 void solve() {
15     int n;
16     cin >> n;
17     vector<int> v(n + 1, 0);
18     v[1] = 1;
19     int cnt = 0;
20     for (int i = 2; i <= sqrt(n); i++){
21         if (v[i] == 0){
22             for (int j = i + i; j <= n; j += i) v[j] = 1;
23         }
24     }
25     for (int i = 1; i <= n; i++) {
26         if (!v[i]) cnt++;
27     }
28     cout << cnt << endl;
29 }
30 // 只是计数的话, 埃氏筛还能改进
31 int ehrlich2(int n) {
32     if (n <= 1) {
33         return 0;
34     }
35     vector<bool> visit(n + 1, false);
36     int cnt = (n + 1) / 2; // 估计质数数量(奇数)

```

```

37     for (int i = 3; i * i <= n; i += 2) {
38         if (!visit[i]) {
39             for (int j = i * i; j <= n; j += 2 * i) {
40                 if (!visit[j]) {
41                     visit[j] = true;
42                     cnt--;
43                 }
44             }
45         }
46     }
47     return cnt;
48 }
49 // 使用线性筛法计算每个数字的最小质因子 `spf` (smallest prime factor)
50 constexpr int N = 1e6 + 1;
51 vector<int> spf(N);
52 auto init() = [&]() {
53     iota(spf.begin(), spf.end(), 0);
54     for(int i = 2; i < N; i++) {
55         if(spf[i] == i) {
56             prime_hash[i] = rng();
57             for(int j = i << 1; j < N; j += i) {
58                 if(spf[j] == j) {
59                     spf[j] = i;
60                 }
61             }
62         }
63     }
64 }
65 // 欧拉筛时间复杂度O(n)
66 int euler(int n) {
67     vector<bool> visit(n + 1, false);
68     vector<int> prime(n / 2 + 1);
69     int cnt = 0;
70     for (int i = 2; i <= n; i++) {
71         if (!visit[i]) {
72             prime[cnt++] = i;
73         }
74         for (int j = 0; j < cnt; j++) {
75             if (i * prime[j] > n) {
76                 break;
77             }
78             visit[i * prime[j]] = true;
79             if (i % prime[j] == 0) {
80                 // 保证每个合数只被最小的质数遍历一次
81                 break;
82             }
83         }
84     }
85     return cnt;
86 }

```

Pick定理

S = i + b / 2 - 1

`s`：图形的面积（可以是实数）；

`i`：图形内部的格点数；

`b`：图形边界上的格点数（包括顶点）；

给定两个整数点 (x_1, y_1) , (x_2, y_2) , 两点间（包括端点）的格点数为 $\text{gcd}(|x_2 - x_1|, |y_2 - y_1|) + 1$

则求某区域内的格点数的方法为 $i = s - 2 / b + 1$

差分

对区间进行快速加法操作

一维

```
1 // 若有原数组则要分开处理，最后加上原数组或者进行以下处理
2 v[i] = original[i] - original[i - 1];
3 void add(int l, int r, int x, vector<int> & v) {
4     v[l] += x;
5     v[r + 1] -= x;
6 }
7 for(int i = 1; i <= n; i++) {
8     v[i] += v[i - 1];
9 }
```

二维

```
1 // 和原数组分开处理，最后加上原数组，或者进行以下处理
2 v[i][j] = a[i][j] - a[i - 1][j] - a[i][j - 1] + a[i - 1][j - 1];
3 void add(int x1, int y1, int x2, int y2, int val, vector<vector<int>> v) {
4     v[x1][y1] += val;
5     v[x2 + 1][y1] -= val;
6     v[x1][y2 + 1] -= val;
7     v[x2 + 1][y2 + 1] += val;
8 }
```

序列/数组问题

最长公共子序列 (LCS)

递归/递推+路径

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 // 递归版
4 int n, m;
5 string s1, s2;
6 vector<vector<int>> dp(1001, vector<int>(1001, -1));
7 int F1(int i, int j){
```

```

8     if(i == n || j == n){
9         return 0;
10    }else{
11        if(dp[i][j] != -1) return dp[i][j];
12        int ans = 0;
13        if(s1[i] == s2[j]){
14            ans = max({F1(i + 1, j + 1) + 1, F1(i + 1, j), F1(i, j + 1)});
15        }else{
16            ans = max(F1(i + 1, j), F1(i, j + 1));
17        }
18        dp[i][j] = ans;
19        return ans;
20    }
21}
22 // 递推版
23 void F2(){
24     string s1, s2;
25     cin >> s1 >> s2;
26     int n = s1.size(), m = s2.size();
27     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
28     for(int i = 1; i <= n; i ++){
29         for(int j = 1; j <= m; j ++){
30             if(s1[i - 1] == s2[j - 1]){
31                 dp[i][j] = dp[i - 1][j - 1] + 1;
32             }else{
33                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
34             }
35         }
36     }
37 // 求路径
38 string s;
39 for(int i = n, j = m; i && j;){
40     if(s1[i - 1] == s2[j - 1]){
41         s += s1[i - 1];
42         i--;
43         j--;
44     }else{
45         if(dp[i - 1][j] > dp[i][j - 1]) i--;
46         else j--;
47     }
48 }
49 if(s.size() == 0){
50     cout << -1 << "\n";
51     return 0;
52 }
53 for(int i = 0, j = s.size() - 1; i < j; i++, j--) swap(s[i], s[j]);
54 cout << s << "\n";
55 return 0;
56}
57

```

最长递增子序列

两个序列包含完全相同的数字使，可以将一个数组的顺序映射到另一个数组上，这时其最长公共子序列变成求最长递增子序列问题（或者反向求最长递减子序列）

普通版

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int F1(int x, vector<int> & v) {
4     int l = 0;
5     int r = v.size() - 1;
6     int ans = -1;
7     int mid;
8     while(l <= r) {
9         mid = (l + r) >> 1;
10        if(x < v[mid]) l = mid + 1;
11        else{
12            r = mid - 1;
13            ans = mid;
14        }
15    }
16    return ans;
17 }
18 int main() {
19     ios::sync_with_stdio(false);
20     cin.tie(nullptr);
21     int n;
22     cin >> n;
23     vector<int> v2(n + 1), temp(n + 1);
24     for(int i = 1, x; i <= n; i ++){
25         cin >> x;
26         temp[x] = i;
27     }
28     for(int i = 1; i <= n ; i ++){
29         cin >> v2[i];
30         v2[i] = temp[v2[i]];
31     }
32     vector<int> ans;
33     for(int i = n, x; i; i --){
34         x = F1(v2[i], ans);
35         if(x == -1) {
36             ans.push_back(v2[i]);
37         } else{
38             ans[x] = v2[i];
39         }
40     }
41     cout << ans.size() << "\n";
42     return 0;
43 }
```

字典序最小版

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int F1(int x, vector<int> & v) {
4     int ans = -1;
5     int l = 0;
6     int r = v.size() - 1;
7     int mid;
8     while(l <= r) {
```

```

9         mid = (l + r) >> 1;
10        if(x < v[mid]){
11            l = mid + 1;
12        }else{
13            r = mid - 1;
14            ans = mid;
15        }
16    }
17    return ans;
18}
19int main(){
20    ios::sync_with_stdio(false);
21    cin.tie(nullptr);
22    cout.tie(nullptr);
23    int n;
24    cin >> n;
25    vector<int> v1(n);
26    for(auto& x : v1) cin >> x;
27    vector<int> v2;
28    vector<int> dp(n);
29    for(int i = n - 1, ans; i >= 0; i --){
30        ans = F1(v1[i], v2);
31        if(ans == -1){
32            v2.push_back(v1[i]);
33            dp[i] = v2.size();
34        }else{
35            v2[ans] = v1[i];
36            dp[i] = ans + 1;
37        }
38    }
39    int m = v2.size();
40    vector<int> ans(m, 1e9);
41    for(auto x : dp) cout << x << " "; cout << "\n";
42    for(int i = 0, x; i < n; i ++){
43        x = dp[i];
44        if(x == m) ans[0] = v1[i];
45        else{
46            if(ans[m - x - 1] < v1[i]){
47                ans[m - x] = v1[i];
48            }
49        }
50    }
51    for(int i = 0; i < m; i ++){
52        cout << ans[i];
53        if(i != m) cout << " ";
54    }
55    cout << "\n";
56    return 0;
57}

```

累加和不大于k的最长子数组

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(){
4     ios::sync_with_stdio(false);

```

```

5     cin.tie(nullptr);
6     int n;
7     int k;
8     cin >> n >> k;
9     vector<int> v1(n, 0);
10    for(auto & x : v1) cin >> x;
11    vector<int> minl(n);
12    vector<int> minend(n);
13    minl[n - 1] = v1[n - 1];
14    minend[n - 1] = n - 1;
15    for(int i = n - 2; i >= 0; i --){
16        if(minl[i + 1] <= 0){
17            minl[i] = v1[i] + minl[i + 1];
18            minend[i] = minend[i + 1];
19        }else{
20            minl[i] = v1[i];
21            minend[i] = i;
22        }
23    }
24    int ans = 0;
25    for(int i = 0, suml = 0, r = 0; i < n; i ++){
26        while(r < n && suml + minl[r] <= k){
27            suml += minl[r];
28            r = minend[r] + 1;
29        }
30        if(r > i){
31            ans = max(ans, r - i);
32            suml -= v1[i];
33        }else r = i + 1;
34    }
35    cout << ans << "\n";
36    return 0;
37 }

```

DP

数位DP

```

1 // 数位dp https://www.luogu.com.cn/problem/P2602
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 int F(int n, int d){
6     int ans = 0;
7     for(int cur, temp = n, l, r = 1; temp; r *= 10, temp /= 10){
8         cur = temp % 10;
9         l = temp / 10;
10        if(d == 0) l--;
11        ans += l * r;
12        if(d < cur){
13            ans += r;
14        }else if(d == cur){
15            ans += n % r + 1;
16        }
17    }

```

