备战篮球杯

初级图论算法：

最短路：

dijkstra

$O(N^2)$

```
//const int oo = 0x0fffffff;//无穷大。一般自己的主程序模板自带。

int n;//边的编号。（小心变量重复。）

int g[N][N];//表示两个节点之间的边权。graph
bool v[N];  //表示节点当前最短路是否已知。

void dijkstra()
{
    fill(v , v + 1 + n, false);
    for(int i = 0; i < n; i++)d[i] = (i == 0 ? 0 : oo);
    for(int i = 0; i < n; i++)
```

```
    {
        int x, m = oo;
        // 选未确定最小的点更新
        for(int y = 0; j < n; j++)if(!v[y] && d[y] < m) m = d[x = y];
        v[x] = 1;
        for(int y = 0; j < n; j++)d[y] = min(d[y] , d[x] + g[x][y]);
    }
}
```

容易出现问题细节：

1. mx 和 d[mx]
2. 初始mx要小心从0 - 1等值开始。避免其开始定义就是一个已经处理的值。导致其它点不会被更新。
3. 避免爆int

$O(logN)$

```
//边结构的存储
struct Edge {
    int from, to, dist;
    Edge(int u, int v, int d) : from(u), to(v), dist(d) {}
};

const int oo = 1E9 + 10;
const int maxn = 2E5 + 10;//最大节点个数。
struct Dijkstra {
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn]; //保存了
    bool done[maxn];
    int d[maxn];       //s到各个点的距离
    int p[maxn];       //记录上一条弧

    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)G[i].clear();
        edges.clear();
    }

    void AddEdge(int from, int to, int dist)
    {
        edges.push_back(Edge(from, to, dist));
        m = edges.size();
```

```
28          G[from].push_back(m - 1);
29      }
30
31      //优先队列使用的结构。当然可以使用pair来储存。
32      struct HeapNode {
33          int u;
34          int d;
35          //优先最小堆
36          //这里怎么那么奇怪？发现一些问题。
37          //抛开习惯。大于小于不过只是函数名称。
38          bool operator <(const HeapNode& rhs)const {
39              return d > rhs.d;
40          }
41      };
42
43      void dijkstra(int s)
44      {
45          priority_queue<HeapNode>que;
46          for (int i = 0; i < n; i++)d[i] = oo;
47          fill(done, done + n, false);
48          que.push({ s , 0 });
49          d[s] = 0;
50          while (que.empty() == false)
51          {
52              int u = que.top().u;que.pop();
53              if (done[u])continue;
54              done[u] = true;
55              //检查所有边进行更新
56              for (auto i : G[u])
57              {
58                  int v = edges[i].to;
59                  if (d[v] > d[u] + edges[i].dist)
60                  {
61                      d[v] = d[u] + edges[i].dist;
62                      p[v] = i;
63                      que.push({ v , d[v] });
64                  }
65              }
66          }
67      }
68
69  }dij;
70
71  /*
72  * 1.不要在局部函数中定义对象。
73  * 2.注意数据范围。考虑将int，改为ll.
```

```
74  * 3.改模板节点管理：节点的下标从0开始。注意是否输入统一。
75  */
```

更友好一点的板子。

```
1   vector<pair<int , int>> g[N];
2   int d[N] , n;
3   bool done[N];
4   const int inf = 1E9 + 7;
5
6   void dijkstra(int s) {
7       fill(d , d + n + 1 , (1LL << 31) - 1);
8       d[s] = 0;
9
10      priority_queue<pair<ll , int>> que;
11      que.push({0 , s});
12
13      while (que.size()) {
14          // 首先取出最短的点。
15          int cur = que.top().second;
16          que.pop();
17          // cout << mx << "\n";
18          if (done[cur])continue;
19          done[cur] = true;
20          for (auto [j , w] : g[cur]) {
21              if (d[j]  >  d[cur] + w) {
22                  d[j] = d[cur] + w;
23                  que.push({ -d[j] , j});
24              }
25          }
26          // dbg();
27      }
28  }
29  signed main() {
30      fsio;
31      //
32      int m , s;
33      cin >> n >> m >> s;
34      for (int i = 0; i < m; i++) {
35          int u , v , w;
36          cin >> u >> v >> w;
37          g[u].push_back({v , w});
38          // g[v].push_back({u , w});
39      }
40      dijkstra(s);
```

```
    for (int i = 1; i <= n; i++) {
        cout << d[i] << " \n"[i == n];
    }
}

