1. 博弈论
2. Wythoff(黄金分割，两堆石头，可以一起拿一样的)

|  |
| --- |
| /\* |
| 有2堆石子。A B两个人轮流拿，A先拿。每次可以从一堆中取任意个或从 |
| 2堆中取相同数量的石子，但不可不取。拿到最后1颗石子的人获胜。假设 |
| A B都非常聪明，拿石子的过程中不会出现失误。给出2堆石子的数量，问 |
| 最后谁能赢得比赛。 |
| \*/ |
| int main() |
| { |
| int t, a, b, m, k; |
| scanf("%d", &t); |
| while (t--) |
| { |
| scanf("%d%d", &a, &b); |
| if (a > b) |
| { |
| a ^= b; |
| b ^= a; |
| a ^= b; |
| } |
| m = b - a; |
| k = (int)(m \* (1 + sqrt(5)) / 2.0); |
| //m = ? \* a |
| //k = m / ? |
| //?:黄金分割数 |
| //如果a == k，则为后手赢，否则先手赢（奇异局） |
| printf("%s\n", a == k ? "B" : "A"); |
| } |
| return 0; |
| } |

1. Bash(石头谁先拿到最后一个)

|  |
| --- |
| /\* |
| 有一堆石子共有N个。A B两个人轮流拿，A先拿。每次最少拿1颗， |
| 最多拿K颗，拿到最后1颗石子的人获胜。假设A B都非常聪明， |
| 拿石子的过程中不会出现失误。给出N和K，问最后谁能赢得比赛。 |
| \*/ |
| /\* bashgame \*/ |
| int bash(int N, int K) |
| { |
| if (N % (K + 1) == 0) return 2; |
| return 1; |
| } |

1. nim游戏，N堆石子，每次可全拿

|  |
| --- |
| /\*有N堆石子。A B两个人轮流拿，A先拿。每次只能从一堆中取若干个， |
| 可将一堆全取走，但不可不取，拿到最后1颗石子的人获胜。假设A |
| B都非常聪明，拿石子的过程中不会出现失误。给出N及每堆石子的数量， |
| 问最后谁能赢得比赛。\*/ |
| /\*Nim 游戏\*/ |
| int main(int argc, const char \* argv[]) |
| { |
| int N, stone, tag = 0; |
| scanf("%d", &N); |
| while (N--){ |
| scanf("%d", &stone); |
| tag ^= stone; |
| } |
| //tag为0则为后手赢，否则为先手赢 |
| printf("%c\n", tag == 0 ? 'B' : 'A'); |
| return 0; |
| } |

1. sg函数

|  |
| --- |
| const int MAX\_DIG = 64; |
| //f[]：可以取走的石子个数 |
| //sg[]:0~n的SG函数值 |
| //hash[]:mex{} |
| int f[MAX\_DIG]={1,3},sg[MAX\_DIG],hash[MAX\_DIG]; |
| int k;//k是f[]的有效长度 |
| void getSG(int n) |
| { |
| memset(sg,0,sizeof(sg)); |
| for(int i=1; i<=n; i++) { |
| memset(hash,0,sizeof(hash)); |
| for(int j=0; f[j]<=i && j < k; j++) //k是f[]的有效长度 |
| hash[sg[i-f[j]]]=1; |
| for(int j=0; ; j++) { //求mes{}中未出现的最小的非负整数 |
| if(hash[j]==0) { |
| sg[i]=j; |
| break;} |
| } |
| } |
| } |
| int main(){ |
| int n; |
| read(n); |
| k=2; |
| getSG(n); |
| for(int i=0;i<=n;i++){ |
| cout<<sg[i]<<” ”; |
| } |
| return 0; |
| } |

1. Fefjasl
2. 树
   * 1. 二叉树
3. 中序遍历+后序遍历建二叉树

|  |
| --- |
| // UVa548 Tree |
| // Rujia Liu |
| // 题意：给一棵点带权（权各不相同，都是正整数）二叉树的中序和后序遍历，找一个叶子使得它到根的路径上的权和最小。如果有多解，该叶子本身的权应尽量小 |
| // 算法：递归建树，然后DFS。注意，直接递归求结果也可以，但是先建树的方法不仅直观，而且更好调试 |
| // 因为各个结点的权值各不相同且都是正整数，直接用权值作为结点编号 |
| const int maxv = 10000 + 10; |
| int in\_order[maxv], post\_order[maxv], lch[maxv], rch[maxv]; |
| int n; |
| bool read\_list(int\* a) { |
| string line; |
| if(!getline(cin, line)) return false; |
| stringstream ss(line); |
| n = 0; |
| int x; |
| while(ss >> x) a[n++] = x; |
| return n > 0; |
| } |
| // 把in\_order[L1..R1]和post\_order[L2..R2]建成一棵二叉树，返回树根 |
| int build(int L1, int R1, int L2, int R2) { |
| if(L1 > R1) return 0; // 空树 |
| int root = post\_order[R2]; |
| int p = L1; |
| while(in\_order[p] != root) p++; |
| int cnt = p-L1; // 左子树的结点个数 |
| lch[root] = build(L1, p-1, L2, L2+cnt-1); |
| rch[root] = build(p+1, R1, L2+cnt, R2-1); |
| return root;} |
| int best, best\_sum; // 目前为止的最优解和对应的权和 |
| void dfs(int u, int sum) { |
| sum += u; |
| if(!lch[u] && !rch[u]) // 叶子 |
| if(sum < best\_sum || (sum == best\_sum && u < best)) { best = u; best\_sum = sum; } |
| if(lch[u]) dfs(lch[u], sum); |
| if(rch[u]) dfs(rch[u], sum); |
| } |
| int main() { |
| while(read\_list(in\_order)) { |
| read\_list(post\_order); |
| build(0, n-1, 0, n-1); |
| best\_sum = 1000000000; |
| dfs(post\_order[n-1], 0); |
| cout << best << "\n"; |
| } |
| return 0; |
| } |

1. 二叉树建树

|  |
| --- |
| // UVa122 Trees on the level |
| const int maxn = 256 + 10; |
| struct Node{ |
| bool have\_value; |
| int v; |
| Node\* left, \*right; |
| Node():have\_value(false),left(NULL),right(NULL){} |
| }; |
| Node\* root; |
| Node\* newnode() { return new Node(); } |
| bool failed; |
| void addnode(int v, char\* s) { |
| int n = strlen(s); |
| Node\* u = root; |
| for(int i = 0; i < n; i++) |
| if(s[i] == 'L') { |
| if(u->left == NULL) u->left = newnode(); |
| u = u->left; |
| } else if(s[i] == 'R') { |
| if(u->right == NULL) u->right = newnode(); |
| u = u->right; |
| } |
| if(u->have\_value) failed = true; |
| u->v = v; |
| u->have\_value = true; |
| } |
| void remove\_tree(Node\* u) { |
| if(u == NULL) return; |
| remove\_tree(u->left); |
| remove\_tree(u->right); |
| delete u; } |
| char s[maxn]; |
| bool read\_input() { |
| failed = false; |
| remove\_tree(root); |
| root = newnode(); |
| for(;;) { |
| if(scanf("%s", s) != 1) return false; |
| if(!strcmp(s, "()")) break; |
| int v; |
| sscanf(&s[1], "%d", &v); |
| addnode(v, strchr(s, ',')+1); |
| } |
| return true; } |
| bool bfs(vector<int>& ans) { |
| queue<Node\*> q; |
| ans.clear(); |
| q.push(root); |
| while(!q.empty()) { |
| Node\* u = q.front(); q.pop(); |
| if(!u->have\_value) return false; |
| ans.push\_back(u->v); |
| if(u->left != NULL) q.push(u->left); |
| if(u->right != NULL) q.push(u->right); |
| } |
| return true; } |
| int main() { |
| vector<int> ans; |
| while(read\_input()) { |
| if(!bfs(ans)) failed = 1; |
| if(failed) printf("not complete\n"); |
| else { |
| for(int i = 0; i < ans.size(); i++) { |
| if(i != 0) printf(" "); |
| printf("%d", ans[i]); |
| } |
| printf("\n"); |
| } } return 0; } |