```

18     return ans;
19 }
20 signed main(){
21     int l, r;
22     cin >> l >> r;
23     for(int i = 0; i < 10; i ++){
24         cout << F(r, i) - F(l - 1, i);
25         if(i != 9) cout << " ";
26     }
27     return 0;
28 }
29 // https://www.luogu.com.cn/problem/P3413
30 constexpr int mod = 1e9 + 7;
31 bool is(string s){
32     for(int i = 1, j; i < s.size(); i ++){
33         j = i - 2;
34         if((j >= 0 && s[j] == s[i]) || s[i] == s[i - 1]) return false;
35     }
36     return true;
37 }
38 // 统计0-n中所有不是回文数的数 f1表示是否选过数 f2表示是否可以任意选
39 int F(int il, string s, int l, int ll, bool f1, bool f2){
40     if(il == s.size()) return 1;
41     int ans = 0;
42     if(!f1){
43         ans = (ans + F(il + 1, s, -1, -1, false, true)) % mod;
44         if(!f2){
45             for(int i = 1; i < s[il] - '0'; i ++){
46                 if(i != l && i != ll){
47                     ans = (ans + F(il + 1, s, i, l, true, true)) % mod;
48                 }
49                 if(s[il] - '0' != l && s[il] - '0' != ll) ans = (ans + F(il + 1,
50 s, s[il] - '0', l, true, false)) % mod;
51             }else{
52                 int sum = 9;
53                 if(++il != s.size()) sum *= 9;
54                 while(++il < s.size()){
55                     sum = (sum * 8) % mod;
56                 }
57                 ans = (ans + sum) % mod;
58             }
59         }else{
60             if(!f2){
61                 for(int i = 0; i < s[il] - '0'; i ++){
62                     if(i != l && i != ll){
63                         ans = (ans + F(il + 1, s, i, l, true, true)) % mod;
64                     }
65                 }
66                 if(s[il] - '0' != l && s[il] - '0' != ll) ans = (ans + F(il + 1,
67 s, s[il] - '0', l, true, false)) % mod;
68             }else{
69                 int sum;
70                 if(ll == -1){
71                     sum = 9;
72                     while(++ il < s.size()){


```

```

72             sum = (sum * 8) % mod;
73         }
74     }else{
75         sum = 8;
76         while(++ i1 < s.size()){
77             sum = (sum * 8) % mod;
78         }
79     }
80     ans = (ans + sum) % mod;
81 }
82 }
83 return ans;
84 }
85 // 统计0-n中的所有数 f1表示是否选过数 f2表示是否可以任意选
86 int F2(int i1, string s, bool f1, bool f2){
87     if(i1 == s.size()) return 1;
88     int ans = 0;
89     if(!f1){
90         ans = (ans + F2(i1 + 1, s, false, true)) % mod;
91         if(!f2){
92             for(int i = 1; i < s[i1] - '0'; i ++){
93                 ans = (ans + F2(i1 + 1, s, true, true)) % mod;
94             }
95             ans = (ans + F2(i1 + 1, s, true, false)) % mod;
96         }else{
97             int sum = 9; // 不能为零
98             while(++ i1 < s.size()){
99                 sum = (sum * 10) % mod;
100            }
101            ans = (ans + sum) % mod;
102        }
103    }else{
104        if(!f2){
105            for(int i = 0; i < s[i1] - '0'; i ++){
106                ans = (ans + F2(i1 + 1, s, true, true)) % mod;
107            }
108            ans = (ans + F2(i1 + 1, s, true, false)) % mod;
109        }else{
110            int sum = 10; // 能为零
111            while(++ i1 < s.size()){
112                sum = (sum * 10) % mod;
113            }
114            ans = (ans + sum) % mod;
115        }
116    }
117    return ans;
118 }
119 signed main(){
120     ios::sync_with_stdio(false);
121     cin.tie(nullptr);
122     string s1, s2;
123     cin >> s1 >> s2;
124     int n1 = F(0, s1, -1, -1, false, false);
125     int n2 = F(0, s2, -1, -1, false, false);
126     int ans1 = (n2 - n1 + is(s1) + mod) % mod;
127     int n3 = F2(0, s1, false, false);

```

```

128     int n4 = F2(0, s2, false, false);
129     int ans2 = (n4 - n3 + 1 + mod) % mod;
130     cout << (ans2 - ans1 + mod) % mod << "\n";
131     return 0;
132 }

```

树形DP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // https://pintia.cn/problem-sets/1859417373569925120/exam/problems/type/7?
4 // problemSetProblemId=1859417373590896646
5 #define int long long
6 #define fr first
7 #define sc second
8 struct my{
9     int w, b;
10    priority_queue<int, vector<int>, greater<>> qw, qb;
11};
12 signed main(){
13     cin.tie(0) -> ios::sync_with_stdio(0);
14     int t = 1;
15     // cin >> t;
16     while (t--) {
17         int n;
18         cin >> n;
19         vector<int> color(n + 1), val(n + 1), root(n + 1, 0);
20         vector adj(n + 1, vector<int>());
21         for(int i = 1; i <= n; i++) {
22             int m;
23             cin >> color[i] >> val[i] >> m;
24             for(int j = 0; j < m; j++) {
25                 int v;
26                 cin >> v;
27                 adj[i].push_back(v);
28                 root[v] --;
29             }
30         }
31         int ans = 0;
32         // 传回可以被改色的节点
33         auto F = [&] (auto && f, int u) -> my {
34             int w1 = 0, b1 = 0;
35             priority_queue<int, vector<int>, greater<>> qw1, qb1;
36             for(auto v : adj[u]) {
37                 my temp = f(f, v);
38                 w1 += temp.w;
39                 b1 += temp.b;
40                 while(!temp.qw.empty()) {
41                     qw1.push(temp.qw.top());
42                     temp.qw.pop();
43                 }
44                 while(!temp.qb.empty()) {
45                     qb1.push(temp.qb.top());
46                     temp.qb.pop();
47                 }
48             }
49         }
50     }
51 }

```

```

48         // cout << u << " " << w1 << " " << b1 << "\n";
49         // priority_queue<int, vector<int>, greater<>> q11 = qw1, q22 =
50 qb1;
51         // cout << "w: ";
52         // while(!q11.empty()){
53             // cout << q11.top() << " ";
54             // q11.pop();
55         // }cout << "\n";
56         // cout << "b: ";
57         // while(!q22.empty()){
58             // cout << q22.top() << " ";
59             // q22.pop();
60         // }cout << "\n";
61         my temp;
62         priority_queue<int, vector<int>, greater<>> q1, q2;
63         temp.w = temp.b = 0;
64         if(color[u] == 0){
65             temp.w++;
66             q1.push(val[u]);
67         }else{
68             temp.b++;
69             q2.push(val[u]);
70         }
71         if(w1 != b1){
72             if(abs(w1 - b1) == 1){
73                 if(w1 > b1){
74                     q1.push(qw1.top());
75                 }else{
76                     q2.push(qb1.top());
77                 }
78             }else{
79                 while(abs(w1 - b1) >= 2){
80                     if(w1 > b1){
81                         ans += qw1.top();
82                         qw1.pop();
83                         w1 --, b1++;
84                     }else{
85                         ans += qb1.top();
86                         qb1.pop();
87                         w1 --, b1++;
88                     }
89                 }
90             }
91             temp.qw = q1, temp.qb = q2;
92             temp.w += w1, temp.b += b1;
93             return temp;
94         };
95         for(int i = 1; i <= n; i ++){
96             if(root[i] == 0){
97                 F(F, i);
98                 break;
99             }
100        }
101        cout << ans << "\n";
102    }

```

```
103     return 0;
104 }
```

区间DP

一般的区间DP是 $O(n^4)$ 复杂度，但可以在第三层时通过搜索优化达到 $O(n^3 \log n)$.

```
1 // https://ac.nowcoder.com/acm/contest/108298/I
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main(){
8     cin.tie(nullptr)->iostream::sync_with_stdio(false);
9     int t = 1;
10    cin >> t;
11    while(t --) {
12        int n;
13        cin >> n;
14        vector<int> v1(n + 1), pre(n + 1);
15        pre[0] = 0;
16        for(int i = 1; i <= n; i++) cin >> v1[i], pre[i] = pre[i - 1] + v1[i];
17        vector dp(n + 1, vector(n + 1, vector<pair<int,int>())));
18        for(int i = 1; i <= n; i++) dp[i][i].push_back({011, 011});
19        auto get = [&](int x, vector<pair<int,int>>& v) -> int {
20            int l = 0, r = v.size() - 1;
21            int mid, ans = -1;
22            while(l <= r) {
23                mid = l + r >> 1;
24                if(v[mid].sc <= x) {
25                    ans = v[mid].fr;
26                    l = mid + 1;
27                } else r = mid - 1;
28            }
29            return ans;
30        };
31        for(int len = 2, dif, val_l, val_r, cost, cost_l, cost_r; len <= n; len
32        ++) {
33            for(int l = 1, r = l + len - 1; r <= n; l++, r++) {
34                // 优化位置
35                for(int k = l; k < r; k++) {
36                    val_l = pre[k] - pre[l - 1], val_r = pre[r] - pre[k];
37                    dif = abs(val_l - val_r);
38                    int cost = min(val_l, val_r) * ceil(log2(val_l + val_r));
39                    // 快速搜索
40                    cost_l = get(dif, dp[l][k]), cost_r = get(dif, dp[k + 1]
41 [r]);
42                    if(len == n) {
43                        if(cost_l == -1 || cost_r == -1) cout << "-1 ";
44                        else cout << cost_l + cost + cost_r << " ";
45                        continue;
46                    }
47                    if(cost_l == -1 || cost_r == -1) continue;
48                    dp[l][r].push_back({cost_l + cost + cost_r, dif});
49                }
50            }
51        }
52    }
53 }
```

```

47         }
48         // 优化策略
49         sort(dp[1][r].begin(), dp[1][r].end(), [&](const pair<int,int>
50             & a, const pair<int,int> & b){
51             if(a.sc == b.sc) return a.fr < b.fr;
52             return a.sc < b.sc;
53         });
54         for(int i = 1; i < dp[1][r].size(); i++) dp[1][r][i].fr =
55             min(dp[1][r][i].fr, dp[1][r][i - 1].fr);
56     }
57 }
58 return 0;
59 }
```