```

Bellman_ford

```
for(int i = 0; i < n; i++)d[i] = oo;
d[0] = 0;
for(int k = 0; k < n - 1; i++)
    for(int i = 0; i < m; i++)
    {
        int x = u[i] , y = v[i];
        if(d[x] < oo) d[y] = min(d[y] , d[x] + w[i]);
    }
```

spfa

```
//边结构的存储
struct Edge {
    int from, to, dist;
    Edge(int u, int v, int d) : from(u), to(v), dist(d) {}
};

const int oo = 0x7fffffff;
const int maxn = 5E5 + 10;//最大节点个数。
struct BellmanFord {
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn]; //保存了
    bool inq[maxn];
    int cnt[maxn];
    ll d[maxn];       //s到各个点的距离
    int p[maxn];       //记录上一条弧

    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)G[i].clear();
        edges.clear();
    }

```

```
25      void AddEdge(int from, int to, int dist)
26      {
27          edges.push_back(Edge(from, to, dist));
28          m = edges.size();
29          G[from].push_back(m - 1);
30      }
31
32      bool bellman_ford(int s)
33      {
34          queue<int>que;
35          fill(inq, inq + n, false);
36          fill(cnt, cnt + n, 0);
37          for (int i = 0; i < n; i++)d[i] = oo;
38          d[s] = 0;
39          inq[s] = true;
40          que.push(s);
41          while (que.empty() == false)
42          {
43              int u = que.front(); que.pop();
44              inq[u] = false ;
45              for (int i = 0; i < G[u].size(); i++)
46              {
47                  Edge& e = edges[G[u][i]];
48                  if (d[u] < oo && d[e.to] > d[u] + e.dist)
49                  {
50                      d[e.to] = d[u] + e.dist;
51                      p[e.to] = G[u][i];
52                      if (!inq[e.to]) { que.push(e.to); inq[e.to] = true; }
53                      if (++cnt[e.to] > n) return  false;
54                  }
55              }
56          }
57          return true;
58      }
59  }bellman;
60
61  /*
62  * 1.不要在局部函数中定义对象。
63  * 2.注意数据范围。考虑将int，改为ll.
64  * 3.改模板节点管理：节点的下标从0开始。注意是否输入统一。
65  * 4.记得init();
66  */
```

floyed

```
/*
*初始化:  d[i][i] = 0;
*无边:   d[i][j] = oo;
*ij有边：d[i][j] = w[i][j]
*/
for(int k =0; k < n; k++)
    for(int  i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
```

最小生成树

prim

prim只有麻瓜采用

kruskal

```
const int N = 10010;
const int M = 1E6+10;
int p[N];
int id[M]; // 用来作为替身，定位大小；

int find(int x){return p[x] == x ? x : p[x] = find(p[x]);}

void unit(int x , int y)
{
    x = find (x);
    y = find (y);
    p[x] = y;
}

struct node{
    int u;
    int v;
    int w;
}e[M];

//当前已经处理好各条边；
ll kruskal()
{
```

```
    ll ans = 0;
    iota(p , p + n ,0);
    iota(id , id + m, 0);

    sort(id , id + m , [&](int x , int y){
        return e[x].w < e[y].w;
    });

    for(int i = 0; i < m; i++)
    {
        int now = id[i];
        int x = find(e[now].u);int y = find(e[now].v);
        if(x != y){ ans += e[now].w; p[x] = y;}
    }
    return ans;
}
```

拓扑排序

提供两种思路：

```
vector<int> order;
vector<vector<int>> g;
bool passed[maxn];
int pos[maxn];

void dfs(int u){
    passed[u] = true;
    for (auto i : g[u])
        if (!passed[i])
            dfs(i);
    order.push_back(u);
}

void topo()
{
    for(int i=0;i<n;i++)
    if(!passed[i])dfs(i);
    reverse(order.begin(), order.end()); //倒转。
}
```

```
    vector<int> topo;
    auto topo_sort = [&]()->void{
```

```
        queue<int> que;
        for (int i = 0; i < n; i++)if (deg[i] == 0) {
                que.push(i);
            }
        while (que.empty() == false) {
            int u = que.front(); que.pop();
            topo.push_back(u);
            for (auto v : g[u]) {
                deg[v]--;
                if (deg[v] == 0)que.push(v);
            }
        }
    };
    topo_sort();
    reverse(topo.begin() , topo.end());
```

