StarsTAR codebook

初始模板

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

int main(){
    ios::sync_with_stdio(0); cin.tie(0);

    return 0;
}
```

規則:D

1. 五小時，ICPC賽制，題目數量 9-13 題
2. 允許語言：`C`、`C++17`、`JAVA`、`PYTHON3`
3. 比賽系統：domjudge
4. `COMPILER-ERROR` 會算 penalty (模擬台灣賽制)

5. 結束前一小時封版
6. 可以攜帶任何紙張資料(包括筆記、codebook等)，不限頁數 (講義、書不可攜帶)
7. 可以攜帶吉祥物、英漢字典、文具
8. 規則6、7兩點內東西，需於開始前交給工作人員檢查
9. 任何與題目相關的問題請使用 domjudge 提問，其餘關於上廁所、電腦等非題目相關問題舉手問工作人員即可
10. 賽中請勿打開domjudge以外的網站、並且只能使用一台電腦
11. 賽中提供影印功能，如需要影印code請舉手告知工作人員

U化

```
// 數字中可以加 ' 方便看出幾位數
#define MXN 1'000'005
 // 1e-6 為科學記號 代表 1 * 10^-6
#define EPS 1e-6
 // 0x3f3f3f3f為一個接近10^9的數字0x為16進位
#define INF 0x3f3f3f3f
 // acos(-1) 等同圓周率
#define PI acos(-1)
vector<int>::iterator iter = vec.begin();
// 上下兩者是一樣的意思
// 等於後面一定要接東西，才能判定型態
auto iter = vec.begin();
#define ll long long
#define ld long double
//range of
vector<int> vec{1,2,3};
for(int i:vec)    cout<<i;    //輸出為123
// 也可使用auto :D
for(auto i:vec)    cout<<i;

for(auto &i:vec)    i=i*10;  //修改值記得使用參照
// vec的元素變成 [10, 20, 30]
//位元運算
x <<= 1    //將x左移1，等同 *2
x >>= 2    //將x右移2，等同 /4
```

DP

設計DP的步驟: 設狀態⇒列轉移⇒算複雜度⇒考慮優化⇒實作出來.__.

最大子陣列和(最大區間和)

```
dp[i]=max(dp[i−1]+a[i],a[i])=max(dp[i−1],0)+a[i]
//dp[i] 代表以a[i]為結尾最大的區間和
```

最長遞增子序列(LIS)

```
//dp[i]代表的是以a[i]為結尾的最長遞增子序列長度
//dp[i]=max(dp[i],dp[j]+1),j<i∧a[j]<a[i]

void init(int *dp, int num) {for (int i = 0; i< num; i++) { dp[i] = 1; }}//初始化,選自己長度為1

for (int i = 0; i < num; i++) {
  for (int j = 0; j < i; j++) {
    if (a[j] < a[i]) {
      dp[i] = max(dp[i], dp[j] + 1);
    }
  }
}
```

最長共同子序列(LCS)

```
//字串s,t，求s,t的最長共同子序列
//邊界條件:dp[i][-1]=dp[-1][j]=0(從1開始,0設為邊界)
//dp[i][j]代表字串前綴s0...i與t0...j的LCS
//轉移式dp[i][j] = dp[i - 1][j - 1] + 1;  s[i] == t[j]
//     dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]); s[i] != t[j]

for (int i = 1; i <= num1; i++) {
  for (int j = 1; j <= num2; j++) {
    if (s[i] == t[j]) {
      dp[i][j] = dp[i - 1][j - 1] + 1;
    }
    else {
      dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
    }
  }
}
//若為字串s[i - 1] != '\0',s[i-1] == t[j-1]
```

矩陣鏈乘積

```
//令dp[l][r]代表Ml...Mr最少需要幾次乘法
//矩陣Mi的大小[i]×c[i]
//轉移式:dp[l][r]=min{dp[l][i]+dp[i+1][r]+r[l]*c[l]*c[r]}//(l≤i<r)
//邊界條件:dp[i][i]=0
```

(1/0)背包問題/無限背包問題