Fliegel–Van Flandern 算法

这个算法可以将从公元前4713年1月1日（即儒略日JD=0）起计算的**儒略日数（JD）**转换为公历日期（年、月、日），并自动判断是在儒略历还是格里历中（1582年10月15日为切换点）。

- JD ≥ 2299161 : 采用**格里历**（即1582年10月15日之后）
- JD < 2299161 : 采用**儒略历**

```

1 // Fliegel–Van Flandern 算法
2 #include<bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 void julianDayToDate(ll JD, ll &day, ll &month, ll &year, bool &isBC) {
6     ll A, B, C, D, E;
7     if (JD >= 2299161) {
8         double alpha = floor((JD - 1867216.25) / 36524.25);
9         A = JD + 1 + alpha - floor(alpha / 4.0);
10    }
11    else {
12        A = JD;
13    }
14    B = A + 1524;
15    C = floor((double)(B - 122.1) / 365.25);
16    D = floor(365.25 * (double)C);
17    E = floor((double)(B - D) / 30.6001);
18    day = B - D - floor(30.6001 * E);
19    if (E < 14)
20        month = E - 1;
21    else
22        month = E - 13;
23
24    if (month > 2)
25        year = C - 4716;
26    else
27        year = C - 4715;
28
29    if (year <= 0) {
30        isBC = true;
```

```

31     day = day;
32     month = month;
33     year = -(year) + 1;
34 }
35 else {
36     isBC = false;
37 }
38 }
39 int main() {
40     ios::sync_with_stdio(false);
41     cin.tie(nullptr);
42     ll Q;
43     cin >> Q;
44     while(Q--) {
45         ll JD;
46         cin >> JD;
47         ll day, month, year;
48         bool isBC = false;
49         julianDayToDate(JD, day, month, year, isBC);
50         if(isBC) {
51             cout << day << " " << month << " " << year << " BC\n";
52         }
53     else{
54             cout << day << " " << month << " " << year << "\n";
55         }
56     }
57     return 0;
58 }
```

逆波兰式

逆波兰表达式（Reverse Polish Notation，简称 RPN），也称为后缀表达式，是一种将运算符放在操作数之后的数学表达方式。

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define mod1 1000000007
5 constexpr int N = 3e5 + 1;
6 vector<ll> stack1(N, 0);
7 int size1;
8 ll a;
9 ll b;
10 ll ans;
11 void solve(void) {
12     size1 = 0;
13     string s;
14     getline(cin, s);
15     istringstream ios(s);
16     string token;
17     while(ios >> token) {
18         if(isdigit(token[0])) {
19             stack1[size1++] = stoll(token);
20             // cout << stoll(token) << "\n";
21         }
22     }
23 }
```

```

21     } else{
22         b = stack1[--size1];
23         a = stack1[--size1];
24         // cout << a << " " << b << "\n";
25         if(token == "+") {
26             ans = (a + b) % mod1;
27         }else if(token == "-") {
28             ans = (a - b) % mod1;
29         }else if(token == "*") {
30             ans = (a * b) % mod1;
31         }else{
32             ans = (a / b) % mod1;
33         }
34         // cout << ans << "\n";
35         stack1[size1++] = ans;
36     }
37 }
38 cout << stack1[size1 - 1] << "\n";
39 }
```

约瑟夫环

约瑟夫环问题描述的是一群人围成一圈，每隔固定人数（即第 k 个人）被淘汰，直到最后剩下一个人。最后存活的人的位置可以通过下面的递推式来求解：

假设只有 1 个人时，存活的位置为 0 （注意：这里用 0 表示编号，从 0 开始）；当有 i 个人时，可以利用 $i-1$ 个人的结果递推出 i 个人的情况。

递推公式为：

$$f(i) = (f(i-1) + k) \bmod i$$

其中， $f(i)$ 表示 i 个人时最后存活者的编号。

$i = 1$ 时， $f[i]$ 显然等于 0

```

1 int josephus(int n, int k) {
2     int result = 0; // 初始只有一个人时，位置是 0
3     for (int i = 2; i <= n; i++) {
4         result = (result + k) % i;
5     }
6     return result;
7 }
```

离散化

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 #define fr first
5 #define sc second
6 signed main(){
7     cin.tie(0) -> ios::sync_with_stdio(0);
```

```

8     int t = 1;
9     // cin >> t;
10    while (t--) {
11        int n;
12        cin >> n;
13        vector<int> v1(n + 1), ord(n + 1);
14        for(int i = 1; i <= n; i++) {
15            cin >> v1[i];
16            ord[i] = v1[i];
17        }
18        sort(ord.begin() + 1, ord.end());
19        int m = 1;
20        for(int i = 2; i <= n; i++) {
21            if(ord[m] != ord[i]) {
22                ord[++m] = ord[i];
23            }
24        }
25        auto find = [&](int x) -> int {
26            int l = 1, r = m;
27            int mid, ans;
28            while(l <= r) {
29                mid = (l + r) >> 1;
30                if(x >= ord[mid]) {
31                    ans = mid;
32                    l = mid + 1;
33                } else {
34                    r = mid - 1;
35                }
36            }
37            return ans;
38        };
39        for(int i = 1; i <= n; i++) {
40            cout << "after: " << find(v1[i]) << "\n";
41        }
42    }
43    return 0;
44 }

```

N皇后问题

位运算版本

每次将状态进行左移和右移一位模拟对角线移动，而且利用位运算快速得到此行能够放置的位置

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n;
4 vector<array<int,11>> sols;
5 void dfs(int row, int colMask, int d1Mask, int d2Mask, array<int,11>& col) {
6     if (row == n) {
7         sols.push_back(col);
8         return;
9     }
10    int avail = ((1 << n) - 1) & ~ (colMask | d1Mask | d2Mask);

```

```

11     while (avail) {
12         int p = avail & ~avail;
13         int c = __builtin_ctz(p);
14         col[row] = c;
15         dfs(row + 1,
16             colMask | p,
17             (d1Mask | p) << 1,
18             (d2Mask | p) >> 1,
19             col);
20         avail ^= p;
21     }
22 }
23
24 int main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
27     cin >> n;
28     array<int, 11> col{};
29     dfs(0, 0, 0, 0, col);
30     long long ans = 0;
31     int S = sols.size();
32     cout << S << '\n';
33     return 0;
34 }
```

凸包

下凸包

```

1 // https://ac.nowcoder.com/acm/contest/108299/H
2 #include <bits/stdc++.h>
3 using namespace std;
4 // 使用 long long 防止溢出
5 #define int long long
6 #define fr first
7 #define sc second
8 // 最大节点数和最大边数，根据题目限制调整
9 constexpr int N = 1e5, M = 3e5;
10 // 邻接表存储
11 vector<int> head(N + 1), next1(M + 1), to(M + 1), weight(M + 1);
12 // 存储从 1 出发的最短距离、到 n 的最短距离
13 vector<int> d1(N + 1), d2(N + 1);
14 // 原始边信息
15 vector<int> u(M + 1), v(M + 1), w(M + 1), t1(M + 1);
16 // Dijkstra 标记用
17 vector<int> has(N + 1);
18 // 直线结构 y = k*x + b
19 struct line {
20     int k, b;
21 };
22 // 用于维护下凸包的直线数组
23 vector<line> l1(M + 1), ans1(M + 1);
24 signed main() {
25     ios::sync_with_stdio(false);
26     cin.tie(nullptr);
```

```

27     int T;
28     cin >> T;
29     while (T--) {
30         int n, m;
31         cin >> n >> m;
32         // 读入所有边
33         for (int i = 1; i <= m; i++) {
34             cin >> u[i] >> v[i] >> w[i] >> t1[i];
35         }
36         int cnt1 = 1;
37         // 邻接表插入函数
38         auto add = [&](int uu, int vv, int ww) {
39             next1[cnt1] = head[uu];
40             to[cnt1] = vv;
41             weight[cnt1] = ww;
42             head[uu] = cnt1++;
43         };
44         // Dijkstra 模板, in 表示起点
45         auto dj = [&] (vector<int> &d, int in) {
46             // 重置图和距离
47             fill(head.begin() + 1, head.begin() + n + 1, 0);
48             fill(has.begin() + 1, has.begin() + n + 1, 0);
49             fill(d.begin() + 1, d.begin() + n + 1, (int)1e18);
50             cnt1 = 1;
51             // 重新插入边
52             for (int i = 1; i <= m; i++) {
53                 add(u[i], v[i], w[i]);
54             }
55             // 最小堆
56             priority_queue<pair<int,int>, vector<pair<int,int>>,
57             greater<pair<int,int>>> pq;
58             d[in] = 0;
59             pq.push({0, in});
60             while (!pq.empty()) {
61                 auto [dist, u0] = pq.top();
62                 pq.pop();
63                 if (has[u0]) continue;
64                 has[u0] = 1;
65                 // 松弛操作
66                 for (int ei = head[u0]; ei; ei = next1[ei]) {
67                     int v0 = to[ei], wt = weight[ei];
68                     if (d[u0] + wt < d[v0]) {
69                         d[v0] = d[u0] + wt;
70                         pq.push({d[v0], v0});
71                     }
72                 }
73             };
74             // 计算从 1 到各点的最短路
75             dj(d1, 1);
76             // 反向图: 交换 u 和 v
77             for (int i = 1; i <= m; i++) swap(u[i], v[i]);
78             // 计算从 n 到各点的最短路 (即原图各点到 n 的最短路)
79             dj(d2, n);
80             // 恢复原向边
81             for (int i = 1; i <= m; i++) swap(u[i], v[i]);

```