* + 1. saff

1. 其他
   1. C++大数

|  |
| --- |
| struct Bign |
| { |
| int len,s[MAXN]; |
| Bign(){ |
| memset(s,0,sizeof(s)); |
| len=1; |
| } |
| Bign(int num){\*this=num;} |
| Bign(const char \*num){\*this=num;} |
| void clean(){while(len>1&&!s[len-1])len--;} |
| Bign operator = (const int num){ |
| char s[MAXN]; |
| sprintf(s,"%d",num); |
| \*this=s; |
| return \*this; |
| } |
| Bign operator = (const char \*num) { |
| len=strlen(num); |
| for(int i=0;i<len;i++)s[i]=num[len-i-1]-'0'; |
| return \*this; |
| } |
| Bign operator + (const Bign& b) { |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;g||i<Max(len,b.len);i++){ |
| int x=g; |
| if(i<b.len)x+=b.s[i]; |
| if(i<len)x+=s[i]; |
| c.s[c.len++]=x%10; |
| g=x/10; |
| } |
| return c; |
| } |
| Bign operator - (const Bign& b){ |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;i<len;i++){ |
| int x=s[i]-g; |
| if(i<b.len)x-=b.s[i]; |
| if(x>=0)g=0; |
| else{g=1;x+=10;} |
| c.s[c.len++]=x; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator \* (const Bign& b){ |
| Bign c; |
| c.len=len+b.len; |
| for(int i=0;i<len;i++){ |
| for(int j=0;j<b.len;j++) c.s[i+j]+=s[i]\*b.s[j]; |
| } |
| for(int i=0;i<c.len;i++) { |
| c.s[i+1]+=c.s[i]/10; |
| c.s[i]%=10; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator \* (const int& b){ |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;g||i<len;i++){ |
| int x; |
| if(i<len)x=s[i]\*b+g; |
| else x=g; |
| c.s[c.len++]=x%10; |
| g=x/10; |
| } |
| return c; |
| } |
| Bign operator / (const Bign& b){ |
| Bign c,f=0; |
| for(int i=len-1;i>=0;i--){ |
| f=f\*10; |
| f.s[0]=s[i]; |
| while(f>=b){ |
| f=f-b; |
| c.s[i]++; |
| } |
| } |
| c.len=len; |
| c.clean(); |
| return c; |
| } |
| Bign operator / (const int& b){ |
| Bign c,d=\*this; |
| c.len=len; |
| for(int i=len-1,g=0;i>=0;i--){ |
| d.s[i]+=g\*10; |
| c.s[i]=d.s[i]/b; |
| g=d.s[i]%b; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator % (const Bign& b){ |
| Bign c=\*this/b; |
| c=\*this-c\*b; |
| return c; |
| } |
| Bign operator += (const Bign& b) |
| {\*this=\*this+b;return \*this;} |
| Bign operator -= (const Bign& b) |
| {\*this=\*this-b;return \*this;} |
| Bign operator \*= (const Bign& b) |
| {\*this=\*this\*b;return \*this;} |
| Bign operator /= (const Bign& b) |
| {\*this=\*this/b;return \*this;} |
| Bign operator \*= (const int& b) |
| {\*this=\*this\*b;return \*this;} |
| Bign operator /= (const int& b) |
| {\*this=\*this/b;return \*this;} |
| Bign operator %= (const Bign& b) |
| {\*this=\*this%b;return \*this;} |
| bool operator < (const Bign& b){ |
| if(b.len!=len)return len<b.len; |
| for(int i=len-1;i>=0;i--) if(s[i]!=b.s[i])return s[i]<b.s[i]; |
| return 0; |
| } |
| bool operator > (const Bign& b){ |
| if(b.len!=len)return len>b.len; |
| for(int i=len-1;i>=0;i--) if(s[i]!=b.s[i])return s[i]>b.s[i]; |
| return 0; |
| } |
| bool operator == (const Bign& b) |
| {return !(\*this>b)&&!(\*this<b);} |
| bool operator <= (const Bign& b) |
| {return !(\*this>b);} |
| bool operator >= (const Bign& b) |
| {return !(\*this<b);} |
| bool operator != (const Bign& b) |
| {return !(\*this==b);} |
| string str() const { |
| string res; |
| for(int i=0;i<len;i++) |
| res=char(s[i]+'0')+res; |
| return res; |
| } |
| }; |
| //cin 读入 |
| istream& operator >> (istream&in,Bign &x){ |
| string s; |
| in>>s; |
| x=s.c\_str(); |
| return in; |
| } |
| ostream& operator << (ostream&out,Bign x){ |
| out<<x.str(); |
| return out; |
| } |
| int main(){ |
| Bign a,b; |
| cin>>a>>b; |
| cout<<a%b<<endl; |
| return 0; |
| } |

* 1. 基姆拉尔森公式,给年月日,计算星期几

W = (D + 2 \* M + 3 \* (M + 1) \ 5 + Y + Y \ 4 - Y \ 100 + Y \ 400) Mod 7;

* 1. fsfs

1. 图
   1. AStar\_K短路

|  |
| --- |
| const int maxn=100010; |
| int n,m,dis[maxn]; |
| int tot,head1[maxn],head2[maxn]; |
| bool flag[maxn]; |
| struct edge{ |
| int to; |
| int w; |
| int next; |
| }e[maxn\*2],e2[maxn\*2]; |
| struct node{ |
| int f; |
| int g; |
| int from; |
| bool operator < (node a)const{ |
| if(a.f==f) |
| return g>a.g; |
| return f>a.f; |
| } |
| }; |
| void add\_edge(int u,int v,int w){ |
| tot++; |
| e[tot].to=v; |
| e[tot].w=w; |
| e[tot].next=head1[u]; |
| head1[u]=tot; |
| e2[tot].to=u; |
| e2[tot].w=w; |
| e2[tot].next=head2[v]; |
| head2[v]=tot; |
| } |
| void prepare(){ |
| for(int i=1;i<=n;i++) |
| dis[i]=maxn;tot=0; |
| memset(head1,0,sizeof(head1)); |
| memset(head2,0,sizeof(head2)); |
| } |
| void spfa(int t){ |
| for(int i=1;i<=n;i++) |
| dis[i]=maxn; |
| dis[t]=0; |
| queue<int> q; |
| q.push(t); |
| flag[t]=1; |
| while(!q.empty()){ |
| int v=q.front(); |
| q.pop();flag[v]=0; |
| for(int i=head2[v];i;i=e2[i].next) |
| if(dis[e2[i].to]>dis[v]+e2[i].w){ |
| dis[e2[i].to]=dis[v]+e2[i].w; |
| if(!flag[e2[i].to]){ |
| q.push(e2[i].to); |
| flag[e2[i].to]=1; |
| } |
| } |
| } |
| } |
| int a\_star(int s,int t,int k){ |
| if(s==t) k++; |
| if(dis[s]==maxn) return -1; |
| priority\_queue<node> q; |
| int cnt=0; |
| node tmp,to; |
| tmp.from=s; |
| tmp.g=0; |
| tmp.f=tmp.g+dis[tmp.from]; |
| q.push(tmp); |
| while(!q.empty()){ |
| tmp=q.top(); |
| q.pop(); |
| if(tmp.from==t) cnt++; |
| if(cnt==k) return tmp.g; |
| for(int i=head1[tmp.from];i;i=e[i].next){ |
| to.from=e[i].to; |
| to.g=tmp.g+e[i].w; |
| to.f=to.g+dis[to.from]; |
| q.push(to); |
| } |
| } |
| return -1; |
| } |
| int main(){ // 该模板能处理带环图 |
| int x,y,z,s,t,k; |
| while(cin>>n>>m) {// 输入n个点 m条边 |
| prepare(); |
| cin>>s>>t>>k; // 输入起点 终点 第k短路 |
| for(int i=1;i<=m;i++) {// 输入边 |
| cin>>x>>y>>z; |
| add\_edge(x,y,z); |
| } |
| spfa(t); |
| int ans=a\_star(s,t,k); // ans 为第k短路的长度 |
| } |
| return 0; |
| } |

* 1. DAG深度优先队列标记

|  |
| --- |
| /\*DAG(有向无环图)的深度优先搜索标记 |
| \* INIT:edge[][]邻接矩阵；pre[], post[], tag全置0 |
| CALL:dfsTag(i, n); pre/post:开始/结束时间\*/ |
| const int V = 1010; |
| int edge[V][V]; |
| int pre[V]; |
| int post[V]; |
| int tag; |
| void dfsTag(int cur, int n){ |
| //vertex:0 ~ n - 1 |
| pre[cur] = ++tag; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (0 == pre[i]){ |
| std::cout << "Three Edge!" << '\n'; |
| dfsTag(i, n); |
| } |
| else{ |
| if (0 == post[i]) std::cout << "Back Edge!" << '\n'; |
| else if (pre[i] > pre[cur]) std::cout << "Down Edge!" << '\n'; |
| else std::cout << "Cross Edge!" << '\n'; |
| }} |
| } |
| post[cur] = ++tag; |
| return ; |
| } |

* 1. 无向图找桥

|  |
| --- |
| /\*无向图找桥 |
| \* INIT: edge[][]邻接矩阵；vis[],pre[],ans[],bridge置0； |
| CALL: dfs(0, -1, 1, n);\*/ |
| const int V = 1010; |
| int bridge; //桥 |
| int edge[V][V]; |
| int ans[V]; |
| int pre[V]; |
| int vis[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex: 0 ~ n - 1 |
| if (bridge) return ; |
| vis[cur] = 1; |
| pre[cur] = ans[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]){ |
| if (pre[i] < ans[cur]) ans[cur] = pre[i]; //back edge |
| } |
| if (0 == vis[i]) { //tree edge |
| dfs(i, cur, dep + 1, n); |
| if (bridge) return ; |
| if (ans[i] < ans[cur]) ans[cur] = ans[i]; |
| if (ans[i] > pre[cur]){bridge = 1; return ;} |
| } |
| } |
| } |
| vis[cur] = 2; |
| } |
| int main(){ |
| // 在这里输入n |
| /\* |
| \* 在这里输入图 |
| \*/ |
| // dfs(0,-1,1,n); 调用函数 |
| } |

* 1. 无向图连通度(割点)

|  |
| --- |
| const int V = 1010; |
| int edge[V][V]; |
| int anc[V]; |
| int pre[V]; |
| int vis[V]; |
| int deg[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex:0 ~ n - 1 |
| int cnt = 0; |
| vis[cur] = 1; |
| pre[cur] = anc[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]) if (pre[i] < anc[cur]) anc[cur] = pre[i]; //back edge |
| if (0 == vis[i]){ //tree edge |
| dfs(i, cur, dep + 1, n); |
| cnt++; //分支个数 |
| if (anc[i] < anc[cur]) anc[cur] = anc[i]; |
| if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur])) deg[cur]++; //link degree of a vertex |
| } |
| } |
| } |
| vis[cur] = 2; |
| } |
| int main(){ |
| /\* INIT: edge[][]邻接矩阵；vis[],pre[],anc[],deg[]置为0； |
| \* CALL: dfs(0, -1, 1, n); |
| \* k = deg[0], deg[i] + 1(i = 1...n - 1)为删除该节点后得到的连通图个数 |
| 注意: 0作为根比较特殊\*/ |
| } |
| const int V = 1010; |
| int edge[V][V]; |
| int anc[V]; |
| int pre[V]; |
| int vis[V]; |
| int deg[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex:0 ~ n - 1 |
| int cnt = 0; |
| vis[cur] = 1; |
| pre[cur] = anc[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]) if (pre[i] < anc[cur]) anc[cur] = pre[i]; //back edge |
| if (0 == vis[i]){ //tree edge |
| dfs(i, cur, dep + 1, n); |
| cnt++; //分支个数 |
| if (anc[i] < anc[cur]) anc[cur] = anc[i]; |
| if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur])) deg[cur]++; //link degree of a vertex |
| } |
| } |