初级数论算法

快速幂运算

```
ll quickly_pow(ll x,ll n,ll mod)
{
    ll res=1;//用来返回结果。
    while(n>0)
    {
        if(n&1)res=res*x%mod;
        x=x*x%mod;
        n>>=1;
    }
    return res;
}
```

- 分析模板
  - 当前的剩余部分是否可以被x^2（x实际意义上是多少次方）整除.
    - 如果可以不需要处理，
    - 如果不可以，余出来的部分必然是x,将它先乘在结果之上即可。

素数

埃氏筛法：

- 来自jly的板 欧拉筛

```
int mp[maxn+1];
vector<int>primes;

for (int i = 2; i <= maxn; i++)
    {
        if (!mp[i])
        {
            mp[i] = i;
            primes.push_back(i);
        }
        for (auto p : primes)
        {
            if (p * i > maxn)
                break;
            mp[i * p] = p;
            if (i % p == 0)
                break;
        }
    }
```

- 这个可以方便的拆分素数。

拆数算法

如上通过记录最大素数，来拆分一个数。

欧几里得以及扩展欧几里得算法

问就是 `__gcd(x , y)`

组合数

组合数板子

一共提供了四种求组合数的方法：资料来源于acwing

第一种方法：性质+递推+预处理：

将组合数得计算分解为小规模得问题实现：

$$C_r^n = C_{r-1}^n + C_{r-1}^{n-1} \tag{1}$$

最小规模得子问题如C_r^0, C_r^1已知（容易计算。）

然后就可以把所有情况求出来：

code

```
const int N_c = 3E3;
const int mod = 1E9 + 7;
int c[N_c][N_c];
void C_init() {
    for (int i = 1; i < N_c; ++i) {
        c[i][0] = c[i][i] = 1;
        for (int j = 1; j < i; ++j) {
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
        }
    }
}
//注意范围
//小心me
```

tips

1. 数组范围允许。

第二种方法： 逆元+预处理+组合数定义

basic

$$C_n^a = \frac{n!}{(n-a)! \times b!} \tag{2}$$

可以考虑先将分母得阶乘法求出。然后求出其逆元（由于mod是一个质数，逆元可以通过费马小定理求出。）

综上：时间复杂度：

$n + alog(a)$:

AcWing 886. 求组合数 II-数论-C++ - AcWing

code

```
using ll = long long;
mod数， 数组大小相等?
const int N_c = 2E5 + 10;
const int mod = 1e9 + 7;

int fac[N_c] , infac[N_c];
ll quickly_pow(ll x, ll n, ll p)
{
    ll res = 1;
    while (n > 0)
    {
        if (n & 1)res = res * x % p;
        x = x * x % p;
        n >>= 1;
    }
    return res;
}
void init() {
    fac[0] = infac[0] = 1;
    for (int i = 1; i < N_c; i++)
    {
        fac[i] = 1LL * fac[i - 1] * i % mod;
        infac[i] = 1LL * infac[i - 1] * quickly_pow(i, mod - 2, mod) % mod;
    }
}
int c(int a , int b) {
    return 1LL * fac[a] * infac[b] % mod * infac[a - b] % mod;
}
/*
*记得初始化。
*/
```

逆元

乘法逆元（inverse element）及四大相关求法详解（含证明）_NothingAtall.的博客-CSDN博客_乘法逆元

还得看大佬博客

定义

对于线性同余方程$ax \equiv 1(mod \quad b)$

称x为a mod b的逆元记作a^{-1}

性质

逆元存在：

1. 当$gcd(a, mod) = 1$时，逆元才有解。

逆元计算

费马小定理：）

1. 内容：

$$
\begin{gathered}
1.\text{假如}a\text{是一个整数。}m\text{是质数。} \\
a^m \equiv a(mod \quad m)
\end{gathered} \tag{3}
$$

$$
\begin{gathered}
2.\text{假设}m\text{不是质数，}gcd(a, m) = 1 \\
a^{m-1} \equiv 1(mod \quad m) \\
a * a^{m-2} \equiv 1(mod \equiv m) \\
\text{综上}a^{m-2}\text{是}a\text{的逆元。}
\end{gathered} \tag{4}
$$