```

82     // 构造每条边对应的一条直线 l: y = -t1*x + (d1[u] + d2[v] + w)
83     for (int i = 1; i <= m; i++) {
84         l1[i] = line{-t1[i], d1[u[i]] + d2[v[i]] + w[i]};
85     }
86     // 按斜率 k 降序排序, 若 k 相同, 按截距 b 降序
87     sort(l1.begin() + 1, l1.begin() + m + 1, [] (auto &A, auto &B) {
88         if (A.k == B.k) return A.b > B.b;
89         return A.k > B.k;
90     });
91     // 计算两条直线相交的 x 坐标 (向下取整)
92     auto getX = [&] (const line &A, const line &B) {
93         // A.k > B.k 保证分母非负
94         return (B.b - A.b) / (A.k - B.k);
95     };
96     // 维护下凸包
97     int len = 0;
98     for (int i = 1; i <= m; i++) {
99         // 去掉失效的或不必要的直线
100        while (len > 0) {
101            // 如果新线在上一条线之上, 则上一条线无效
102            if (l1[i].b <= ans1[len-1].b) len--;
103            // 或者相交点不递增, 也要去掉上一条
104            else if (len > 1 && getX(l1[i], ans1[len-1]) <= getX(ans1[len-1], ans1[len-2])) len--;
105            else break;
106        }
107        ans1[len++] = l1[i];
108    }
109    // 处理查询
110    int q; cin >> q;
111    while (q--) {
112        int x; cin >> x;
113        // 二分找在凸包上对应 x 的最佳直线
114        int l = 0, r = len - 2, best = 0;
115        while (l <= r) {
116            int mid = (l + r) >> 1;
117            if (getX(ans1[mid], ans1[mid+1]) < x) {
118                best = mid + 1;
119                l = mid + 1;
120            } else r = mid - 1;
121        }
122        // 输出答案
123        cout << ans1[best].k * x + ans1[best].b << "\n";
124    }
125 }
126 return 0;
127 }
```

FFT

核心思想

FFT (快速傅里叶变换) 的目标是高效计算两个多项式的卷积。
设

$$C(x) = A(x)B(x), \quad A(x) = \sum_{i=0}^n a_i x^i, \quad B(x) = \sum_{j=0}^m b_j x^j$$

那么卷积的系数是：

$$c_k = \sum_{i+j=k} a_i b_j$$

朴素算法是枚举 i, j , 复杂度 $O(n^2)$ 。FFT 的作用是利用 **离散傅里叶变换 (DFT)** 的性质, 把这个卷积问题降到 $O(n \log n)$ 。

离散傅里叶变换 (DFT) 与逆变换

给定序列 a_0, a_1, \dots, a_{n-1} , 定义 DFT:

$$A_k = \sum_{j=0}^{n-1} a_j \cdot \omega_n^{jk}, \quad k = 0, \dots, n-1$$

其中

$$\omega_n = e^{2\pi i/n}$$

是 n 次单位根。

逆变换公式为:

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} A_k \cdot \omega_n^{-jk}$$

性质:

- 在点值表示下, 多项式乘法等价于卷积。
- 即: 在频域做乘法, 等价于时域做卷积。

卷积定理

1. 把 $A(x), B(x)$ 变换到频域:

得到 $\{A(\omega_n^k)\}, \{B(\omega_n^k)\}$

2. 点值相乘:

$$C(\omega_n^k) = A(\omega_n^k) \cdot B(\omega_n^k)$$

3. 逆变换:

得到 $C(x)$ 的系数。

因此，卷积过程就是：

FFT(A) → FFT(B) → 点乘 → IFFT。

总复杂度 $O(n \log n)$ 。

FFT 加速

如果直接算 DFT，需要 $O(n^2)$ 。

FFT 利用了**分治思想**：

$$A(x) = A_{even}(x^2) + xA_{odd}(x^2)$$

代入 ω_n^k 得到：

$$A(\omega_n^k) = A_{even}(\omega_{n/2}^k) + \omega_n^k A_{odd}(\omega_{n/2}^k)$$

这说明一个 n 点 DFT 可以分解为两个 $n/2$ 点 DFT，再加上合并时的若干乘法。
递归展开，复杂度降为 $O(n \log n)$ 。

在代码层面，分治被转化为**迭代形式**：

1. **位逆序重排**：先将数组下标按二进制反转排序。

2. **逐层蝴蝶操作**：从长度 2 的小段开始，不断倍增，每一层都用旋转因子 ω 做合并。

蝴蝶操作形式为：

$$u = a[i], \quad v = w \cdot a[i + mid]$$

$$a[i] = u + v, \quad a[i + mid] = u - v$$

这就是代码中循环的核心。

代码关键点对应数学

- `for (int i = 0; i < tot; i++)`：遍历 FFT 长度（`tot` 为最近的 2 次幂）。
- 位逆序数组 `rev[i]`：保证每一层分治合并时顺序正确。
- 每一层长度 `len` 翻倍，旋转因子 `wn = exp(±2πi/len)`。
- `if (op == 1)` 正变换 (FFT)，`if (op == -1)` 逆变换 (IFFT)。
- 逆变换最后要除以 `tot`，对应公式里的 $1/n$ 。
- `<tot` 而不是 `<=tot`：因为数组下标从 0 到 `tot-1`，总共 `tot` 个点。

在题目中的作用

在这道题里，给定两个多项式（或 01 串映射为系数数组），需要快速计算卷积。
FFT 的流程是：

1. 读入序列，扩充到 2 次幂长度。

2. FFT 变换。

3. 点值相乘。

4. IFFT 还原，得到卷积结果。

5. 再做题目所需的“进制归约”（例如 $\sqrt{-2}$ 进位）。

```
1 // https://ac.nowcoder.com/acm/contest/108305/J
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 constexpr int N = 3e5;
8 const double PI = acos(-1);
9 // 简单复数结构，支持 +,-,*
10 struct Complex{
11     double x, y;
12     Complex operator +(const Complex &t) const { return {x + t.x, y + t.y}; }
13     Complex operator -(const Complex &t) const { return {x - t.x, y - t.y}; }
14     Complex operator *(const Complex &t) const { return {x * t.x - y * t.y, x * t.y + y * t.x}; }
15 };
16 // 工作数组
17 vector<Complex> a(N + 1), b(N + 1);
18 vector<int> rev(N + 1), res(N << 1 | 1);
19 int bit, tot; // FFT 长度信息
20 signed main(){
21     cin.tie(nullptr)->ios_base::sync_with_stdio(false);
22     int t = 1;
23     cin >> t;
24     // FFT: inv=1 正变换, inv=-1 逆变换
25     auto FFT = [&](vector<Complex> &v, int inv) {
26         for(int i = 0; i < tot; i++) if(i < rev[i]) swap(v[i], v[rev[i]]);
27         for(int mid = 1; mid < tot; mid <<= 1) {
28             Complex w1 = {cos(PI / mid), inv * sin(PI / mid)};
29             for(int i = 0; i < tot; i += (mid << 1)) {
30                 Complex wk = {1, 0};
31                 for(int j = 0; j < mid; j++, wk = wk * w1) {
32                     Complex x = v[i + j], y = wk * v[i + j + mid];
33                     v[i + j] = x + y;
34                     v[i + j + mid] = x - y;
35                 }
36             }
37         }
38     };
39     while(t --) {
40         string s1, s2;
41         cin >> s1 >> s2;
42         int n1 = s1.size(), n2 = s2.size();
43         // 低位在前存储
44         for(int i = 0; i < n1; i++) a[i] = {double(s1[n1 - i - 1] - '0'), 0};
45         for(int i = 0; i < n2; i++) b[i] = {double(s2[n2 - i - 1] - '0'), 0};
46         // 取最近 2 次幂长度
47         bit = 0;
48         while((1 << bit) < n1 + n2 - 1) bit++;
49         tot = 1 << bit;
```

```

50     // 补 0
51     for(int i = n1; i < tot; i++) a[i] = {0, 0};
52     for(int i = n2; i < tot; i++) b[i] = {0, 0};
53     // 位逆序表
54     for(int i = 0; i < tot; i++)
55         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
56     // FFT → 逐点相乘 → IFFT
57     FFT(a, 1), FFT(b, 1);
58     for(int i = 0; i < tot; i++) a[i] = a[i] * b[i];
59     FFT(a, -1);
60     // 卷积结果 (未进位)
61     int len = 0;
62     for (int i = 0; i < n1 + n2 - 1; ++i)
63         res[i] = (int)(a[i].x / tot + 0.5);
64     // √-2 进制进位归约
65     for(len = 0; len < n1 + n2 - 1 || res[len + 1] || res[len]; len++) {
66         if(res[len] < 0) {
67             res[len] = -res[len];
68             res[len + 2] += res[len];
69         }
70         if(res[len] > 0) {
71             res[len + 2] -= res[len] >> 1;
72             res[len] %= 2;
73         }
74     }
75     // 去掉前导 0
76     while(len > 1 && !res[len - 1]) len--;
77     // 输出 (高位在前)
78     for(int i = len - 1; i >= 0; i--) cout << char('0' + res[i]);
79     cout << "\n";
80     // 清理
81     fill(res.begin(), res.begin() + len + 1, 0);
82 }
83 return 0;
84 }
```

STL

set

基于红黑树实现的自动排序、无重复元素的关联容器