* 1. 曼哈顿最小生成树

|  |
| --- |
| const int MAXN = 100010; |
| const int INF = 0x3f3f3f3f; |
| struct Point{ |
| int x; |
| int y; |
| int id; |
| }poi[MAXN]; |
| bool cmp(Point a, Point b){ |
| if (a.x != b.x) return a.x < b.x; |
| else return a.y < b.y; |
| } |
| //树状数组，找y - x大于当前的，但是y + x最小的 |
| struct BIT{ |
| int minVal; |
| int pos; |
| void init(){ |
| minVal = INF; |
| pos = -1; |
| } |
| }bit[MAXN]; |
| //所有有效边 |
| struct Edge{ |
| int u; |
| int v; |
| int d; |
| }edge[MAXN << 2]; |
| bool cmpEdge(Edge a, Edge b){ return a.d < b.d;} |
| int tot; |
| int n; |
| int F[MAXN]; |
| int find(int x){ |
| if (F[x] == -1) return x; |
| else return F[x] = find(F[x]); |
| } |
| void addEdge(int u, int v, int d){ |
| edge[tot].u = u; |
| edge[tot].v = v; |
| edge[tot++].d = d; |
| return ; |
| } |
| int lowbit(int x){ return x & (-x);} |
| //更新bit |
| void update(int i, int val, int pos){ |
| while (i > 0){ |
| if (val < bit[i].minVal){ |
| bit[i].minVal = val; |
| bit[i].pos = pos; |
| } |
| i -= lowbit(i); |
| } |
| return ; |
| } |
| //查询[i, m]的最小值位置 |
| int ask(int i, int m){ |
| int minVal = INF; |
| int pos = -1; |
| while (i <= m){ |
| if (bit[i].minVal < minVal){ |
| minVal = bit[i].minVal; |
| pos = bit[i].pos; |
| } |
| i += lowbit(i); |
| } |
| return pos; |
| } |
| int dist(Point a, Point b){ return abs(a.x - b.x) + abs(a.y - b.y);} |
| void ManhattanMinimumSpanningTree(int n, Point p[]){ |
| int a[MAXN], b[MAXN]; |
| tot = 0; |
| for (int dir = 0; dir < 4; dir++){ |
| //变换4种坐标 |
| if (dir == 1 || dir == 3){ |
| for (int i = 0; i < n; i++) std::swap(p[i].x, p[i].y); |
| } |
| else if (dir == 2){ |
| for (int i = 0; i < n; i++) p[i].x = -p[i].x; |
| } |
| std::sort(p, p + n, cmp); |
| for (int i = 0; i < n; i++) a[i] = b[i] = p[i].y - p[i].x; |
| std::sort(b, b + n); |
| int m = (int)(std::unique(b, b + n) - b); |
| for (int i = 1; i <= m; i++) bit[i].init(); |
| for (int i = n - 1; i >= 0; i--){ |
| int pos = (int)(std::lower\_bound(b, b + m, a[i]) - b + 1); |
| int ans = ask(pos, m); |
| if (ans != -1) addEdge(p[i].id, p[ans].id, dist(p[i], p[ans])); |
| update(pos, p[i].x + p[i].y, i); |
| } |
| } |
| return ; |
| } |
| int solve(int k){ |
| ManhattanMinimumSpanningTree(n, poi); |
| memset(F, -1, sizeof(F)); |
| std::sort(edge, edge + tot, cmpEdge); |
| for (int i = 0; i < tot; i++){ |
| int u = edge[i].u; |
| int v = edge[i].v; |
| int tOne = find(u); |
| int tTwo = find(v); |
| if (tOne != tTwo){ |
| F[tOne] = tTwo; |
| k--; |
| if (k == 0) return edge[i].d; |
| } |
| } |
| return -1; |
| } |
| int main(int argc, const char \* argv[]){ |
| //freopen("in.txt", "r", stdin); |
| //freopen("out.txt", "w", stdout); |
| int k; |
| while ((std::cin >> n >> k) && n){ |
| for (int i = 0; i < n; i++){ |
| std::cin >> poi[i].x >> poi[i].y; |
| poi[i].id = i; |
| } |
| std::cout << solve(n - k) << std::endl; |
| } |
| return 0; |
| } |

* 1. 最小生成树(prim)

|  |
| --- |
| #define inf 0x3f3f3f3f |
| typedef struct { |
| int point; |
| int value; |
| }node; |
| int N; |
| int sum; |
| vector<node>point[1000]; |
| int hashtable[1000] = {0}; |
| int num[1000]; |
| void init() { |
| int i; |
| for (i = 0; i < N; i++) num[i] = inf; |
| num[0] = 0; |
| sum = 0; |
| } |
| void prim() { |
| int n = N,i,k,min,min\_num; |
| while (n--) { |
| min=-1; |
| while (n--) { |
| min\_num = inf; |
| for (i = 0; i < N; i++) { |
| if (hashtable[i] == 0 && num[i] != inf) { |
| if (num[i] < min\_num) { |
| Min=i; |
| min\_num = num[i]; |
| } |
| } |
| } |
| sum += num[min]; |
| hashtable[min] = 1; |
| if (min == -1) return; |
| for (k = 0; k < point[min].size(); k++) { |
| int v = point[min][k].point; |
| int value = point[min][k].value; |
| if (num[v] > value && hashtable[v] == 0) num[v] = value; |
| } |
| } |
| } |
| int main() { |
| int M; |
| scanf("%d", &N); |
| scanf("%d", &M); |
| int x, y, i, check, value; |
| for (i = 0; i < M; i++) { |
| scanf("%d%d%d", &x, &y, &value); |
| node new\_node = { y,value }; |
| point[x].push\_back(new\_node); |
| new\_node.point = x; |
| point[y].push\_back(new\_node); |
| } |
| init(); |
| prim(); |
| printf("%d",sum); |
| scanf("%d", &check); |
| } |

* 1. 次小生成树

|  |
| --- |
| int g[M][M],path[M][M];//path求的是i到j最大的边权 |
| int dist[M],pre[M],vis[M]; |
| bool used[M][M];//是否在最小生成树中 |
| int n,m,mst; |
| void init(){ |
| for(int i=0;i<=n;i++) |
| for(int j=i+1;j<=n;j++) g[i][j]=g[j][i]=inf; |
| } |
| int prime() { |
| int mst=0; |
| memset(path,0,sizeof(path)); |
| memset(vis,0,sizeof(vis)); |
| memset(used,0,sizeof(used)); |
| vis[1]=1; |
| for(int i=1;i<=n;i++){ |
| dist[i]=g[1][i]; |
| pre[i]=1; |
| } |
| for(int i=1;i<n;i++) { |
| int u=-1; |
| for(int j=1;j<=n;j++){ |
| if(!vis[j]) if(u==-1||dist[j]<dist[u]) u=j; |
| } |
| used[u][pre[u]]=used[pre[u]][u]=true;//加入mst |
| mst+=g[pre[u]][u]; |
| vis[u]=1; |
| for(int j=1;j<=n;j++) |
| { |
| if(vis[j]&&j!=u)//从u到j这条路径上最大边的权值 |
| path[j][u]=path[u][j]=max(path[j][pre[u]],dist[u]); |
| if(!vis[j]) |
| if(dist[j]>g[u][j]){//更新相邻节点的距离 |
| dist[j]=g[u][j]; |
| pre[j]=u;//记录他的前驱 |
| } |
| } |
| } |
| return mst; |
| } |
| int second\_tree(){//求次小生成树 |
| int res=inf; |
| for(int i=1;i<=n;i++) |
| for(int j=1;j<=n;j++) |
| if(i!=j&&!used[i][j]) |
| res=min(res,mst-path[i][j]+g[i][j]);//删除树上权值最大的路径并且加上这条路径其它边 |
| return res; |
| } |
| int main() { |
| int t; |
| scanf("%d",&t); |
| while(t--) { |
| scanf("%d%d",&n,&m); |
| init(); |
| mst=prime();//最小生成树 |
| int second\_mst=second\_tree();//次小生成树 |
| } |
| } |

* 1. 欧拉路径

|  |
| --- |
| /\*SGU 101 \*/ |
| struct Edge{ |
| int to; |
| int next; |
| int index; |
| int dir; |
| bool flag; |
| } edge[220]; |
| int head[10]; //前驱 |
| int tot; |
| void init(){ |
| memset(head, -1, sizeof((head))); |
| tot = 0; |
| } |
| void addEdge(int u, int v, int index){ |
| edge[tot].to = v; |
| edge[tot].next = head[u]; |
| edge[tot].index = index; |
| edge[tot].dir = 0; |
| edge[tot].flag = false; |
| head[u] = tot++; |
| edge[tot].to = u; |
| edge[tot].next = head[v]; |
| edge[tot].index = index; |
| edge[tot].dir = 1; |
| edge[tot].flag = false; |
| head[v] = tot++; |
| return ; |
| } |
| int du[10]; |
| std::vector<int>ans; |
| void dfs(int u){ |
| for (int i = head[u]; i != -1; i = edge[i].next){ |
| if (!edge[i].flag){ |
| edge[i].flag = true; |
| edge[i ^ 1].flag = true; |
| dfs(edge[i].to); |
| ans.push\_back(i); //容器尾部插入i |
| } |
| } |
| return ; |
| } |
| int main(){ |
| //freopen("in.txt", "r", stdin); |
| //freopen("out.txt", "w", stdout); |
| int n; |
| while (std::cin >> n) { |
| init(); |
| int u, v; |
| memset(du, 0, sizeof(du)); |
| for (int i = 1; i <= n; i++){ |
| std::cin >> u >> v; |
| addEdge(u, v, i); |
| du[u]++; |
| du[v]++; |
| } |
| int s = -1; |
| int cnt = 0; |
| for (int i = 0; i <= 6; i++){ |
| if (du[i] & 1){ |
| cnt++; |
| s = i; |
| } |
| if (du[i] > 0 && s == -1) s = i; |
| } |
| if (cnt != 0 && cnt != 2){ |
| std::cout << "No solution" << '\n'; |
| continue; |
| } |
| ans.clear(); |
| dfs(s); |
| if (ans.size() != n){ |
| std::cout << "No solution" << '\n'; |
| continue; |
| } |
| for (int i = 0; i < ans.size(); i++){ |
| printf("%d ", edge[ans[i]].index); |
| if (edge[ans[i]].dir == 0) std::cout << "-" << '\n'; |
| else std::cout << "+" << '\n'; |
| } |
| } |
| return 0; |
| } |