- 前提约束：mod数是质数。

其实就是写一个快速幂运算。

```
//inverse_element.
ll mod;
ll qpow( ll x , ll n , ll p = mod)
{
    ll res = 1;
    while (n > 0) {
        if (n & 1) res = res * x % p;
        x = x * x % mod;
```

```
        n >>= 1;
    }
    return res;
}

ll inv(ll x , ll p = mod)
{
    return qpow( x , p - 2);
}
/*
* 1. mod定义
* 2. 使用前提：p是质数，且x，p互质。
*/
```

其它的有机会再补。靠队友啦 qaq

初级数据结构

树上问题

LCA

倍增与树

利用倍增的思想， 可以给树上的点建立父亲st表：
应用st表： 可以完成以下任务：

1. 最近公共祖先：

   原理： 可以用$2^0.....2^{31}$这些二进制数字， 且每个数字只使用一次， 拼凑出所有的在范围内的数字。

code

```
const int LOGN = 18;
int n , q;
vector<int> e[N];
int par[LOGN + 1][N] , dep[N];
```

```
5   void dfs(int u , int fa) {
6       dep[v] = dep[u] + 1;
7       for (auto v : e[u]) {
8           if (v != fa) {
9               par[0][v] = u;
10              dfs(v , u);
11          }
12      }
13  }
14  int query(int u , int v) {
15      if (dep[u] > dep[v]) swap(u , v);
16      int d = dep[v] - dep[u];
17      for (int i = LOGN; i >= 0; i--) {
18          if ((1 << i) <= d) {
19              d -= (1 << i);
20              v = par[i][v];
21          }
22      }
23      if (v == u) return v;
24      // 降到同一个高度上;
25      // 接着， 两个一起跳;
26      for (int i = LOGN; i >= 0; i--) {
27          // 如果不是同一个父亲就跳。
28          // 如果是同一个， 就用更小的操作去跳。
29          // 最后它们会相差1.
30          if (par[i][v] != par[i][u]) {
31              v = par[i][v];
32              u = par[i][u];
33          }
34      }
35      return par[0][u];
36  }
37  int main() {
38      int s;
39      cin >> n >> q >> s;
40      for (int i = 1; i < n; i++) {
41          int u , v;
42          cin >> u >> v;
43          e[u].push_back(v);
44          e[v].push_back(u);
45      }
46      dfs(s , 0);
47      for (int i = 1; i <= LOGN; i++) {
48          for (int j = 1; j <= n; j++) {
49              par[i][j] = par[i - 1][par[i - 1][j]];
50          }
```

```
    }
    for (int i = 0; i < q; i++) {
        int u , v;
        cin >> u >> v;
        cout << query(u , v) << "\n";
    }
}
```

2. 两点之间的路径最小值：

http://oj.daimayuan.top/course/15/problem/793

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

const int N = 201000;
const int LOGN = 18;

int n, q;
int dep[N], par[N][LOGN + 1], val[N][LOGN + 1];
vector<pair<int, int>> e[N];

void dfs(int u, int f) {
    dep[u] = dep[f] + 1;
    for (auto p : e[u]) {
        int v = p.first;
        if (v == f) continue;
        par[v][0] = u;
        val[v][0] = p.second;
        dfs(v, u);
    }
}
int query(int u, int v) {
    int ans = 1 << 30;
    if (dep[u] > dep[v]) swap(u, v);
    int d = dep[v] - dep[u];
    for (int j = LOGN; j >= 0; j--) if (d & (1 << j)) {
        ans = min(ans, val[v][j]);
        v = par[v][j];
    }
    if (u == v) return ans;
    for (int j = LOGN; j >= 0; j--) if (par[u][j] != par[v][j]) {
        ans = min(ans, min(val[u][j], val[v][j]));
        u = par[u][j];
        v = par[v][j];
```

```
    }
    ans = min(ans, min(val[u][0], val[v][0]));
    return ans;
}

int main() {
    scanf("%d%d",&n, &q);
    for (int i = 1; i < n; i++) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        e[u].push_back(make_pair(v, w));
        e[v].push_back(make_pair(u, w));
    }
    dfs(1, 0);
    for (int j = 1; j <= LOGN; j++) {
        for (int u = 1; u <= n; u++) {
            par[u][j] = par[par[u][j - 1]][j - 1];
            val[u][j] = min(val[u][j - 1], val[par[u][j - 1]][j - 1]);
        }
    }
    for (int i = 1; i <= q; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        printf("%d\n", query(u, v));
    }
}
```

3. 路径上的最小点权值；
4. 第k祖先：略；

树上差分

树上直径

树的直径

概念

直径： 两点之间的最大距离。

求直径的方法：

1. 树形dp求直接直径：
2. 两次bfs求直径：

拓展

1. 给定一个点， 求出其与其它点之间的最大距离：
    1. 换根dp求法： 比较困难。
    2. 从直径上的两个点开始进行bfs。求出最大距离。

bfs方法下求

依然是下述的例题：

jls代码