```

1 #include <set>
2 using namespace std;
3 set<int> s;                      // 升序, 默认 less<int>
4 set<string> names;
5 set<int, greater<int>> s2;      // 降序排列
6 s.insert(5);          // 插入 5, 若已存在则无效
7 s.erase(5);          // 删除值为5的元素
8 s.erase(it);          // 通过迭代器删除
9 s.erase(s.begin(), s.end()); // 清空
```

```

10 auto it = s.find(5);      // 找到返回迭代器，否则为 s.end()
11 if (it != s.end()) cout << *it << " found\n";
12 if (s.count(5)) cout << "存在";
13 s.size();
14 s.empty();
15 s.clear();
16 for (auto it = s.begin(); it != s.end(); ++it)
17     cout << *it << " ";
18 for (int x : s)
19     cout << x << " ";
20 auto it1 = s.lower_bound(5); // 返回  $\geq 5$  的第一个元素迭代器
21 auto it2 = s.upper_bound(5); // 返回  $> 5$  的第一个元素迭代器
22 struct cmp {
23     bool operator()(const pair<int,int>& a, const pair<int,int>& b) const {
24         return a.first < b.first; // 可定义复合排序规则
25     }
26 };
27 set<pair<int,int>, cmp> s;

```

插入/删除/查找都是 $O(\log n)$

ordered_set

解释各个部分的作用

代码部分	功能	说明
s.insert(5)	插入元素	ordered_set 自动去重
s.erase(7)	删除元素	需先检查元素是否存在
find_by_order(k)	获取第 k 小元素	0-based，需先检查是否为空
order_of_key(10)	获取小于某个值的元素数量	$O(\log n)$ 复杂度
find(5)	查找元素	直接返回迭代器
s.size()	获取集合大小	$O(1)$ 复杂度
s.clear()	清空集合	直接清空

时间复杂度

- 插入: $O(\log n)$
- 删 除: $O(\log n)$
- 查 找: $O(\log n)$
- 获取第 k 小元素: $O(\log n)$
- 获取小于 x 的元素数量: $O(\log n)$

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 using namespace __gnu_pbds;
5 using namespace std;
6 typedef tree<
7     int,
8     null_type,

```

```

9     less<int>,
10    rb_tree_tag,
11    tree_order_statistics_node_update
12 > ordered_set;
13
14 using namespace std;
15
16 int main() {
17     ordered_set s;
18     s.insert(5);
19     if (s.find(7) != s.end()) s.erase(7);
20     if (s.size() > 1) {
21         cout << "The 2nd smallest element is: " << *s.find_by_order(1) << '\n';
22     }
23     if (s.find(5) != s.end()) {
24         cout << "5 exists in the set\n";
25     }
26     if (!s.empty()) {
27         cout << "Minimum element: " << *s.find_by_order(0) << '\n';
28         cout << "Maximum element: " << *s.find_by_order(s.size() - 1) << '\n';
29     }
30     s.clear();
31     return 0;
32 }
```

string

数值与字符串转化

函数签名	作用说明
to_string(val)	把 val 转换成 string
stoi(s, p, b)	把字符串 s 从下标 p 开始转换成 b 进制的 int
stol(s, p, b)	转换成 long
stoul(s, p, b)	转换成 unsigned long
stoll(s, p, b)	转换成 long long
stoull(s, p, b)	转换成 unsigned long long
stof(s, p, b)	转换成 float
stod(s, p, b)	转换成 double
stold(s, p, b)	转换成 long double

p 和 b 可省，默认为十进制

字符串输入与分割

```

getline(cin, s)

stringstream s(ss)

1 | string s1;
2 | getline(cin, s1);
```

```

3 vector<int> v1;
4 stringstream ss(s1);
5 string temp;
6 // while(ss >> temp) 按空格读取
7 while(getline(ss, temp, ', ')){
8     if(temp[0] == '-'){
9         temp = temp.substr(1);
10    v1.push_back(-stoi(temp));
11 } else {
12    v1.push_back(stoi(temp));
13 }
14 }
```

改变大小写

```

1 transform(起点, 终点, 输出位置, 处理函数)
2 // 变小写
3 transform(s1.begin(), s1.end(), s1.begin(), ::tolower);
4 // 变大写
5 transform(s1.begin(), s1.end(), s1.begin(), ::toupper);
6 // 转换到另一个字符串
7 string src = "Hello";
8 string dest;
9 dest.resize(src.size());
10 transform(src.begin(), src.end(), dest.begin(), ::tolower);
11 // dest 现在是 "hello", 原始 src 不变
```

子串 substr

substr(st, len)

len省略时表示到en

查找

find

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int main()
4 {
5     ios::sync_with_stdio(false);
6     std::string str ("There are two needles in this haystack with needles.");
7     std::string str2 ("needle");
8     // 在str当中查找第一个出现的needle，找到则返回出现的位置，否则返回结尾
9     std::size_t found = str.find(str2);
10    if (found!=std::string::npos) // 结尾
11        // 在str当中，从第found+1的位置开始查找参数字符串的前6个字符
12        found = str.find("needles are small", found+1, 6);
13 }
```

```
rfind
1 //rfind是找最后一个出现的匹配字符串
2 std::size_t found = str.rfind(str2);
```

```
find_...of
1 find_first_of(args) // **查找args中任何一个字符第一次出现的位置**
2 find_last_of(args) // **最后一个出现的位置**
3 find_fist_not_of(args) // **查找第一个不在args中的字符**
4 find_last_not_of // **查找最后一个不在args中出现的字符**
```

插入insert

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     ios::sync_with_stdio(false);
5     string str="to be question";
6     string str2="the ";
7     string str3="or not to be";
8     string::iterator it;
9     //s.insert(pos,str)//在s的pos位置插入str
10    str.insert(6,str2);
11    //s.insert(pos,str,a,n)在s的pos位置插入str中插入位置a到后面的n个字符
12    str.insert(6,str3,3,4);
13    //s.insert(pos,cstr,n)//在pos位置插入cstr字符串从开始到后面的n个字符
14    str.insert(10,"that is cool",8);
15    //s.insert(pos,n,ch)在s.pos位置上面插入n个ch
16    str.insert(15,1,':');
17    //s.insert(s.it,ch)在s的it指向位置前面插入一个字符ch, 返回新插入的位置的迭代器
18    it = str.insert(str.begin()+5,',');
19    //s.insert(s.it,n,ch)//在s的it所指向位置的前面插入n个ch
20    str.insert(str.end(),3,'.');
21    //s.insert(it,str.ita,str.itb)在it所指向的位置的前面插入[ita,itb)的字符串
22    str.insert(it+2,str3.begin(),str3.begin()+3);
23    return 0;
24 }
```

map & unordered_map

map基于红黑树（或其他平衡二叉树）实现，所有元素按照key的大小有序存储。

unordered_map基于哈希表实现，元素按照哈希桶无序存储。

操作	map	unordered_map
查找/插入/删除	O(logN) (严格)	平均O(1),最坏O(N) (哈希冲突)

| 如果对 key 的顺序有需求，或数据量不极端、性能要求严格稳定，就用 map。

如果数据随机分布、对遍历顺序无要求，希望获得极快的平均查找速度，可用 `unordered_map`，但要注意：

1. **适当 `reserve`**：提前 `reserve(bucket_count)` 或 `max_load_factor` 来减少扩容；
2. **自定义哈希**：在 CF 等平台上可用 `splittmix64` 等更安全的哈希函数，防止碰撞攻击。

cout

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int main() {
5     int num = 255;
6     double pi = 3.1415926;
7     bool flag = true;
8     // 布尔输出
9     cout << boolalpha << "bool: " << flag << " " << false << endl;
10    // 进制
11    cout << "dec: " << dec << num
12        << " hex: " << hex << num
13        << " oct: " << oct << num << endl;
14    cout << dec; // 恢复十进制
15    // 浮点数格式
16    cout << fixed << setprecision(2) << "fixed: " << pi << endl;
17    cout << scientific << setprecision(3) << "sci: " << pi << endl;
18    cout.unsetf(ios::floatfield);
19    // 宽度、对齐、填充
20    cout << setfill('-') << left << setw(10) << "Left" << right << setw(10) <<
21 "Right" << endl;
22    cout << setfill('_') << right << setw(10) << "hello\n";
23    // 显示正号
24    cout << showpos << 123 << " " << -123 << endl;
```

tips

重载运算符号（比较）

```
1 // 结构体
2 struct my{
3     int l, r, index;
4     my(int a, int b, int c){
5         l = a, r = b, index = c;
6     }
7     bool operator < (const my x) const {
8         // 优先队列中比较关系反着来
9         return l > x.l;
```

```

10     }
11 };
12 // lambda (示例为小根堆)
13 auto cmp = [] (pair<int, int> x, pair<int, int> y) {
14     return x.second > y.second;
15 };
16 priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(cmp)> q(cmp);

```

求最大的最小或者最小的最大问题

大概率是用二分来查询最合适的结果，具体算法看题

圆周率π

```

1 const double pi = acos(-1);

```

求容器最大值

```

1 *max_element(vec.begin(), vec.end());

```

全排列

```

1 std::string s = "aba";
2 do{
3     std::cout << s << '\n';
4 }while (std::next_permutation(s.begin(), s.end()));
5 std::cout << s << '\n';

```

求二维平面上的三角形面积

给定三个顶点

```

1 S = abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2;
2 // or
3 S = abs((x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1)) / 2;
4 // 两个参数相较于三个参数，有如下优势，可以取一个坐标为基准（2坐标与1坐标，3坐标与1坐标，每个参数都进行了一次碰撞），进行点与点之间的逻辑处理，如判断处坐标异或值

```

给定三条边