* 1. 迪杰斯特拉优化模板

|  |
| --- |
| typedef struct { |
| int point;//能够到达的点 |
| int value;//第一尺度 |
| int cost; //第二尺度 |
| }node; |
| int N; |
| int num[1001];//第一尺度的最小值储存单位 |
| int cost[1001];//第二尺度的最小值储存单位 |
| int hashtable[2000] = { 0 };//哈希表，判断点是否访问过 |
| vector<node>point[2000];//邻接表 |
| void init(int start) {//初始化 |
| int i; |
| for (i = 0; i < N; i++) { |
| num[i] = Max; |
| cost[i] = Max; |
| } |
| num[start] = 0; |
| cost[start] = 0; |
| } |
| void djistra(int start) { |
| init(start); |
| int i; |
| int min = 0; |
| int min\_num; |
| int check; |
| while (1) { |
| min\_num = Max; |
| check = 0; |
| for (i = 0; i < N; i++) {//找出当前离起点最近的且未访问过的节点 |
| if (hashtable[i] == 0 && num[i] != Max) { |
| check = 1; |
| if (num[i] < min\_num) { |
| min = i; |
| min\_num = num[i]; |
| } |
| } |
| } |
| if (check == 0) |
| return;//如果没有就说明优化距离结束 |
| hashtable[min] = 1; |
| for (i = 0; i < point[min].size(); i++) { |
| if (hashtable[point[min][i].point] == 0) { |
| if (num[point[min][i].point] > point[min][i].value + num[min]) {  //以第一尺度为标准，先计算出第一尺度的最小值下的第二尺度的值 |
| num[point[min][i].point] = point[min][i].value + num[min]; |
| cost[point[min][i].point] = point[min][i].cost + cost[min]; |
| } |
| else if (num[point[min][i].point] == point[min][i].value + num[min]) {  //以计算出的第二尺度值为标准，计算出第二尺度的最小值 |
| if (cost[point[min][i].point] > point[min][i].cost + cost[min]) |
| cost[point[min][i].point] = point[min][i].cost + cost[min]; |
| } |
| } |
| } |
| } |
| } |
| int main() { //点标号0开头 |
| int M, start, end; |
| int x, y, value,cost\_value; |
| while (scanf("%d%d", &N, &M) && (N != 0 || M != 0) ) { |
| while (M--) { |
| scanf("%d%d%d%d", &x, &y, &value, &cost\_value); |
| node new\_node = { y,value,cost\_value }; |
| point[x].push\_back(new\_node);//无向边 |
| new\_node.point = x; |
| point[y].push\_back(new\_node); |
| } |
| scanf("%d%d", &start, &end); |
| djistra(start); |
| printf("%d %d\n", num[end], cost[end]); |
| } |
| } |

* 1. 最小树形图

|  |
| --- |
| const int INF = 0x3f3f3f3f; |
| const int MAXN = 1010; |
| const int MAXM = 1000010; |
| struct Edge{ int u, v, cost;}; |
| Edge edge[MAXM]; |
| int pre[MAXN], id[MAXN], visit[MAXN], in[MAXN]; |
| int zhuliu(int root, int n, int m){ |
| int res = 0, v; |
| while (true){ |
| memset(in, 0x3f, sizeof(in)); |
| for (int i = 0; i < m; i++){ |
| if (edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v]){ |
| pre[edge[i].v] = edge[i].u; |
| in[edge[i].v] = edge[i].cost; |
| } |
| } |
| for (int i = 0; i < n; i++){ |
| if (i != root && in[i] == INF) return -1; // 不存在最小树形图 |
| } |
| int tn = 0; |
| memset(id, -1, sizeof(id)); |
| memset(visit, -1, sizeof(visit)); |
| in[root] = 0; |
| for (int i = 0; i < n; i++){ |
| res += in[i]; |
| v = i; |
| while (visit[v] != i && id[v] == -1 && v != root){ |
| visit[v] = i; |
| v = pre[v]; |
| } |
| if (v != root && id[v] == -1){ |
| for (int u = pre[v]; u != v ; u = pre[u]) id[u] = tn; |
| id[v] = tn++; |
| } |
| } |
| if (tn == 0) break; // 没有有向环 |
| for (int i = 0; i < n; i++) if (id[i] == -1) id[i] = tn++; |
| for (int i = 0; i < m; i++){ |
| v = edge[i].v; |
| edge[i].u = id[edge[i].u]; |
| edge[i].v = id[edge[i].v]; |
| if (edge[i].u != edge[i].v) edge[i].cost -= in[v]; |
| } |
| n = tn; |
| root = id[root]; |
| } |
| return res; |
| } |
| int main(){ |
| /\*最小树形图 |
| \* int型 |
| \* 复杂度O(NM) |
| \* 点从0开始\*/} |

* 1. 生成树计数

|  |
| --- |
| /\*取模\*/ |
| // 求生成树计数部分代码,计数对10007取模 |
| const int MOD = 10007; |
| int INV[MOD]; // 逆元打表数组 |
| int g[MAXN][MAXN]; |
| // 求ax = 1(mod m)的x值,就是逆元(0<a<m) |
| long long inv(long long a, long long m){ |
| if (a == 1) return 1; |
| return inv(m % a, m) \* (m - m / a) % m; |
| } |
| struct Matrix{ |
| int mat[330][330]; |
| void init() { memset(mat, 0, sizeof(mat));} |
| int det(int n){ // 求行列式的值模上MOD,需要使用逆元 |
| for (int i = 0; i < n; i++){ |
| for (int j = 0; j < n; j++) mat[i][j] = (mat[i][j] % MOD + MOD) % MOD; |
| } |
| int res = 1; |
| for (int i = 0; i < n; i++){ |
| for (int j = i; j < n; j++){ |
| if (mat[j][i] != 0){ |
| for (int k = i; k < n; k++) swap(mat[i][k], mat[j][k]); |
| if (i != j) res = (-res + MOD) % MOD; |
| break; |
| } |
| } |
| if (mat[i][i] == 0){ |
| res = -1; // 不存在(也就是行列式值为0) |
| break; |
| } |
| for (int j = i + 1; j < n; j++){ |
| int mut = (mat[j][i]\*INV[mat[i][i]])%MOD;//打表逆元 |
| int mut = (mat[j][i] \* inv(mat[i][i], MOD)) % MOD; |
| for (int k = i; k < n; k++){ |
| mat[j][k] = (mat[j][k] - (mat[i][k] \* mut) % MOD + MOD) % MOD; |
| } |
| } |
| res = (res \* mat[i][i]) % MOD; |
| } |
| return res; |
| } |
| }; |
| int main() |
| { |
| Matrix ret; |
| ret.init(); |
| int n; |
| scanf("%d",&n); |
| for(int i = 0;i < n;i++){ |
| int u,v; |
| scanf("%d%d",&u,&v); // 输入数据 |
| u--;v--; |
| g[u][v] = g[v][u] = 1; |
| } |
| for (int i = 0; i < n; i++){ |
| for (int j = 0; j < n; j++){ |
| if (i != j && g[i][j]){ |
| ret.mat[i][j] = -1; |
| ret.mat[i][i]++; |
| } |
| } |
| } |
| printf("%d\n", ret.det(n - 1)); |
| return 0; |
| } |
|  |
| /\*不取模\*/ |
| const double eps = 1e-8; |
| const int MAXN = 110; |
| int sgn(double x){ |
| if (fabs(x) < eps) return 0; |
| if (x < 0) return -1; |
| else return 1; |
| } |
| double b[MAXN][MAXN]; |
| double det(double a[][MAXN], int n){ |
| int i, j, k, sign = 0; |
| double ret = 1; |
| for (i = 0; i < n; i++){ |
| for (j = 0; j < n; j++) b[i][j] = a[i][j]; |
| } |
| for (i = 0; i < n; i++){ |
| if (sgn(b[i][i]) == 0){ |
| for (j = i + 1; j < n; j++){ |
| if (sgn(b[j][i]) != 0) break; |
| } |
| if (j == n) return 0; |
| for (k = i; k < n; k++) swap(b[i][k], b[j][k]); |
| sign++; |
| } |
| ret \*= b[i][i]; |
| for (k = i + 1; k < n; k++) b[i][k] /= b[i][i]; |
| for (j = i+1; j < n; j++){ |
| for (k = i+1; k < n; k++) b[j][k] -= b[j][i] \* b[i][k]; |
| } |
| } |
| if (sign & 1) ret = -ret; |
| return ret; |
| } |
| double a[MAXN][MAXN]; |
| int g[MAXN][MAXN]; |
| int main(){ |
| int T,n,m,u,v; |
| scanf("%d", &T); |
| while (T--){ |
| scanf("%d%d", &n, &m); |
| memset(g, 0, sizeof(g)); |
| while (m--){ |
| scanf("%d%d", &u, &v); // 输入数据 |
| u--; |
| v--; |
| g[u][v] = g[v][u] = 1; |
| } |
| memset(a, 0, sizeof(a)); |
| for (int i = 0; i < n; i++){ |
| for (int j = 0; j < n; j++){ |
| if (i != j && g[i][j]){ |
| a[i][i]++; |
| a[i][j] = -1; |
| } |
| } |
| } |
| double ans = det(a, n - 1); |
| printf("%.0lf\n", ans); |
| } |
| return 0; |
| } |