```cpp
#include <bits/stdc++.h>

using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n;
    std::cin >> n;

    std::vector<std::vector<int>> adj(n);
    for (int i = 1; i < n; i++) {
        int u, v;
        std::cin >> u >> v;
        u--, v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    std::vector<int> d(n, -1);
    auto bfs = [&](int x) {
        std::queue<int> q;
        d.assign(n, -1);
        q.push(x);
        d[x] = 0;

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

            for (auto y : adj[x]) {
                if (d[y] == -1) {
                    d[y] = d[x] + 1;
```

```
                q.push(y);
            }
        }
    }

    return std::max_element(d.begin(), d.end()) - d.begin();
};

int x = bfs(0);
int y = bfs(x);
auto dx = d;
bfs(y);
auto dy = d;

std::vector<int> ans(n + 1);
for (int i = 0; i < n; i++) {
    ans[std::max(dx[i], dy[i]) + 1] += 1;
}
ans[0] += 1;
ans[dx[y] + 1] -= 1;
for (int i = 1; i <= n; i++) {
    ans[i] += ans[i - 1];
}
for (int i = 1; i <= n; i++) {
    std::cout << ans[i] << " \n"[i == n];
}

return 0;
}
```

例题：

D. A Wide, Wide Graph

Problem - D - Codeforces

换根dp求树直径.md

换根dp处理给定一个点， 到其它点的最大距离：

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

```cpp
using ll = long long;
const int N = 1E6 + 10;
vector<int> g[N];
int d[N];
int pa[N];
int sum[N];
//子树深度
void dfs(int u, int fa) {
    d[u] = 0;
    for (auto v : g[u]) {
        if (fa != v) {
            dfs(v, u);
            d[u] = max(d[v] + 1, d[u]);
        }
    }
}
//换根部dp;
void dfs2(int u, int fa) {

    int sz = g[u].size();
    vector<int>pre(sz + 5, -1), suf(sz + 5, -1);
    d[u] = max(d[u], pa[fa] + 1);
    for (int i = 1; i <= sz; i++) {
        int v = g[u][i - 1];
        if (fa != v) {
            pre[i] = max(pre[i - 1], d[g[u][i - 1]]);
        }
        else pre[i] = pre[i - 1];
    }
    for (int i = sz; i >= 1; i--) {
        int v = g[u][i - 1];
        if (fa != v) {
            suf[i] = max(suf[i + 1], d[g[u][i - 1]]);
        }
        else suf[i] = suf[i + 1];
    }
    for (int i = 1; i <= sz; i++) {
        int v = g[u][i - 1];
        if (fa != v) {
            pa[u] = max({ pre[i - 1] , suf[i + 1] , pa[fa]}) + 1;
            dfs2(g[u][i - 1], u);
        }
    }
}
int main()
{
```

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

    int n; cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1, 0);
    // for (int i = 1; i <= n; i++) {
    //  cout << d[i] << " \n"[i == n];
    // }
    pa[0] = -1;
    dfs2(1, 0);
    for (int i = 1; i <= n; i++) {
        sum[d[i] + 1]++;
    }
    // for (int i = 1; i <= n; i++) {
    //  cout << d[i] << " \n"[i == n];
    // }
    for (int i = 1; i <= n; i++) {
        sum[i] = sum[i - 1] + sum[i];
        cout << min(1 + sum[i], n) << " \n"[i == n];
    }
}
/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

树上两点最短距离

基础数据结构

树状数组

BIT

```
class BIT {
public:
    ll c[(int)1E6 + 10];
    ll query(int x) {
        ll res = 0;
        for (; x ; x -= x & (-x))
            res += c[x];
        return res;
    }
    void modify(int x, ll d) {
        assert(x != 0);
        for (; x <= n; x += x & (-x)) {
            c[x] += d;
        }
    }
    int bineray_serach(ll x) {
        int pos = 0;
        ll  t = 0;
        //18对应5e5
        //19对应1e6
        for (int i = 18; i >= 0; i--) {
            //t的水平一直是小于等于x的关系。
            if (pos + (1 << i) <= n && t + c[pos + (1 << i)] <= x) {
                pos += (1 << i);
                t += c[pos];
            }
        }
        //dbg(pos)
        return pos;
    }
};
//树状数组求区间和公式：
//cout << (x + 1)*d1.query(x) - d2.query(x) - (x)*d1.query(x - 1) + d2.query(x - 1) << '\n';
//区间修改仔细点，前加后减。小心记错结论。
//求和问题非常容易溢出。
```

单调队列

单调栈

ST表

st表 dls