```

1 p = (a + b + c) / 2;
2 S = sqrt(p * (p - a) * (p - b) * (p - c));

```

求三维空间中两条直线最短距离

L1 经过 A(x1, y1, z1), B(x2, y2, z2)

L2 经过 C(x3, y3, z3), D(x4, y4, z4)

L_1 的方向向量 $d_1 = B - A$, L_2 的方向向量 $d_2 = D - C$

$n = d_1 \times d_2$

如果 n 为 0, 则两直线平行

任选 L_1 上的点 A 和 L_2 上的点 C , 计算 $AC = C - A$

不平行时, 两直线最短距离 $d = |AC| \cdot \|n\| / \|n\|$

平行时, $d = |AC| * |d_1| / |d_1|$

不平行情况

若 $\vec{d}_1 \times \vec{d}_2 \neq 0$, 则最短距离公式为

$$d = \frac{|(C - A) \cdot (\vec{d}_1 \times \vec{d}_2)|}{\|\vec{d}_1 \times \vec{d}_2\|}.$$

其中:

$(C - A)$ 表示向量, \cdot 为点乘, \times 为叉乘。

该公式的几何意义是: $\vec{d}_1 \times \vec{d}_2$ 垂直于两条直线, $(C - A)$ 在此垂直方向上的投影长度即为两条直线的最短距

平行情况

若 $\vec{d}_1 \times \vec{d}_2 = 0$, 则说明两条直线平行或重合。此时最短距离可以由下式给出:

$$d = \frac{\|(C - A) \times \vec{d}_1\|}{\|\vec{d}_1\|}.$$

如果出现退化情况 (例如 $A=B$ 或 $C=D$) , 需要进一步做特殊处理。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 #define fr first
5 #define sc second
6 // baidu 14 2
7 struct point{
8     double x, y, z;
9     point(double x1 = 0, double y1 = 0, double z1 = 0): x(x1), y(y1), z(z1) {}
10};
11 signed main(){
12     cin.tie(0) -> ios::sync_with_stdio(0);
13     int t = 1;
14     cin >> t;
15     while (t--) {
```

```

16     double x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4;
17     cin >> x1 >> y1 >> z1 >> x2 >> y2 >> z2 >> x3 >> y3 >> z3 >> x4 >> y4
18     >> z4;
19     point a = point{x1, y1, z1}, b = {x2, y2, z2}, c = {x3, y3, z3}, d =
20     {x4, y4, z4};
21     // 向量减法: 返回 b - a
22     auto vecSub = [] (const point & a, const point & b) -> point {
23         point temp = point(b.x - a.x, b.y - a.y, b.z - a.z);
24         return temp;
25     };
26     // 向量叉乘 (仅适用于 3D)
27     auto crossProduct = [] (const point & a, const point & b) -> point{
28         point temp = point(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
29         a.x * b.y - a.y * b.x);
30         return temp;
31     };
32     // 向量点乘
33     auto dotProduct = [] (const point & a, const point & b) -> double {
34         return a.x * b.x + a.y * b.y + a.z * b.z;
35     };
36     // 向量的模长
37     auto norm = [] (const point & a) -> double {
38         return sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
39     };
40     point d1 = vecSub(a, b), d2 = vecSub(c, d);
41     point n = crossProduct(d1, d2);
42     double norm_n = norm(n);
43     point AC = vecSub(c, a);
44     double distance = 0.0;
45     // 如果 n != 0, 说明两条直线不平行
46     if(fabs(norm_n) > 1e-12){
47         // 最短距离公式: |(C - A) · (d1 x d2)| / ||d1 x d2||
48         distance = fabs(dotProduct(AC, n)) / norm_n;
49     }else{
50         // n = 0, 说明两条直线平行或共线
51         // 使用平行直线距离公式: ||(C - A) x d1|| / ||d1||
52         point temp1 = crossProduct(AC, d1);
53         double norm_temp1 = norm(temp1), norm_d1 = norm(d1);
54         if(fabs(norm_d1) < 1e-12){
55             // 若 d1Norm=0, 说明 A、B 是同一点; 若 d2Norm=0, C、D 同理。此时需特殊
56             // 处理
57             double norm_d2 = norm(d2);
58             if(fabs(norm_d2) < 1e-12){
59                 // 两条“直线”都是点, 计算 A 到 C 的距离
60                 distance = sqrt(pow(a.x - c.x, 2) + pow(a.y - c.y, 2) +
61                 pow(a.z - c.z, 2));
62             }else{
63                 // 只需计算点 A 到直线 CD 的距离
64                 // 距离 = ||(C - A) x d2|| / ||d2||
65                 point temp2 = crossProduct(AC, d2);
66                 distance = norm(temp2) / norm(d2);
67             }
68         }else{
69             distance = norm(temp1) / norm(d1);
70         }
71     }
72 }
```

```

67         printf("%.3lf\n", distance);
68     }
69     return 0;
70 }
```

点到直线的距离公式 & 关系 (三点之间)

设 $P(x_0, y_0)$ 为平面上一点， 直线 L 的方程为

$$Ax + By + C = 0.$$

则 P 到直线 L 的距离 d 为

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}.$$

平面上三个点 A(x_1, y_1), B(x_2, y_2), C(x_3, y_3), 判断点 C 与 \overrightarrow{AB} 的位置关系

$$S(A, B, C) = \frac{(x_1 - x_3)(y_2 - y_3) - (y_1 - y_3)(x_2 - x_3)}{2}$$

若 S 大于 0，则 C 在矢量 AB 的左侧

若 S 小于 0，则 C 在矢量 AB 的右侧

若 S 等于 0，则 C 在直线 AB 上。

随机数

```

1 | srand((unsigned)time(NULL));
2 | cout << (rand()) % 17;
```

完全平方数

一个数为完全平方数当且仅当该分解后该数所有质因数个数均为偶数

随机哈希编码

```

1 | mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
2 | int a_hash = rng();
```

批量操作

```
1 int a[5];
2 // 连续填充
3 iota(a, a + 5, 1); // a = {1, 2, 3, 4, 5}
4 // 适合设置为0、-1、255（即所有字节一致的值）
5 memset(a, 0, sizeof(a)); // 将数组 a 中的所有字节设为 0
6 // 支持所有基础类型和STL容器
7 vector<int> v;
8 fill(a, a + 5, 42); // a = {42, 42, 42, 42, 42}
9 fill(v.begin(), v.end(), 42);
```

lower_bound & upper_bound

lower_bound()和upper_bound()都是利用二分查找的方法在一个排好序的数组中进行查找的。

在从小到大的排序数组中,

- lower_bound(begin, end, value)

返回第一个 \geq value 的位置

- upper_bound(begin, end, value)