* 1. 一般图匹配带花树

|  |
| --- |
| const int maxn = 300; |
| int N; |
| bool G[maxn][maxn]; |
| int match[maxn]; |
| bool InQueue[maxn], InPath[maxn], InBlossom[maxn]; |
| int head, tail; |
| int Queue[maxn]; |
| int Start; |
| int finish; |
| int NewBase; |
| int father[maxn], Base[maxn]; |
| int Count; |
| void CreateGraph(){ |
| int u, v; |
| memset(G, 0, sizeof(G)); |
| scanf("%d", &N); |
| while (scanf("%d%d",&u,&v) != EOF) G[u][v] = G[v][u] = true; |
| } |
| void Push(int u){ |
| Queue[tail++] = u; |
| InQueue[u] = true; |
| } |
| int Pop(){ |
| int res = Queue[head++]; |
| return res; |
| } |
| int FindCommonAncestor (int u, int v){ |
| memset(InPath, 0, sizeof(InPath)); |
| while (true){ |
| u = Base[u]; |
| InPath[u] = 1; |
| if (u == Start) break; |
| u = father[match[u]]; |
| } |
| while (true){ |
| v = Base[v]; |
| if (InPath[v]) break; |
| v = father[match[v]]; |
| } |
| return v; |
| } |
| void ResetTrace(int u){ |
| int v; |
| while (Base[u] != NewBase){ |
| v = match[u]; |
| InBlossom[Base[u]] = InBlossom[Base[v]] = 1; |
| u = father[v]; |
| if (Base[u] != NewBase) father[u] = v; |
| } |
| } |
| void BlossomContract(int u, int v){ |
| NewBase = FindCommonAncestor(u, v); |
| memset(InBlossom, 0, sizeof(InBlossom)); |
| ResetTrace(u); |
| ResetTrace(v); |
| if (Base[u] != NewBase) father[u]=v; |
| if (Base[v] != NewBase) father[v]=u; |
| for (int tu=1; tu <= N; tu++){ |
| if (InBlossom[Base[tu]]){ |
| Base[tu] = NewBase; |
| if (!InQueue[tu]) Push(tu); |
| } |
| } |
| } |
| void FindAugmentingPath(){ |
| memset(InQueue, 0, sizeof(InQueue)); |
| memset(father, 0, sizeof(father)); |
| for (int i = 1; i <= N; i++) Base[i] = i; |
| head = tail = 1; |
| Push(Start); |
| finish = 0; |
| while (head < tail){ |
| int u = Pop(); |
| for (int v = 1; v <= N; v++){ |
| if (G[u][v] && (Base[u] != Base[v]) && match[u] != v){ |
| if ((v == Start) || ((match[v] > 0) && father[match[v]] > 0)) BlossomContract(u, v); |
| else if (father[v] == 0){ |
| father[v] = u; |
| if (match[v] > 0) Push(match[v]); |
| else{ |
| finish = v; |
| return ; |
| }}}}} |
| } |
| void AugmentPath(){ |
| int u, v, w; |
| u = finish; |
| while (u > 0){ |
| v = father[u]; |
| w = match[v]; |
| match[v] = u; |
| match[u] = v; |
| u = w; |
| } |
| } |
| void Edmonds(){ |
| memset(match, 0, sizeof(match)); |
| for (int u = 1; u <= N; u++){ |
| if (match[u] == 0){ |
| Start = u; |
| FindAugmentingPath(); |
| if (finish > 0) AugmentPath(); |
| } |
| } |
| } |
| void PrintMatch(){ |
| Count = 0; |
| for (int u = 1; u <= N; u++){ |
| if (match[u] > 0) Count++; |
| } |
| printf("%d\n", Count); |
| for (int u = 1; u <= N; u++){ |
| if (u < match[u]) printf("%d %d\n", u, match[u]); |
| } |
| } |
| int main(){ |
| CreateGraph(); |
| Edmonds(); // 进行匹配 |
| PrintMatch(); // 输出匹配 |
| return 0; |
| } |

* 1. 最大团

|  |
| --- |
| const int V = 10010; |
| int g[V][V]; |
| int dp[V]; |
| int stk[V][V]; |
| int mx; |
| int dfs(int n, int ns, int dep){ |
| if (0 == ns){ |
| if (dep > mx) mx = dep; |
| return 1; |
| } |
| int i, j, k, p, cnt; |
| for (i = 0; i < ns; i++){ |
| k = stk[dep][i]; |
| cnt = 0; |
| if (dep + n - k <= mx) return 0; |
| if (dep + dp[k] <= mx) return 0; |
| for (j = i + 1; j < ns; j++){ |
| p = stk[dep][j]; |
| if (g[k][p]) stk[dep + 1][cnt++] = p; |
| } |
| dfs(n, cnt, dep + 1); |
| } |
| return 1; |
| } |
| int clique(int n){ |
| int i, j, ns; |
| for (mx = 0, i = n - 1; i >= 0; i--){ // vertex: 0 ~ n-1 |
| for (ns = 0, j = i + 1; j < n; j++){ |
| if (g[i][j]) stk[1][ns++] = j; |
| } |
| dfs(n, ns, 1); |
| dp[i] = mx; |
| } |
| return mx; |
| } |
| int main(){ |
| /\*INIT: g[][]邻接矩阵 |
| \* CALL: res = clique(n);\*/ |
| /\*在这里输入n |
| \* 在这里输入邻接矩阵 g[][] \*/ |
| } |

* 1. Sflsj

1. 数据结构
2. 划分树,查询区间第K大

|  |
| --- |
| /\*划分树(查询区间第k大) \*/ |
| const int MAXN = 100010; |
| int tree[20][MAXN]; // 表示每层每个位置的值 |
| int sorted[MAXN]; // 已经排序好的数 |
| int toleft[20][MAXN]; // toleft[p][i]表示第i层从1到i有数分入左边 |
| void build(int l, int r, int dep) |
| { |
| if (l == r) return; |
| int mid = (l + r) >> 1; |
| int same = mid - l + 1; // 表示等于中间值而且被分入左边的个数 |
| for (int i = l; i <= r; i++){ // 注意是l,不是one |
| if (tree[dep][i] < sorted[mid]) same--; |
| } |
| int lpos = l; |
| int rpos = mid + 1; |
| for (int i = l; i <= r; i++){ |
| if (tree[dep][i] < sorted[mid]) tree[dep + 1][lpos++] = tree[dep][i]; |
| else if (tree[dep][i] == sorted[mid] && same > 0){ |
| tree[dep + 1][lpos++] = tree[dep][i]; |
| same--; |
| } |
| else tree[dep + 1][rpos++] = tree[dep][i]; |
| toleft[dep][i] = toleft[dep][l - 1] + lpos - l; |
| } |
| build(l, mid, dep + 1); |
| build(mid + 1, r, dep + 1); |
| return ; |
| } |
| // 查询区间第k大的数,[L,R]是大区间,[l,r]是要查询的小区间 |
| int query(int L, int R, int l, int r, int dep, int k){ |
| if(l == r) return tree[dep][l]; |
| int mid = (L + R) >> 1; |
| int cnt = toleft[dep][r] - toleft[dep][l - 1]; |
| if (cnt >= k){ |
| int newl = L + toleft[dep][l - 1] - toleft[dep][L - 1]; |
| int newr = newl + cnt - 1; |
| return query(L, mid, newl, newr, dep + 1, k); |
| } |
| else{ |
| int newr = r + toleft[dep][R] - toleft[dep][r]; |
| int newl = newr - (r - l - cnt); |
| return query(mid + 1, R, newl, newr, dep + 1, k - cnt); |
| } |
| } |
| /\*int tree[20][MAXN]; // 表示每层每个位置的值 |
| int sorted[MAXN]; // 已经排序好的数 |
| int toleft[20][MAXN]; // toleft[p][i]表示第i层从1到i有数分入左边\*/ |
| int main(){ |
| int n, m; //n个数 , m次查询 |
| while (scanf("%d%d", &n, &m) == 2){ |
| memset(tree, 0, sizeof(tree)); |
| for (int i = 1; i <= n; i++){ |
| scanf("%d", &tree[0][i]); |
| sorted[i] = tree[0][i]; |
| } |
| sort(sorted + 1, sorted + n + 1); |
| build(1, n, 0); |
| int s, t, k; |
| while(m--){ |
| scanf("%d%d%d", &s, &t, &k); |
| printf("%d\n", query(1, n, s, t, 0, k)); |
| } |
| } |
| return 0; |
| } |