```cpp
const int N = (int)1E6 + 10;
//把较小的维度放在前面有利于优化维度。
ll f[22][N] , a[N];
//int log[N];
void init() {
    //预处理log。
    //for(int i = 2; i <= n; i++) log[i] = log[i / 2] + 1;
    for (int i = 1; i <= n; i++) f[0][i] = a[i];
    for (int j = 1; j <= 20; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            f[j][i] = max(f[j - 1][i] , f[j - 1][i + (1 << (j - 1))]);
        }
    }
}
int query(int l , int r) {
    assert(l <= r);
    //len = log[r- l + 1];
    int len = 31 - __builtin_clz(r - l + 1);
    return max(f[len][l] , f[len][r - (1 << len) + 1]);
}
//这个下标默认从1开始。
```

tourist的st表

```cpp
/**
 *    author:  tourist
 *    created: 31.05.2022 18:35:43
**/
template <typename T, class F = function<T(const T&, const T&)>>
class SparseTable {
public:
    int n;
    vector<vector<T>> mat;
```

```
    F func;

    SparseTable(const vector<T>& a, const F& f) : func(f) {
        n = static_cast<int>(a.size());
        int max_log = 32 - __builtin_clz(n);
        mat.resize(max_log);
        mat[0] = a;
        for (int j = 1; j < max_log; j++) {
            mat[j].resize(n - (1 << j) + 1);
            for (int i = 0; i <= n - (1 << j); i++) {
                mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
            }
        }
    }

    T get(int from, int to) const {
        assert(0 <= from && from <= to && to <= n - 1);
        int lg = 32 - __builtin_clz(to - from + 1) - 1;
        return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
    }
};
//应用示例。挺好用的
/*int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    vector<ll> a(n + 1);
    SparseTable<long long> smin(a, [&](long long i, long long j) { return min(i, j);});
    SparseTable<long long> smax(a, [&](long long i, lon long j) { return max(i, j);});
}*/

```

线段树

http://oj.daimayuan.top/course/15/problem/654
除了一些实现bug。
代码写的很臭；
主要看几份代码

比较以及生长

1. 代码长度上，为什么能够做到两倍：
   因为自己没有写多了一些其它的懒惰标记维护。
   另外，关于递归后，处理两个儿子时。其实进行的是两个区间信息的合并操作。这个操作在build和modify，change 都存在。
   所以可以进行一个封装。加法封装。
   一般而言，线段树只是管理者一个区间。因此不太需要引入一个类。面向过程即可。

目标是理解这种封装角度，以及在这种高度封装的模板上完成迁移，信息维护利用等等。
数据结构 应该这样学。用这样一套东西，面对新问题的时候知道该删哪改哪。这样就理解了数据结构的成员，行为。

关于节点的定义：

线段树节点的定义。

```
struct info{
    int minva;
    int micnt;
};