```
//(1/0)
struct MyStruct
{
  int w;int v;//重量 價值
}tt[505];
n為物品數,k為總重量
for (int i = 1; i <= n; i++) {
  for (int j = k; j !=0; j--) {
    if (j >= tt[i].w) {
      dp[j] = max(dp[j], dp[j - tt[i].w] + tt[i].v);
    }
    else {
      dp[j] = dp[j];
    }
  }
}
//無限
dp[i][j]=max(dp[i−1][j],dp[i][j−wi]+pi);//j≥wi
dp[i][j]=dp[i−1][j]; //j<wi
```

子集集和

```
bool subset[105][100005];//[n][sum]
    for (int l = 0; l <= n; l++) {
      subset[l][0] = 1;
    }
    for (int l = 1; l <= sum; l++) {
      subset[0][l] = 0;
    }
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= sum; j++) {
        subset[i][j] = subset[i-1][j];
        if (j >= num[i-1]) {//num[i-1]是因為num從0開始數
          subset[i][j] = subset[i][j] || subset[i -1][j- num[i - 1]];
        }
      }
    }
```

Levenshtein距離(最小更改次數使兩字串變為相同)

```
for (i = 1; s[i - 1] != '\0'; i++) {
    for (j = 1; t[j - 1] != '\0'; j++) {
      if (s[i - 1] == t[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1];
      }
      else {
        dp[i][j] = min(dp[i - 1][j - 1],min( dp[i][j - 1], dp[i - 1][j]))+1;
      }
    }
  }
```

jimmy

```
#include<iostream>
#include<string>
#include<algorithm>
#include<queue>
#include<cmath>
#include<math.h>

using namespace std;

struct rec {
    int l, r, h;
};

bool cmp(rec u,rec v) {
    return u.h < v.h;
}

int main() {
    int t;
    cin >> t;
    for (int i = 0; i < t; i++) {
        int N, X, Y, MAX;
        cin >> N >> X >> Y >> MAX;
        int ml = 20001, mr = -20001;
        rec a[1005] = {};
        for (int i = 1; i <= N; i++) {
            cin >> a[i].l >> a[i].r >> a[i].h;
        }

        a[N + 1].l = X; a[N + 1].r = X; a[N + 1].h = Y;

        a[0].l = -20001; a[0].r = 20001; a[0].h = 0;
```

```
        sort(a, a + N + 1, cmp);

        int dp[1005][2] = {0};

        dp[1][0] = a[1].h;
        dp[1][1] = a[1].h;

        for (int i = 1; i <= N+1; i++) {
            int lf = 0, rf = 0;
            for (int j = i-1; j > 0; j--) {
                if (a[i].r <= a[j].r&&a[i].h - a[j].h<=MAX&&!rf&&a[i].r >= a[j].l) {
                    dp[i][1] = min(dp[j][0] + a[i].r - a[j].l + a[i].h - a[j].h,
                    dp[j][1] + a[j].r - a[i].r + a[i].h - a[j].h);
                    rf++;
                }
                if (a[i].l >= a[j].l&&a[i].h - a[j].h <= MAX&&!lf&&a[i].l <= a[j].r) {
                    dp[i][0] = min(dp[j][0] + a[i].l - a[j].l + a[i].h - a[j].h,
                    dp[j][1] + a[j].r - a[i].l + a[i].h - a[j].h);
                    lf++;
                }
            }
            if (!rf) {
                if(a[i].h <= MAX)dp[i][1] = a[i].h;
                else dp[i][1] = 20001;
            }
            if (!lf) {
                if (a[i].h <= MAX)dp[i][0] = a[i].h;
                else dp[i][0] = 20001;
            }
        }

        int ans = 20001;
        ans = min(dp[N+1][0], dp[N+1][1]);
        cout << ans << endl;
    }
}
```

greedy

合併區間