1. 伸展树,区间子序列最小值

|  |
| --- |
| /\* 伸展树(Splay Tree) |
| \* 题目:维修数列。 |
| \* 经典题,插入、删除、修改、翻转、求和、求和最大的子序列\*/ |
| #define Key\_value ch[ch[root][1]][0] |
| const int MAXN = 500010; |
| const int INF = 0x3f3f3f3f; |
| int pre[MAXN], ch[MAXN][2], key[MAXN], size[MAXN]; |
| int root, tot1; |
| int sum[MAXN], rev[MAXN], same[MAXN]; |
| int lx[MAXN], rx[MAXN], mx[MAXN]; |
| int s[MAXN], tot2; // 内存池和容量 |
| int a[MAXN]; |
| int n, q; |
| // debug Start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| void Treavel(int x){ |
| if (x){ |
| Treavel(ch[x][0]); |
| printf("结点:%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d\n", x, ch[x][0], ch[x][1], pre[x], size[x]); |
| Treavel(ch[x][1]); |
| } |
| return ; |
| } |
| void debug(){ |
| printf("root:%d\n", root); |
| Treavel(root); |
| return ; |
| } |
| // debug End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| void NewNode(int &r, int father, int k){ |
| if (tot2) r = s[tot2--]; // 取的时候是tot2--,存的时候就是++tot2 |
| else r = ++tot1; |
| pre[r] = father; |
| ch[r][0] = ch[r][1] = 0; |
| key[r] = k; |
| sum[r] = k; |
| rev[r] = same[r] = 0; |
| lx[r] = rx[r] = mx[r] = k; |
| size[r] = 1; |
| return ; |
| } |
| void Update\_Rev(int r){ |
| if (!r) return ; |
| swap(ch[r][0], ch[r][1]); |
| swap(lx[r], rx[r]); |
| rev[r] ^= 1; |
| return ; |
| } |
| void Update\_Same(int r, int v) |
| { |
| if (!r) return ; |
| key[r] = v; |
| sum[r] = v \* size[r]; |
| lx[r] = rx[r] = mx[r] = max(v, v \* size[r]); |
| same[r] = 1; |
| return ; |
| } |
| void push\_up(int r){ |
| int lson = ch[r][0], rson = ch[r][1]; |
| size[r] = size[lson] + size[rson] + 1; |
| sum[r] = sum[lson] + sum[rson] + key[r]; |
| lx[r] = max(lx[lson], sum[lson] + key[r] + max(0, lx[rson])); |
| rx[r] = max(rx[rson], sum[rson] + key[r] + max(0, rx[lson])); |
| mx[r] = max(0, rx[lson]) + key[r] + max(0, lx[rson]); |
| mx[r] = max(mx[r], max(mx[lson], mx[rson])); |
| return ; |
| } |
| void push\_down(int r){ |
| if (same[r]){ |
| Update\_Same(ch[r][0], key[r]); |
| Update\_Same(ch[r][1], key[r]); |
| same[r] = 0; |
| } |
| if(rev[r]){ |
| Update\_Rev(ch[r][0]); |
| Update\_Rev(ch[r][1]); |
| rev[r] = 0; |
| } |
| return ; |
| } |
| void Build(int &x, int l, int r, int father){ |
| if (l > r) return ; |
| int mid = (l + r) / 2; |
| NewNode(x, father, a[mid]); |
| Build(ch[x][0], l, mid - 1, x); |
| Build(ch[x][1], mid + 1, r, x); |
| push\_up(x); |
| return ; |
| } |
| void Init(){ |
| root = tot1 = tot2 = 0; |
| ch[root][0] = ch[root][1] = size[root] = pre[root] = 0; |
| same[root] = rev[root] = sum[root] = key[root] = 0; |
| lx[root] = rx[root] = mx[root] = -INF; |
| NewNode(root, 0, -1); |
| NewNode(ch[root][1], root, -1); |
| for (int i = 0; i < n; i++) scanf("%d", &a[i]); |
| Build(Key\_value, 0, n - 1, ch[root][1]); |
| push\_up(ch[root][1]); |
| push\_up(root); |
| } |
| // 旋转,0为左旋,1为右旋 |
| void Rotate(int x,int kind){ |
| int y = pre[x]; |
| push\_down(y); |
| push\_down(x); |
| ch[y][!kind] = ch[x][kind]; |
| pre[ch[x][kind]] = y; |
| if (pre[y]) |
| ch[pre[y]][ch[pre[y]][1]==y] = x; |
| pre[x] = pre[y]; |
| ch[x][kind] = y; |
| pre[y] = x; |
| push\_up(y); |
| } |
| // Splay调整,将r结点调整到goal下面 |
| void Splay(int r, int goal){ |
| push\_down(r); |
| while (pre[r] != goal){ |
| if (pre[pre[r]] == goal){ |
| push\_down(pre[r]); |
| push\_down(r); |
| Rotate(r, ch[pre[r]][0] == r); |
| } |
| else{ |
| push\_down(pre[pre[r]]); |
| push\_down(pre[r]); |
| push\_down(r); |
| int y = pre[r]; |
| int kind = ch[pre[y]][0] == y; |
| if (ch[y][kind] == r){ |
| Rotate(r, !kind); |
| Rotate(r, kind); |
| } |
| else{ |
| Rotate(y, kind); |
| Rotate(r, kind); |
| } |
| } |
| } |
| push\_up(r); |
| if (goal == 0) root = r; |
| return ; |
| } |
| int Get\_kth(int r, int k){ |
| push\_down(r); |
| int t = size[ch[r][0]] + 1; |
| if (t == k) return r; |
| if (t > k) return Get\_kth(ch[r][0], k); |
| else return Get\_kth(ch[r][1], k - t); |
| } |
| // 在第pos个数后面插入tot个数 |
| void Insert(int pos, int tot){ |
| for (int i = 0; i < tot; i++) scanf("%d",&a[i]); |
| Splay(Get\_kth(root, pos + 1), 0); |
| Splay(Get\_kth(root, pos + 2), root); |
| Build(Key\_value, 0, tot - 1, ch[root][1]); |
| push\_up(ch[root][1]); |
| push\_up(root); |
| return ; |
| } |
| // 删除子树 |
| void erase(int r){ |
| if (!r) return ; |
| s[++tot2] = r; |
| erase(ch[r][0]); |
| erase(ch[r][1]); |
| return ; |
| } |
| // 从第pos个数开始连续删除tot个数 |
| void Delete(int pos, int tot){ |
| Splay(Get\_kth(root, pos), 0); |
| Splay(Get\_kth(root, pos + tot + 1), root); |
| erase(Key\_value); |
| pre[Key\_value] = 0; |
| Key\_value = 0; |
| push\_up(ch[root][1]); |
| push\_up(root); |
| return ; |
| } |
| // 将从第pos个数开始的连续的tot个数修改为c |
| void Make\_Same(int pos, int tot, int c){ |
| Splay(Get\_kth(root, pos), 0); |
| Splay(Get\_kth(root, pos + tot + 1), root); |
| Update\_Same(Key\_value, c); |
| push\_up(ch[root][1]); |
| push\_up(root); |
| return ; |
| } |
| // 将第pos个数开始的连续tot个数进行反转 |
| void Reverse(int pos, int tot){ |
| Splay(Get\_kth(root, pos), 0); |
| Splay(Get\_kth(root,pos+tot + 1), root); |
| Update\_Rev(Key\_value); |
| push\_up(ch[root][1]); |
| push\_up(root); |
| return ; |
| } |
| // 得到第pos个数开始的tot个数的和 |
| int Get\_Sum(int pos, int tot){ |
| Splay(Get\_kth(root, pos), 0); |
| Splay(Get\_kth(root, pos + tot + 1), root); |
| return sum[Key\_value]; |
| } |
| // 得到第pos个数开始的tot个数中最大的子段和 |
| int Get\_MaxSum(int pos, int tot){ |
| Splay(Get\_kth(root, pos), 0); |
| Splay(Get\_kth(root, pos + tot + 1), root); |
| return mx[Key\_value]; |
| } |
| void InOrder(int r){ |
| if (!r) return ; |
| push\_down(r); |
| InOrder(ch[r][0]); |
| printf("%d ",key[r]); |
| InOrder(ch[r][1]); |
| return ; |
| } |
| int main(){ |
| // freopen("in.txt", "r", stdin); |
| // freopen("out.txt", "w", stdout); |
| while (scanf("%d%d", &n, &q) == 2){ |
| Init(); |
| char op[20]; |
| int x, y, z; |
| while (q--){ |
| scanf("%s", op); |
| if (strcmp(op, "INSERT") == 0){ |
| scanf("%d%d", &x, &y); |
| Insert(x, y); |
| } |
| else if (strcmp(op, "DELETE") == 0){ |
| scanf("%d%d", &x, &y); |
| Delete(x,y); |
| } |
| else if (strcmp(op, "MAKE-SAME") == 0){ |
| scanf("%d%d%d", &x, &y, &z); |
| Make\_Same(x, y, z); |
| } |
| else if (strcmp(op, "REVERSE") == 0){ |
| scanf("%d%d", &x, &y); |
| Reverse(x, y); |
| } |
| else if (strcmp(op, "GET-SUM") == 0){ |
| scanf("%d%d", &x, &y); |
| printf("%d\n", Get\_Sum(x, y)); |
| } |
| else if (strcmp(op, "MAX-SUM") == 0){ |
| printf("%d\n", Get\_MaxSum(1, size[root] - 2)); |
| } |
| } |
| } |
| return 0; |
| } |