返回第一个 $>$ value 的位置

在从小到大的排序数组中,

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5 int main() {
6     vector<int> v = {-1, 1, 2, 4, 7, 15, 34};
7     sort(v.begin() + 1, v.end());
8
9     int pos1 = lower_bound(v.begin() + 1, v.end(), 7) - v.begin();
10    int pos2 = upper_bound(v.begin() + 1, v.end(), 7) - v.begin();
11    cout << pos1 << " " << v[pos1] << endl;
12    cout << pos2 << " " << v[pos2] << endl;
13
14    sort(v.begin() + 1, v.end(), greater<int>());
15    int pos3 = lower_bound(v.begin() + 1, v.end(), 7, greater<int>()) -
16        v.begin();
17    int pos4 = upper_bound(v.begin() + 1, v.end(), 7, greater<int>()) -
18        v.begin();
19    cout << pos3 << " " << v[pos3] << endl;
20    cout << pos4 << " " << v[pos4] << endl;
21    return 0;
22 }
```

折尺定理

一堆有各自长度($a_1, a_2, a_3 \dots$)木棍能否连接在两个点上，取决于是否 $L_{\min} \leq D \leq L_{\max}$, 其中,
 $L_{\max} = \sum_{i=1}^n a_i$, $L_{\min} = \max(0, 2 * a_{\max} - L_{\max})$, D 为两点间距离

两两绝对值求和

当你把一组数排序后，对所有「两两差的绝对值之和」 $\sum_{0 \leq p < q \leq n} |a_q - a_p|$ 有一个经典结论：

$$\sum_{0 \leq p < q \leq n} (a_q - a_p) = \sum_{k=0}^n a_k (k - (n - k))$$

这是因为：

- 对于固定的 a_k , 它作为“右端” q_q 的时候, 会和前面所有 k 个更小的数 a_0, \dots, a_{k-1} 相减, 累计贡献 $+ a_k \times k$ 。
 - 同时它也会作为“左端” a_p 被后面所有 $(n-k)$ 个更大的数 a_{k+1}, \dots, a_n 相减, 贡献 $- a_k \times (n - k)$ 。
 - 合并起来, 就是
- $$a_k(k - (n - k)) = a_k(2k - n).$$

注意这里不需要再写绝对值了, 因为排序保证了“后者减前者”总是非负的。

ASCII Table

ASCII Table

0 0x00 0000 0000 NUL	1 0x01 0000 0001 SOH	2 0x02 0000 0010 STX	3 0x03 0000 0011 ETX	4 0x04 0000 0100 EOT	5 0x05 0000 0101 ENQ	6 0x06 0000 0110 ACK	7 0x07 0000 0111 BEL	8 0x08 0000 1000 BS	9 0x09 0000 1001 HT	10 0x0a 0000 1010 LF \n	11 0x0b 0000 1011 VT	12 0x0c 0000 1100 FF	13 0x0d 0000 1101 CR \r	14 0x0e 0000 1110 SO	15 0x0f 0000 1111 SI
16 0x10 0001 0000 DLE	17 0x11 0001 0001 DC1	18 0x12 0001 0011 DC2	19 0x13 0001 0011 DC3	20 0x14 0001 0100 DC4	21 0x15 0001 0110 NAK	22 0x16 0001 0111 SYN	23 0x17 0001 0111 ETB	24 0x18 0001 1000 CAN	25 0x19 0001 1001 EM	26 0x1a 0001 1010 SUB	27 0x1b 0001 1011 ESC	28 0x1c 0001 1100 FS	29 0x1d 0001 1101 GS	30 0x1e 0001 1110 RS	31 0x1f 0001 1111 US
32 0x20 0010 0000 SPACE	33 0x21 0010 0001 !	34 0x22 0010 0010 "	35 0x23 0010 0011 #	36 0x24 0010 0100 \$	37 0x25 0010 0101 %	38 0x26 0010 0110 &	39 0x27 0010 0111 (40 0x28 0010 1000)	41 0x29 0010 1001 *	42 0x2a 0010 1010 +	43 0x2b 0010 1011 ,	44 0x2c 0010 1100 -	45 0x2d 0010 1101 .	46 0x2e 0010 1110 /	47 0x2f 0010 1111 ?
48 0x30 0011 0000 0	49 0x31 0011 0001 1	50 0x32 0011 0010 2	51 0x33 0011 0011 3	52 0x34 0011 0100 4	53 0x35 0011 0101 5	54 0x36 0011 0110 6	55 0x37 0011 0111 7	56 0x38 0011 1000 8	57 0x39 0011 1001 9	58 0x3a 0011 1010 :	59 0x3b 0011 1011 <br;">;</br;">	60 0x3c 0011 1100 <	61 0x3d 0011 1101 =	62 0x3e 0011 1110 >	63 0x3f 0011 1111 ?
64 0x40 0100 0000 @	65 0x41 0100 0001 A	66 0x42 0100 0010 B	67 0x43 0100 0011 C	68 0x44 0100 0100 D	69 0x45 0100 0101 E	70 0x46 0100 0110 F	71 0x47 0100 0111 G	72 0x48 0100 1000 H	73 0x49 0100 1001 I	74 0x4a 0100 1010 J	75 0x4b 0100 1011 K	76 0x4c 0100 1100 L	77 0x4d 0100 1101 M	78 0x4e 0100 1110 N	79 0x4f 0100 1111 O
80 0x50 0101 0000 P	81 0x51 0101 0001 Q	82 0x52 0101 0010 R	83 0x53 0101 0011 S	84 0x54 0101 0100 T	85 0x55 0101 0101 U	86 0x56 0101 0110 V	87 0x57 0101 0111 W	88 0x58 0101 1000 X	89 0x59 0101 1001 Y	90 0x5a 0101 1010 Z	91 0x5b 0101 1011 [92 0x5c 0101 1100 \\	93 0x5d 0101 1101]	94 0x5e 0101 1110 ^	95 0x5f 0101 1111 -
96 0x60 0110 0000 0	97 0x61 0110 0001 a	98 0x62 0110 0010 b	99 0x63 0110 0011 c	100 0x64 0110 0100 d	101 0x65 0110 0101 e	102 0x66 0110 0110 f	103 0x67 0110 0111 g	104 0x68 0110 1000 h	105 0x69 0110 1001 i	106 0x6a 0110 1010 j	107 0x6b 0110 1011 k	108 0x6c 0110 1100 l	109 0x6d 0110 1101 m	110 0x6e 0110 1110 n	111 0x6f 0110 1111 o
112 0x70 0111 0000 p	113 0x71 0111 0001 q	114 0x72 0111 0010 r	115 0x73 0111 0011 s	116 0x74 0111 0100 t	117 0x75 0111 0101 u	118 0x76 0111 0110 v	119 0x77 0111 0111 w	120 0x78 0111 1000 x	121 0x79 0111 1001 y	122 0x7a 0111 1010 z	123 0x7b 0111 1011 {	124 0x7c 0111 1100 	125 0x7d 0111 1101 }	126 0x7e 0111 1110 ~	127 0x7f 0111 1111 DEL

优先级

附录 B 运算符和结合性

优先级	运算符	含 义	结合方向
1	::		
2	() [] -> . . ++ --	域运算符 括号, 函数调用 数组下标运算符 指向成员运算符 成员运算符 自增运算符(后置)(单目运算符) 自减运算符(后置)(单目运算符)	自左至右
3	++ -- ~ ! - + * & (类型) sizeof new delete	自增运算符(前置) 自减运算符(前置) 按位取反运算符 按位取反运算符 逻辑非运算符 负号运算符 正号运算符 指针运算符 取地址运算符 类型转换运算符 长度运算符 动态分配空间运算符 释放空间运算符 (以上为单目运算符)	自右至左
4	*	乘法运算符	
	/	除法运算符	
	%	求余运算符	自左至右
5	+	加法运算符	
	-	减法运算符	自左至右
6	<< >>	按位左移运算符 按位右移运算符	自左至右
7	< <= > >=	关系运算符	自左至右
8	== !=	等于运算符 不等于运算符	自左至右
9	&	按位与运算符	自左至右
10	^	按位异或运算符	自左至右
11		按位或运算符	自左至右
12	&&	逻辑与运算符	自左至右
13		逻辑或运算符	自左至右
14	? :	条件运算符(三目运算符)	自右至左
15	= += -= *= /= %= >>= <<= &= ^ = !=	赋值运算符	自右至左
16	throw	抛出异常运算符	自右至左
17	,	逗号运算符	自左至右

规模

问题规模和可用算法

	logn	n	n*logn	n*根号n	n^2	2^n	n!
n <= 11	Yes	Yes	Yes	Yes	Yes	Yes	Yes
n <= 25	Yes	Yes	Yes	Yes	Yes	Yes	No
n <= 5000	Yes	Yes	Yes	Yes	Yes	No	No
n <= 10^5	Yes	Yes	Yes	Yes	No	No	No
n <= 10^6	Yes	Yes	Yes	No	No	No	No
n <= 10^7	Yes	Yes	No	No	No	No	No
n >= 10^8	Yes	No	No	No	No	No	No

隔板法 (Stars and Bars)

无界非负整数组合

- 题型：求非负整数解 $x_1 + \dots + x_k = N$ 的方案数
- 答案： $\binom{N+k-1}{k-1}$
- 含义：把 N 个「星星」放入 k 个「盒子」，盒子间用 $k-1$ 道隔板分隔。

单调序列差分

非增/非减序列 ($a_1 \geq a_2 \geq \dots \geq a_k \geq 0$) 可转为差分 $d_i = a_i - a_{i-1}$, 再用隔板法算
 $\sum d_i = a_k - a_1$

普通生成函数 (OGF)

定义

- 给定序列 $\{a_n\}$, 其 OGF 为

$$A(t) = \sum_{n \geq 0} a_n t^n.$$

乘法对应「独立事件的卷积」

- 若要把两类结构独立拼接, 总方案数的序列是两生成函数相乘:

$$A(t) \times B(t) \implies [t^n](A(t)B(t)) = \sum_{i+j=n} a_i b_j.$$

幂运算

- $A(t)^k$ 表示独立选取 k 份相同结构并联。

广义二项式与系数提取

广义二项式定理

$$(1-t)^{-\alpha} = \sum_{n=0}^{\infty} \binom{\alpha+n-1}{n} t^n, \quad \binom{\alpha+n-1}{n} = \frac{\alpha(\alpha+1)\cdots(\alpha+n-1)}{n!}.$$

系数提取

- 对于已知闭式 $G(t)$, 常用:

$$[t^n] G(t) \quad \text{或} \quad \text{前缀和 } \sum_{i=0}^n [t^i] G(t).$$

累加恒等式

$$\sum_{i=0}^m \binom{i+r}{r} = \binom{m+r+1}{r+1}, \quad \sum_{i=0}^m \binom{r}{i} = \binom{r}{0} + \cdots + \binom{r}{m}.$$

曼哈顿距离转换（菱形转矩形）

原菱形覆盖（曼哈顿距离 \leq ）在 (i,j) 平面上难以直接判断，另 $u = i+j, v = i-j$, 则菱形在 (u, v) 空间投影为轴对齐矩形

逆序对

一个排列中，交换任意两个数，会导致整个排列的逆序对个数奇偶性改变

证明：交换任意两个数 L 和 R ($L < R$, 当 L 比 R 大时证明思路一致), L 左边的和 R 右边的数对 $[L, R]$ 内的数的贡献、 (L, R) 内的数相互之间的贡献不发生改变，再看 (L, R) 内，对于其中任意一个数，如果这个数在 L 和 R 中间，交换之前的该数与 L 和 R 的贡献值之和为 0, 那么 L 和 R 交换后，其贡献值还是 2，若比 L 和 R 都小或者都大，交换前后的贡献值都是 1，故不管三个数之间的关系是什么，改变都是偶数，都不改变总体奇偶性，而 L 和 R 的交换本身会带来绝对值为 1 的改变，故总体奇偶性最后改变

一个排列中，对任意区间进行循环单向移动时，当移动区间的长度为奇数时，每次移动的改变都是偶数，不改变奇偶，长度偶数时，每次移动的改变都是奇数，故每次移动都会改变奇偶，

证明：对于任何一个数，在长度为 d 的区间中，有 a 个数比他大，有 b 个数比他小， $a + b = d - 1$ ，两个数的和为偶数，那么两个数的差也为偶数，而将一侧的数循环移动到另一侧时，在循环左移时，其贡献就是 $b - a$ ，因此，其奇偶性确定

组合数求和

$$\sum_{i=1}^n \frac{i*(i+1)}{2}$$
 的值为 $(n * (n + 1) * (n + 2)) / 6$.

数位和之和

数位和的意思是对于 $f(100)$, 其值为每一位上的数之和, 值为 $1+0+0=1$, 而对于 $f(123)$, 其值同理为 $1+2+3=6$.

若要求 zh , 方法如下, 将视角转移至数位上, 把所有数的数位以补零的方式与 n 的数位对齐, 然后分三部分进行值的计算:

计算之前要指出, 对于 10^l 次方以前的数, 每一个 10^l 都对应着 $l * 10^{l-1}$ 个45, 可自行枚举再稍加思考发现规律

1. 当前数字以前的所有数字所对应的45的组数
2. 当前位的当前数字以前的值之和
3. 当前位的当前数字的和