```
struct MyStruct{
  int r, l;
}num[50005];

bool cmp(MyStruct s1, MyStruct s2) {
  return s1.l < s2.l;
}
```

```
int main() {
  int task;
  cin >> task;
  for (int i = 0; i < task; i++) {
    cin >> num[i].l >> num[i].r;
  }
  sort(num, num + task, cmp);
  int ll = num[0].l;
  int rr = num[0].r;
  for (int i = 0; i < task; i++) {
    if (num[i].l <= rr && num[i].l >= ll) {
      if (num[i].r > rr) {
        rr = num[i].r;
      }
    }
    else {
      cout << ll << ' ' << rr << endl;
      ll = num[i].l;
      rr = num[i].r;
    }
  }
  cout << ll << ' ' << rr << endl;
  return 0;
}
```

分治法(D&C)

分(divide):將大問題分成小問題
治(conquer):遞迴計算出小問題的答案
合(merge):將小問題的答案合併成大問題的答案

quick sort

```
void qqsort(int l, int r) {

    if (l < r) {

        int beg = l;
        int end = r + 1;
        int pivot = a[l];

        while (1) {

            while (beg < r&&pivot > a[++beg]);
            while (end > 0 && pivot < a[--end]);
```

```
            if (beg >= end)break;

            swap(a[beg], a[end]);
        }
            swap(a[l], a[end]);

            qqsort(l, end - 1);
            qqsort(end+1, r);


    }
}


int main(){

    qqsort(0, 4);
    for (int i = 0; i < 5; i++)cout << a[i] << ' ';

}
```

merge sort

將長度為n的序列排序

```
void mergesort(vector <int> array[],int start,int end)
{
  while(start<end)
  {
    int mid = (start+end)/2;
    mergesort(array[],start,mid);
    mergesort(array[],mid+1,end);
    merge(array[],start,mid,end);
  }

}
#define Max 無限大
void merge(vector <int> array[],int start,int mid,int end)
{
  vector <int> leftsub(array.begin()+front,array.begin()+mid+1)
  vector <int> rightsub(array.begin()+mid+1,array.begin()+end+1)
  leftsub.insert(leftsub.end(), Max);
  rightsub.insert(rightsub.end(), Max);
  int idxLeft = 0, idxRight = 0;
  for (int i = front; i <= end; i++)
  {
    if (leftsub[idxLeft] <= rightsub[idxRight] )
    {
      array[i] = leftsub[idxLeft];
      idxLeft++;
```

```
    }
    else
    {
       array[i] = rightsub[idxRight];
       idxRight++;
    }
  }
}
```

Binary Search

條件:窮舉的對象與判斷條件具有單調性(遞增/遞減關係)

細節

- **確定有單調性**
- **決定範圍**
- **往哪個方向更新**
- **中點要選進位還是捨去的值**
- **結束條件**

找出x使得f(x)=c

```
mid = (low+high)/2
if(c==f(mid)) x = mid;
else if(c<f(mid)) high = mid-1;
else low = mid+1;
```

找出最小x使符合條件

```
mid = (low+high)/2
if(c<=f(mid)) high = mid;
else if(c>f(mid)) low = mid+1;
```

```
mid = (low+high+1)/2
if(c<=f(mid)) low = mid;
else if(c>f(mid)) high = mid-1;
```

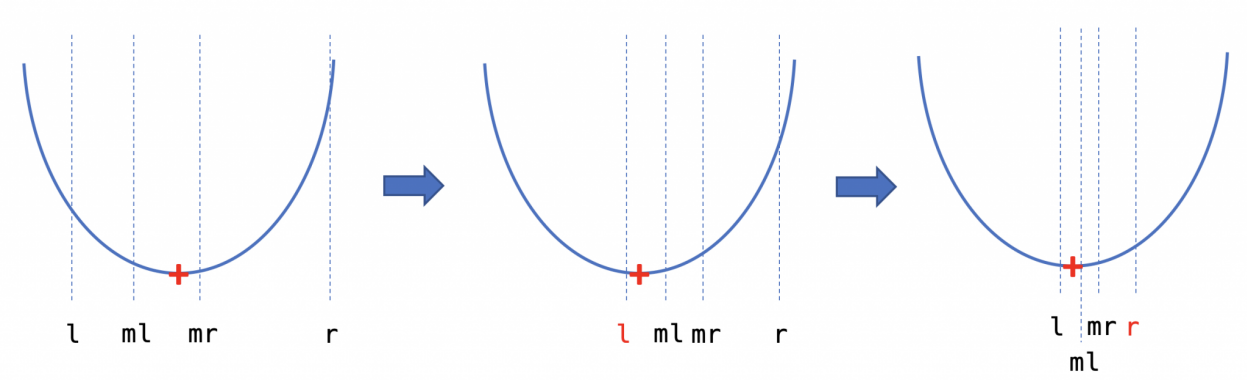
浮點數二分搜

更新範圍:改成+誤差值/-誤差值

結束:上界-下界<誤差值

Ternary Search(三分搜)

函數可找到斜率:用二分搜