1. 树状数组(全功能)

|  |
| --- |
| #define Max 500010 |
| int N; |
| int data[Max] = {0}; |
| int C[Max] = {0}; //差分数组 |
| int C2[Max] = {0}; // C2[i] = (i-1)\*C[i] |
| int BIT[Max] = {0}; |
| int BIT1[Max] = {0}; |
| int BIT2[Max] = {0}; |
| int lowbit(int x){ return (x)&(-x);} |
| int getsum(int x){ // 数据数组求和 |
| int sum = 0; |
| for(;x > 0;x-=lowbit(x)){ sum+=BIT[x]; } |
| return sum; |
| } |
| int getsum1(int x){ // 差分数组求和 |
| int sum = 0; |
| for(;x > 0;x-=lowbit(x)){ sum+=BIT1[x]; } |
| return sum; |
| } |
| int getsum2(int x){ |
| int sum = 0; |
| for(;x > 0;x-=lowbit(x)){ sum+=BIT2[x]; } |
| return sum; |
| } |
| void add(int i,int add){ // 数据BIT更新 |
| for (;i <= N;i+=lowbit(i)){ BIT[i]+=add; } |
| } |
| void add1(int i,int add){// 差分BIT更新 |
| for(;i <= N;i+=lowbit(i)){ BIT1[i]+=add;} |
| } |
| void add2(int i,int add){ |
| for(;i <= N;i+=lowbit(i)){ BIT2[i]+=add;} |
| } |
| int main(){ |
| //输入数据到data , 更新BIT数组 |
| /\* 区间修改, 单点查询 |
| \* C[i] = data[i]-data[i-1] |
| \* add1(i,C[i]) |
| \* 第x个数为getsum1(x)\*/ |
| /\*区间修改, 区间查询 |
| \* C2[i] = (i-1)\*C[i] |
| \* add2(i,C2[i]) |
| \* 前n项和为n\*getsum1(n)-getsum2(n)\*/ |
| /\*cin >> N; |
| for(int i = 1;i <= N;i++){ 输入数据 |
| cin >> data[i]; |
| C[i] = data[i]-data[i-1]; |
| C2[i] = (i-1)\*C[i]; |
| add1(i,C[i]); |
| add2(i,C2[i]); |
| } |
| int l,r; |
| cin >> l >> r;修改区间 |
| int v; |
| cin >> v; |
| C[l] +=v; |
| C[r+1] -=v; |
| add1(l,v); |
| add1(r+1,-v); |
| int x; |
| cin >> x; 查询单点 |
| cout << getsum1(x) << endl; |
| C2[l] += v\*(l-1); |
| C2[r+1] += (-v)\*(r); |
| add2(l,v\*(l-1)); |
| add2(r+1,(-v)\*r); |
| cin >> l >> r; 查询区间 |
| cout << r\*getsum1(r)-getsum2(r)-((l-1)\*getsum1(l-1)-getsum2(l-1)) << endl; \*/ |
| } |

1. 线段树

|  |
| --- |
| //tree[]表示每个区间的最小值,sum[]区间和,col[]打标记,表明该区间每个元素增加了多少 |
| //懒标记用在update上，意思是可以不用都更新，用到的时候再更新 |
| void build(int node,int begin,int end){ |
| if(begin==end){ |
| tree[node]=array[begin]; |
| sum[node]=array[begin]; |
| return; |
| } |
| int mid=begin+(end-begin) |
| build(2\*node,begin,mid); |
| build(2\*node+1,mid+1,end); |
| //用tree记录每个区间的最值 |
| if(tree[node\*2]<=tree[node\*2+1]) tree[node]=tree[2\*node]; |
| else tree[node]=tree[2\*node+1]; |
| } |
| //区间查询 |
| int query1(int node,int begin,int end,int left,int right){//查询3-5之间的最小值 |
| int p1,p2; |
| if(left>end || right<begin) return -1; |
| if(begin>=right && end<=right) return tree[node]; |
| int mid=begin+(end-begin)/2; |
| p1=query(2\*node,begin,mid); |
| p2=query(2\*node+1,mid+1,end); |
| if(p1==-1)return p2; |
| if(p2==-1)return p1; |
| if(p1<=p2)return p1; |
| else return p2; |
| } |
| //单点更新 |
| void updata(int node ,int begin,int end,int ind,int add){//将哪个点ind更新了 add |
| if(begin==end) tree[node]+=add; |
| int mid=begin+(end-begin)/2; |
| if(ind<=m) updata(node\*2,left,mid,ind,add); |
| else updata(node\*2+1,mid+1,right,ind,add); |
| tree[node]=min(tree[node\*2,node\*2+1]); |
| } |
| //下放标记 |
| void pushdown(int rt,int m){ |
| if(col[rt]){ |
| col[rt<<1]=col[rt]; |
| col[rt<<1|1]=col[rt]; |
| sum[rt<<1]=col[rt]\*(m-m/2);//这个区间要稍微大一点 |
| sum[rt<<1|1]=col[rt]\*(m/2); |
| col[rt]=0; |
| } |
| } |
| //区间修改 |
| //以修改区间的值并求整个数组的和为例 |
| void change(int node,int begin,int end,int left,int right,int c){ |
| if(left<=begin && end >=right){ |
| col[node]=c;//在该节点上打一个标记 标记打给谁了，就一个么？ |
| sum[node]=c\*(end-begin+1); //做和的时候用了，但是查询的时候不是这个区间了，这时就用到标记下放 |
| return; |
| } |
| pushdown(node,end-begin+1); |
| int mid=begin+(end-begin)/2; |
| if(left<=mid) change(node\*2,begin,end,left,mid,c) ; |
| if(right>mid) change(node\*2+1,begin,end,mid+1,right,c); |
| sum[node]=sum[node\*2]+sum[node\*2+1]; |
| } |
| //查询区间之和 |
| int query2(int node,int begin,int end,int left,int right){ |
| if(left<=begin && end>=right) return sum[node]; //不需要就不下放了 |
| Pushdown(node,end-begin+1);//需要的时候再下放 |
| int ret=0; |
| int mid=begin+(end-begin)/2; |
| if(left<mid) ret+=query2(node\*2,begin,end,left,mid,c) ; |
| if(right>mid) ret+=query2(node\*2+1,begin,end,mid+1,right,c); |
| return ret; |
| } |

1. 主席树,静态区间第k小

|  |
| --- |
| const int MAXN = 100010; |
| const int M = MAXN \* 30; |
| int n, q, m, tot; |
| int a[MAXN], t[MAXN]; |
| int T[MAXN], lson[M], rson[M], c[M]; |
| void Init\_hash(){ |
| for (int i = 1; i <= n; i++) t[i] = a[i]; |
| sort(t + 1, t + 1 + n); |
| m = (int)(unique(t + 1, t + 1 + n) - t - 1); |
| } |
| int build(int l, int r){ |
| int root = tot++; c[root] = 0; |
| if (l != r){ |
| int mid = (l + r) >> 1; |
| lson[root] = build(l, mid); |
| rson[root] = build(mid + 1, r); |
| } |
| return root; |
| } |
| int hash\_(int x){ |
| return (int)(lower\_bound(t + 1, t + 1 + m, x) - t); |
| } |
| int update(int root, int pos, int val){ |
| int newroot = tot++, tmp = newroot; |
| c[newroot] = c[root] + val; |
| int l = 1, r = m; |
| while (l < r){ |
| int mid = (l + r) >> 1; |
| if (pos <= mid){ |
| lson[newroot] = tot++; |
| rson[newroot] = rson[root]; |
| newroot = lson[newroot]; |
| root = lson[root]; |
| r = mid; |
| } |
| else{ |
| rson[newroot] = tot++; |
| lson[newroot] = lson[root]; |
| newroot = rson[newroot]; |
| root = rson[root]; |
| l = mid + 1; |
| } |
| c[newroot] = c[root] + val; |
| } |
| return tmp; |
| } |
| int query(int left\_root, int right\_root, int k){ |
| int l = 1, r = m; |
| while ( l < r){ |
| int mid = (l + r) >> 1; |
| if (c[lson[left\_root]] - c[lson[right\_root]] >= k ){ |
| r = mid; |
| left\_root = lson[left\_root]; |
| right\_root = lson[right\_root]; |
| } |
| else{ |
| l = mid + 1; |
| k -= c[lson[left\_root]] - c[lson[right\_root]]; |
| left\_root = rson[left\_root]; |
| right\_root = rson[right\_root]; |
| } |
| } |
| return l; |
| } |
| int main(){ |
| // freopen("in.txt","r",stdin); |
| // freopen("out.txt","w",stdout); |
| while (scanf("%d%d", &n, &q) == 2){ |
| tot = 0; |
| for (int i = 1; i <= n; i++) scanf("%d", &a[i]); |
| Init\_hash(); |
| T[n + 1] = build(1, m); |
| for (int i = n; i; i--){ |
| int pos = hash\_(a[i]); |
| T[i] = update(T[i + 1], pos, 1); |
| } |
| while (q--){ |
| int l, r, k; |
| scanf("%d%d%d", &l, &r, &k); |
| printf("%d\n", t[query(T[l], T[r + 1], k)]); |
| } |
| } |
| return 0; |
| } |

1. 主席树,区间有多少不重复的数

|  |
| --- |
| /\*给出一个序列,查询区间内有多少个不相同的数 \*/ |
| const int MAXN = 30010; |
| const int M = MAXN \* 100; |
| int n, q, tot; |
| int a[MAXN]; |
| int T[MAXN], lson[M], rson[M], c[M]; |
| int build(int l, int r){ |
| int root = tot++; |
| c[root] = 0; |
| if (l != r){ |
| int mid = (l + r) >> 1; |
| lson[root] = build(l, mid); |
| rson[root] = build(mid + 1, r); |
| } |
| return root; |
| } |
| int update(int root, int pos, int val){ |
| int newroot = tot++, tmp = newroot; |
| c[newroot] = c[root] + val; |
| int l = 1, r = n; |
| while (l < r){ |
| int mid = (l + r) >> 1; |
| if (pos <= mid){ |
| lson[newroot] = tot++; |
| rson[newroot] = rson[root]; |
| newroot = lson[newroot]; |
| root = lson[root]; |
| r = mid; |
| } |
| else{ |
| rson[newroot] = tot++; |
| lson[newroot] = lson[root]; |
| newroot = rson[newroot]; |
| root = rson[root]; |
| l = mid + 1; |
| } |
| c[newroot] = c[root] + val; |
| } |
| return tmp; |
| } |
| int query(int root, int pos){ |
| int ret = 0; |
| int l = 1, r = n; |
| while (pos < r){ |
| int mid = (l + r) >> 1; |
| if (pos <= mid){ |
| r = mid; |
| root = lson[root]; |
| } |
| else{ |
| ret += c[lson[root]]; |
| root = rson[root]; |
| l = mid + 1; |
| } |
| } |
| return ret + c[root]; |
| } |
| int main(){ |
| // freopen("in.txt", "r", stdin); |
| // freopen("out.txt", "w", stdout); |
| while (scanf("%d", &n) == 1){ |
| tot = 0; |
| for (int i = 1; i <= n; i++) scanf("%d", &a[i]); |
| T[n + 1] = build(1, n);//T为树 |
| map<int,int> mp; |
| for (int i = n; i >= 1; i--){ |
| if (mp.find(a[i]) == mp.end()) T[i] = update(T[i + 1], i, 1); |
| else{ |
| int tmp = update(T[i + 1], mp[a[i]], -1); |
| T[i] = update(tmp, i, 1); |
| } |
| mp[a[i]] = i; |
| } |
| scanf("%d", &q); |
| while (q--){ |
| int l, r; |
| scanf("%d%d", &l, &r); |
| printf("%d\n", query(T[l], r)); |
| } |
| } |
| return 0; |
| } |