struct node{
    info val;
    type lazy;//懒惰标记
};
```

当前问题dls的代码

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

const int N = 201000;

int n, q;
int a[N];

struct info {
    int minv, mincnt;
};

info operator + (const info &l, const info &r) {
    info a;
    a.minv = min(l.minv, r.minv);
    if (l.minv == r.minv) a.mincnt = l.mincnt + r.mincnt;
    else if (l.minv < r.minv) a.mincnt = l.mincnt;
    else a.mincnt = r.mincnt;
    return a;
```

```
21      }
22
23      struct node {
24          info val;
25      } seg[N * 4];
26
27      // [l, r]
28
29      void update(int id) {
30          seg[id].val = seg[id * 2].val + seg[id * 2 + 1].val;
31      }
32
33      void build(int id, int l, int r) {
34          if (l == r) {
35              seg[id].val = {a[l], 1};
36          } else {
37              int mid = (l + r) / 2;
38              build(id * 2, l, mid);
39              build(id * 2 + 1, mid + 1, r);
40              update(id);
41          }
42      }
43
44      // 节点为id，对应的区间为[l, r]，修改a[pos] -> val
45      void change(int id, int l, int r, int pos, int val) {
46          if (l == r) {
47              seg[id].val = {val, 1};
48          } else {
49              int mid = (l + r) / 2;
50              if (pos <= mid) change(id * 2, l, mid, pos, val);
51              else change(id * 2 + 1, mid + 1, r, pos, val);
52              // 重要!!
53              update(id);
54          }
55      }
56      // [ql, qr]表示查询的区间
57      info query(int id, int l, int r, int ql, int qr) {
58          if (l == ql && r == qr) return seg[id].val;
59          int mid = (l + r) / 2;
60          // [l, mid] , [mid + 1, r]
61          if (qr <= mid) return query(id * 2, l, mid, ql, qr);
62          else if (ql > mid) return query(id * 2 + 1, mid + 1, r, ql,qr);
63          else {
64              // qr > mid, ql <= mid
65              // [ql, mid], [mid + 1, qr]
66              return query(id * 2, l, mid, ql, mid) +
```

```
            query(id * 2 + 1, mid + 1, r, mid + 1, qr);
        }
    }

    int main() {
        scanf("%d%d", &n, &q);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &a[i]);
        }
        build(1, 1, n);
        for (int i = 0; i < q; i++) {
            int ty;
            scanf("%d", &ty);
            if (ty == 1) {
                int x, d;
                scanf("%d%d", &x, &d);
                change(1, 1, n, x, d);
            } else {
                int l, r;
                scanf("%d%d", &l, &r);
                auto ans = query(1, 1, n, l, r);
                printf("%d %d\n", ans.minv, ans.mincnt);
            }
        }
    }
```

典型求区间和的问题：dls code

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

const int N = 201000;

int n, q;
int a[N];

struct node {
    int minv;
} seg[N * 4];

```

```
14    // [l, r]
15
16    void update(int id) {
17        seg[id].minv = min(seg[id * 2].minv, seg[id * 2 + 1].minv);
18    }
19
20    void build(int id, int l, int r) {
21        if (l == r) {
22            seg[id].minv = a[l];
23        } else {
24            int mid = (l + r) / 2;
25            build(id * 2, l, mid);
26            build(id * 2 + 1, mid + 1, r);
27            update(id);
28        }
29    }
30
31    // 节点为id，对应的区间为[l, r]，修改a[pos] -> val
32    void change(int id, int l, int r, int pos, int val) {
33        if (l == r) {
34            seg[id].minv = val;
35        } else {
36            int mid = (l + r) / 2;
37            if (pos <= mid) change(id * 2, l, mid, pos, val);
38            else change(id * 2 + 1, mid + 1, r, pos, val);
39            // 重要!!
40            update(id);
41        }
42    }
43    // [ql, qr]表示查询的区间
44    int query(int id, int l, int r, int ql, int qr) {
45        if (l == ql && r == qr) return seg[id].minv;
46        int mid = (l + r) / 2;
47        // [l, mid] , [mid + 1, r]
48        if (qr <= mid) return query(id * 2, l, mid, ql, qr);
49        else if (ql > mid) return query(id * 2 + 1, mid + 1, r, ql,qr);
50        else {
51            // qr > mid, ql <= mid
52            // [ql, mid], [mid + 1, qr]
53            return min(query(id * 2, l, mid, ql, mid),
54                query(id * 2 + 1, mid + 1, r, mid + 1, qr));
55        }
56    }
57
58    int main() {
59        scanf("%d%d", &n, &q);
```

```
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }
    build(1, 1, n);
    for (int i = 0; i < q; i++) {
        int ty;
        scanf("%d", &ty);
        if (ty == 1) {
            int x, d;
            scanf("%d%d", &x, &d);
            change(1, 1, n, x, d);
        } else {
            int l, r;
            scanf("%d%d", &l, &r);
            printf("%d\n", query(1, 1, n, l, r));
        }
    }
}
```

我的臭代码

```
//维护区间中最小值出现的次数
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 2E5 + 10;