```
1 // https://codeforces.com/contest/2132/problem/D
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define int long long
5 #define fr first
6 #define sc second
7 signed main() {
8     cin.tie(nullptr)->iostream::sync_with_stdio(false);
9     int t = 1;
10    cin >> t; // 输入测试用例数量
11    while(t --) {
12        int n;
13        cin >> n; // 输入 k (保留的总位数)
14        // get(x): 计算从 1 拼到 x 一共用了多少位数字
15        auto get = [&](int x) -> int {
16            int len1 = 0; // 记录总长度
17            int p = 1; // 当前区间的起始数: 1, 10, 100, ...
18            int d = 1; // 当前区间的位数 (1 位数, 2 位数, ...)
19            // 先处理完整区间 (例如 1~9, 10~99, ...)
20            while(p * 10 <= x) {
21                len1 += 9 * p * d; // 该区间一共的位数 = 数字个数 * 位数
22                d++;
23                p *= 10;
24            }
25            // 最后剩余区间 (p..x)
26            len1 += (x - p + 1) * d;
27            return len1;
28        };
29        // 二分查找: 找到最小的 ans 使得写到 ans 的位数 ≥ n
30        int l = 1, r = 1e15, mid, ans;
31        while(l <= r) {
32            mid = (l + r) >> 1;
33            if(get(mid) >= n) ans = mid, r = mid - 1;
34            else l = mid + 1;
35        }
36        int val = 0; // 结果: 数位和
37        // 如果 get(ans) > n, 说明 ans 的最后一个数可能多取了几位, 需要回退
38        if(get(ans) > n) {
39            int x = ans;
```

```

40     int len2 = get(ans --); // 先退到 ans-1
41     // 继续退掉多余的位
42     while(len2 > n) len2 --, x /= 10;
43     // 把保留下来的 x 的每一位加到答案
44     while(x) val += x % 10, x /= 10;
45 }
46 // 处理完整的部分 (1 到 ans)
47 if(ans) {
48     string s = to_string(ans); // 把 ans 转成字符串
49     int l = 1; // 当前处理的位数
50     int base = 1; // 位权
51     int sum = 0; // 前缀和 (用来算贡献)
52     // 从低位到高位枚举 ans 的每一位
53     for(auto i = s.rbegin(); i < s.rend(); i++, base *= 10, l++) {
54         int x = *i - '0'; // 当前位的数值
55         sum += x * base;
56         // 复杂的公式: 计算该位对数位和的贡献
57         val += x * (x - 1) / 2 * base // 当前位取 0..(x-1)
      的贡献
58             + 45 * base / 10 * (l - 1) * x // 低位的循环贡献
59             + x * (sum - x * base + 1); // 高位固定时的贡献
60     }
61 }
62 cout << val << "\n"; // 输出结果
63 }
64 return 0;
65 }

```

分层补齐高度问题

要把一个区间通过子区间加一的操作将区间统一高度，可以通过以下方法

- 若i为第一个数，其代价为 $\max - \text{val}[i]$
- 否则，当i要被填补的高度大于*i - 1*时，才进行高度差距的补充，类似与只有上楼梯才有消耗，下楼梯没有消耗，代价为 $(\max - \text{val}[i]) - (\max - \text{val}[i - 1])$

而区间可容纳两种最终高度时，可dp每个点变为最终两个值中的某一个的代价，假设最终高度为x,y,可参考以下代码

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 #define fr first
5 #define sc second
6 constexpr int N = 400;
7 // dp[i][0/1]: 处理到下标 i (0-based)，并且把 v1[i] 提升到“高位 x(=0)”或“低位 y(=1)”时,
8 //           使得前缀 0..i 达到目标高度序列所需的最小“操作代价”
9 vector<int> dp(N + 1, vector<int>(2));
10 signed main() {
11     cin.tie(nullptr)->iostream::sync_with_stdio(false);
12     int t = 1;
13     cin >> t;

```

```

14     while(t --) {
15         int n, m;
16         cin >> n >> m;
17         vector<int> v1(n);
18         int max1 = 0;
19         for(auto & x : v1) cin >> x, max1 = max(max1, x);
20         int ans = 0;
21         if(m == 1) {
22             // 只允许选择 1 个“目标高度”时，最优即取 H = max(v1)
23             // 下面这段等价于：把所有位置都“补到” H 的最少“前缀加一”操作次数。
24             // 令 delta[i] = H - v1[i]，则
25             // 代价 = delta[0] + Σ_{i=1..n-1} max(0, delta[i] - delta[i-1])
26             for(int i = 0; i < n; i++) {
27                 if(i == 0) ans += max1 - v1[i];
28                 else ans += max(0ll, (max1 - v1[i]) - (max1 - v1[i - 1]));
29                 //           ↑↑↑           即 delta[i] - delta[i-1]
30             }
31         } else {
32             // 允许选择 2 个“目标高度”：高位 x 与低位 y (并约束 y <= x)
33             // 我们要在每个位置自由选择 x 或 y，使得 h[i] >= v1[i] 且总代价最小。
34             auto get = [&](int x, int y) -> int {
35                 // 初始化 DP 为 +∞
36                 fill(dp.begin(), dp.begin() + 1 + n, vector<int>(2, 1e18));
37                 // i = 0 的转移：若 v1[0] 不超过目标位，则代价为补到该位的“新开层数”
38                 if(v1[0] <= x) dp[0][0] = x - v1[0]; // delta[0] = x - v1[0]
39                 if(v1[0] <= y) dp[0][1] = y - v1[0]; // delta[0] = y - v1[0]
40                 for(int i = 1; i < n; i++) {
41                     // 若当前位置选择“高位 x”
42                     if(v1[i] <= x) {
43                         // 从上一位也是 x 转来：
44                         // 需要额外付出 max(0, (x - v[i]) - (x - v[i-1])) =
max(0, delta[i]-delta[i-1])
45                         dp[i][0] = min({
46                             dp[i][0],
47                             dp[i - 1][0] + max(0ll, (x - v1[i]) - (x - v1[i - 1])),
48                         });
49                         // 从上一位是 y 转来：
50                         // 比较 delta[i]=x-v[i] 与 前一位的 delta[i-1]=y-v[i-1]
51                         dp[i - 1][1] + max(0ll, (x - v1[i]) - (y - v1[i - 1]))
52                     );
53                     // 若当前位置选择“低位 y”
54                     if(v1[i] <= y) {
55                         dp[i][1] = min({
56                             dp[i][1],
57                             dp[i - 1][0] + max(0ll, (y - v1[i]) - (x - v1[i - 1])),
58                         });
59                     );
60                 }
61             }
62             return min(dp[n - 1][0], dp[n - 1][1]);
63         };

```

```

64 // 穷举所有 (x, y) (0..200 且 y<=x) 取最小值
65 // 注: 范围 200 来自题面 (或数据范围) 假设
66 ans = 1e18;
67 for(int i = 0; i <= 200; i++)
68     for(int j = 0; j <= i; j++)
69         ans = min(ans, get(i, j));
70 }
71 cout << ans << "\n";
72 }
73 return 0;
74 }
```

行列取模构造

对于给定的行n和列m,如果需要给定n个值a和m个值b, 那么所构成的n*m矩阵M中,
 $M_{ij} = a_i b_j \pmod{nm}$, $1 \leq i \leq n$, $1 \leq j \leq m$, 每个数都不一样, 且所有的a和b都处于 $[0, n * m - 1]$ 中, 应该如何给定每一个a和b的值。

答案为: 构造

$$a_i = (i * m + 1) \pmod{n * m}, (1 \leq i \leq n) \text{ 和 } b_j = (j * m + 1) \pmod{n * m}, (1 \leq j \leq m).$$

证明: 对于 $i = 1$ 到 n , $a_i = (i * m + 1) \pmod{n * m}$; 对于 $j = 1$ 到 m ,
 $b_j = (j * m + 1) \pmod{n * m}$ 。这样产生的 $n \times m$ 个值 $c_{i,j} = a_i * b_j \pmod{(n * m)}$ 。为简便起见, 令 $N = n * m$ 。

题解中的证明显示, 如果 $a_i b_j \equiv a_{i'} b_{j'} \pmod{N}$, 则 $m(i - i') \equiv n(j' - j) \pmod{N}$ 。重新整理后, 可得 $m(i - i' - kn) = n(j' - j)$ 对于某个整数 k 。由于 $\gcd(m, n) = 1$, 由此可知 m 整除 $(j' - j)$ 。但 $|j' - j| < m$, 因此唯一可能的是 $j' = j$, 进而 $i' = i$ 。从而, 所有 $c_{i,j}$ 在模 N 下两两不同。

共有恰好 N 个这样的 $c_{i,j}$ 值 (对应 $n \times m = N$ 个 (i, j) 对), 且它们均为 $\{0, 1, \dots, N - 1\}$ 中的不同整数。因此, 集合 $\{c_{i,j}\}$ 必须恰好是 $\{0, 1, \dots, N - 1\}$, 因为这是一个从大小为 N 的集合到大小为 N 的集合的单射映射 (模 N 下)。

参考资料

[字符串 CSDN](#)

[左程云](#)

[三角形面积](#)

[lower_bound & upper_bound](#)

[C++ 参考手册](#)

[ASCII WIKI](#)

[判断点与直线的位置关系](#)

[母函数](#)

[傅里叶变换](#)

[AVL树](#)

[Morris遍历文案](#)