1. 主席树+树状数组,动态区间第k大

|  |
| --- |
| /\*树状数组套主席树\*/ |
| const int MAXN = 60010; |
| const int M = 2500010; |
| int n, q, m, tot; |
| int a[MAXN], t[MAXN]; |
| int T[MAXN], lson[M], rson[M],c[M]; |
| int S[MAXN]; |
| struct Query{ |
| int kind; |
| int l, r, k; |
| } query[10010]; |
| void Init\_hash(int k){ |
| sort(t, t + k); |
| m = (int)(unique(t, t + k) - t); |
| return ; |
| } |
| int hash\_(int x){ |
| return (int)(lower\_bound(t, t + m, x) - t); |
| } |
| int build(int l, int r){ |
| int root = tot++; |
| c[root] = 0; |
| if (l != r){ |
| int mid = (l + r) / 2; |
| lson[root] = build(l, mid); |
| rson[root] = build(mid + 1, r); |
| } |
| return root; |
| } |
| int Insert(int root, int pos, int val){ |
| int newroot = tot++, tmp = newroot; |
| int l = 0, r = m - 1; |
| c[newroot] = c[root] + val; |
| while (l < r){ |
| int mid = (l + r) >> 1; |
| if (pos <= mid){ |
| lson[newroot] = tot++; |
| rson[newroot] = rson[root]; |
| newroot = lson[newroot]; |
| root = lson[root]; |
| r = mid; |
| } |
| else{ |
| rson[newroot] = tot++; |
| lson[newroot] = lson[root]; |
| newroot = rson[newroot]; |
| root = rson[root]; |
| l = mid + 1; |
| } |
| c[newroot] = c[root] + val; |
| } |
| return tmp; |
| } |
| int lowbit(int x){ return x & (-x);} |
| int use[MAXN]; |
| void add(int x, int pos, int val){ |
| while (x <= n){ |
| S[x] = Insert(S[x], pos, val); |
| x += lowbit(x); |
| } |
| return ; |
| } |
| int sum(int x){ |
| int ret = 0; |
| while (x > 0){ |
| ret += c[lson[use[x]]]; |
| x -= lowbit(x); |
| } |
| return ret; |
| } |
| int Query(int left, int right, int k){ |
| int left\_root = T[left - 1]; |
| int right\_root = T[right]; |
| int l = 0, r = m - 1; |
| for (int i = left - 1; i; i -= lowbit(i)) use[i] = S[i]; |
| for (int i = right; i; i -= lowbit(i)) use[i] = S[i]; |
| while (l < r){ |
| int mid = (l + r) / 2; |
| int tmp = sum(right) - sum(left - 1) + c[lson[right\_root]] - c[lson[left\_root]]; |
| if (tmp >= k){ |
| r = mid; |
| for (int i = left - 1; i; i -= lowbit(i)) use[i] = lson[use[i]]; |
| for (int i = right; i; i -= lowbit(i)) use[i] = lson[use[i]]; |
| left\_root = lson[left\_root]; |
| right\_root = lson[right\_root]; |
| } |
| else{ |
| l = mid + 1; |
| k -= tmp; |
| for (int i = left - 1; i; i -= lowbit(i)) use[i] = rson[use[i]]; |
| for (int i = right; i; i -= lowbit(i)) use[i] = rson[use[i]]; |
| left\_root = rson[left\_root]; |
| right\_root = rson[right\_root]; |
| } |
| } |
| return l; |
| } |
| void Modify(int x, int p, int d){ |
| while (x <= n){ |
| S[x] = Insert(S[x], p, d); |
| x += lowbit(x); |
| } |
| return ; |
| } |
| int main(){ |
| // freopen("in.txt", "r", stdin); |
| // freopen("out.txt", "w", stdout); |
| int Tcase; |
| scanf("%d", &Tcase); |
| while (Tcase--){ |
| scanf("%d%d", &n, &q); |
| tot = 0; |
| m = 0; |
| for (int i = 1; i <= n; i++){ |
| scanf("%d", &a[i]); |
| t[m++] = a[i]; |
| } |
| char op[10]; |
| for (int i = 0; i < q; i++){ |
| scanf("%s", op); |
| if (op[0] == 'Q'){ |
| query[i].kind = 0; |
| scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k); |
| } |
| else{ |
| query[i].kind = 1; |
| scanf("%d%d", &query[i].l, &query[i].r); |
| t[m++] = query[i].r; |
| } |
| } |
| Init\_hash(m); |
| T[0] = build(0, m - 1); |
| for (int i = 1; i <= n; i++) T[i] = Insert(T[i - 1], hash\_(a[i]), 1); |
| for (int i = 1; i <= n; i++) S[i] = T[0]; |
| for (int i = 0; i < q; i++){ |
| if (query[i].kind == 0){ |
| printf("%d\n", t[Query(query[i].l, query[i].r, query[i].k)]); |
| } |
| else{ |
| Modify(query[i].l, hash\_(a[query[i].l]), -1); |
| Modify(query[i].l, hash\_(query[i].r), 1); |
| a[query[i].l] = query[i].r; |
| } |
| } |
| } |
| return 0; |
| } |

1. 左偏树 ,小堆的合并,有序序列合并

|  |
| --- |
| /\* 合并复杂度 O(log N) |
| \* INIT: init()读入数据并进行初始化; |
| \* CALL: merge() 合并两棵左偏树; |
| \* ins() 插入一个新节点; |
| \* top() 取得最小结点; |
| \* pop() 取得并删除最小结点; |
| \* del() 删除某结点; |
| \* add() 增/减一个结点的键值; |
| \* iroot() 获取结点i的根;\*/ |
| #define typec int // type of key val |
| const int na = -1; |
| const int N = 1010; |
| struct node{ |
| typec key; |
| int l, r, f, dist; |
| } tr[N]; |
| int iroot(int i) |
| { // find i's root |
| if (i == na) return i; |
| while (tr[i].f != na){ i = tr[i].f;} |
| return i; |
| } |
| int merge(int rx, int ry){ |
| // two root: rx, ry |
| if (rx == na) return ry; |
| if (ry == na) return rx; |
| if (tr[rx].key > tr[ry].key) swap(rx, ry); |
| int r = merge(tr[rx].r, ry); |
| tr[rx].r = r; |
| tr[r].f = rx; |
| if (tr[r].dist > tr[tr[rx].l].dist) swap(tr[rx].l, tr[rx].r); |
| if (tr[rx].r == na) tr[rx].dist = 0; |
| else tr[rx].dist = tr[tr[rx].r].dist + 1; |
| return rx; // return new root |
| } |
| int ins(int i, typec key, int root) |
| { // add a new node(i, key) |
| tr[i].key = key; |
| tr[i].l = tr[i].r = tr[i].f = na; |
| tr[i].dist = 0; |
| return root = merge(root, i); // return new root |
| } |
| int del(int i) |
| { // delete node i |
| if (i == na) return i; |
| int x, y, l, r; |
| l = tr[i].l; |
| r = tr[i].r; |
| y = tr[i].f; |
| tr[i].l = tr[i].r = tr[i].f = na; |
| tr[x = merge(l, r)].f = y; |
| if (y != na && tr[y].l == i) tr[y].l = x; |
| if (y != na && tr[y].r == i) tr[y].r = x; |
| for (; y != na; x = y, y = tr[y].f){ |
| if (tr[tr[y].l].dist < tr[tr[y].r].dist) swap(tr[y].l, tr[y].r); |
| if (tr[tr[y].r].dist + 1 == tr[y].dist) break; |
| tr[y].dist = tr[tr[y].r].dist + 1; |
| } |
| if (x != na) return iroot(x); // return new root |
| else return iroot(y); |
| } |
| node top(int root){ return tr[root];} |
| node pop(int &root){ |
| node out = tr[root]; |
| int l = tr[root].l, r = tr[root].r; |
| tr[root].l = tr[root].r = tr[root].f = na; |
| tr[l].f = tr[r].f = na; |
| root = merge(l, r); |
| return out; |
| } |
| int add(int i, typec val){ // tr[i].key += val |
| if (i == na) return i; |
| if (tr[i].l == na && tr[i].r == na && tr[i].f == na){ |
| tr[i].key += val; |
| return i; |
| } |
| typec key = tr[i].key + val; |
| int rt = del(i); |
| return ins(i, key, rt); |
| } |
| void init(int n){ |
| for (int i = 1; i <= n; i++){ |
| scanf("%d", &tr[i].key); // %d: type of key |
| tr[i].l = tr[i].r = tr[i].f = na; |
| tr[i].dist = 0; |
| } |
| return ; |
| } |

1. kjlij
2. Fsjlaf