//信息不够紧凑。
int a[N];
//维护信息
int mi[N << 2]; //维护区间中的最小值。
int c[N << 2]; //维护最小值出现的次数。
//修改管理。

int lz[N << 2];//懒惰标记。


//传递子树的信息。
//收集子树的信息。

//建树函数里面主要完成几种功能。
```

```
23    //一直往下递归。
24    //返回子区间信息
25    //整理两个子区间信息。称为合并操作。
26    void build(int no , int l , int r)
27    {
28        if (l == r) {
29            c[no] = 1;
30            mi[no] = a[r];
31            return;
32        }
33        int mid = (l + r) >> 1;
34        build(no << 1, l, mid );
35        build(no << 1 | 1, mid + 1, r);
36        if (mi[no << 1] == mi[no << 1 | 1])
37        {
38            c[no] = c[no << 1] + c[no << 1 | 1];
39            mi[no] = mi[no << 1];
40        }
41        else if (mi[no << 1] < mi[no << 1 | 1])
42        {
43            c[no] = c[no << 1];
44            mi[no] = mi[no << 1];
45        }
46        else
47        {
48            c[no] = c[no << 1 | 1];
49            mi[no] = mi[no << 1 | 1];
50        }
51    }
52
53    void pd(int no, int l, int r) // 区间信息管理的节点编号。//左右区间。
54    {
55        int mid = (l + r) >> 1;
56        lz[no << 1] = lz[no << 1 | 1] = lz[no];
57        c[no << 1] = (mid - l + 1);
58        c[no << 1 | 1] = (r - mid);
59        mi[no << 1] = mi[no << 1 | 1] = lz[no];
60        lz[no] = 0;
61    }
62
63    void set_tag(int no , int l, int r, int k)
64    {
65        lz[no] = k;
66        mi[no] = k;
67        c[no] = r - l + 1;
68    }
```

```
69    void modify(int no, int l, int r, int ql, int qr, int k)
70    {
71        if (l >= ql && r <= qr)
72        {
73            set_tag(no, l, r, k);
74            return;
75        }
76        if (lz[no]) {pd(no, l, r);}
77        int mid = (l + r) >> 1;
78
79        if (l <= qr && mid >= ql)
80            modify(no << 1, l, mid, ql, qr, k);
81        if ( mid + 1 <= qr && r >= ql)
82            modify(no << 1 | 1, mid + 1, r, ql, qr, k);
83
84        //进行更新:
85        if (mi[no << 1] == mi[no << 1 | 1])
86        {
87            mi[no] = mi[no << 1];
88            c[no] = c[no << 1] + c[no << 1 | 1];
89        }
90        else if (mi[no << 1] < mi[no << 1 | 1])
91        {
92            mi[no] = mi[no << 1];
93            c[no] = c[no << 1];
94        }
95        else
96        {
97            mi[no] = mi[no << 1 | 1];
98            c[no] = c[no << 1 | 1];
99        }
100   }
101
102
103
104   //感觉这个查询不大方便。
105   //其实就是分成若干个块最终会分为若干个块。怎么处理这若干个块的信息呢?
106   //简单return一个int并不可以。因为合并的选择的时候要关注两个量。
107   //直接在全局里面定位两个?
108   struct node {
109       int mi;
110       int sum;
111   };
112   node query(int no  , int l, int r , int ql , int qr)
113   {
114       if (l >= ql && r <= qr)
```

```
115         {
116             return {mi[no], c[no]};
117         }
118         if (lz[no])
119             pd(no, l, r);
120         int mid = (l + r) >> 1;
121 
122         node res = {int(1E9 + 10), 0};
123 
124         if (l <= qr && mid >= ql)
125             res = query(no << 1, l, mid, ql, qr);
126         if (mid + 1 <= qr && r >= ql)
127         {
128             node temp = query (no << 1 | 1, mid + 1, r , ql , qr);
129             if (temp.mi == res.mi)
130                 res.sum += temp.sum;
131             else if (temp.mi < res.mi)
132                 res = temp;
133         }
134         return res;
135     }
136 
137 
138     int main()
139     {
140         ios::sync_with_stdio(false);
141         cin.tie(0);
142 
143         int n , m;
144         cin >> n >> m;
145         for (int i = 1; i <= n; i++)
146             cin >> a[i];
147 
148         build(1, 1, n);
149         for (int i = 1; i <= m; i++)
150         {
151             int ty, x, y;
152             cin >> ty >> x >> y;
153             if (ty == 1)
154             {
155                 modify(1, 1, n, x, x, y);
156             }
157             else {
158                 auto ans = query(1, 1, n, x, y);
159                 cout << ans.mi << ' ' << ans.sum  << '\n';
160             }
```

```
    }
}

/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

一个明智地追求快乐的人，除了培养生活赖以支撑的主要兴趣之外，总得设法培养其他许多闲情逸致。