```
//l:下界 r:上界
//無限逼近極值
//剩幾個點時可枚舉
while(！停止條件)
{
  double mr = (l+r)/2;
  double ml = (l+mr)/2;
  //f():函式
  if(f(ml)>=f(mr))
  {
    l = ml;
  }
  else
  {
    r = mr;
  }

}
```

---

graph

定義

圖(graph)：由一些點與一些邊所組成，通常以G=(V,E)表示
點(vertex)：節點，通常以V表示
邊(edge)：連接兩點，通常以E表示，e=(u,v)代表邊e連接u,v兩點，也就是u,v為e的端點
權重(weight)：指圖的點/邊附帶的數值
點數/邊數：點/邊的數量，通常記為V/E(n/m)
有向邊：邊可以分為無向邊(雙向)與有向邊(單向)
重邊(multiple edge)：指兩點之間有多條邊連接
自環(self loop)：指兩端為同一點的邊e=(u,u)
度數(degree)：一個點所連接的邊的數量，若是有向邊則又分為出度與入度
相鄰(adjacent)：指兩個點間有無向邊相連
指向(consecutive)：有向邊的起點"指向"終點
路徑(path)：不會描述，路徑上的點/邊皆可重複
行跡(trace)：邊不重複的路徑
迴路(circuit)：邊不重複且起終點相同的路徑
簡單路徑(track)：點不重複的路徑
環(cycle)：點不重複且起終點相同的路徑
連通(connected)：u,v連通若且唯若存在u到v或v到u的路徑，一群點連通代表這群點兩兩連通
簡單圖(simple graph)：沒有重邊與自環的圖
無向圖(undirected graph)/有向圖(directed graph)：只有無向邊/有向邊的圖
連通圖(connected graph)：任兩點皆連通的圖
樹(tree)：無向無環連通圖(其實也可以有向)
森林(forest)：很多棵樹的圖(無向無環圖)
完全圖(complete graph)：任兩點都相鄰的圖
二分圖(bipartite graph)：圖上的點可以分為兩群U,V使得同群點之間皆不相鄰
子圖(subgraph)：邊/點皆為原圖的子集
補圖(complement graph)：若兩張圖點集相同，邊集互斥且聯集為完全圖，則兩張圖互為補圖
同構(isomorphic)：不考慮編號長的完全相同的圖
生成樹(spanning tree)：點集相同又是樹的子圖

鄰接串列

```
vector<int> node[205];
for(int i = 0; i < n; i++) {
```

```
    cin >> u >> v;
    node[u].push_back(v);
    node[v].push_back(u);
}
```

DFS

```
void dfs(int x){
    vis[x]=1;
    for(int i:arr[x]){//int i = 0; i < node[x].size(); i++
        if(!vis[i])//i = node[x][i];
            dfs(i);
    }
}
```

```
void dfs(int x) {
  vis[x] = 1;
  for (int i = 0; i < node[x].size(); i++) {
    if (!vis[node[x][i]]) {
      dfs(node[x][i]);
    }
  }
}
```

BFS

```
void bfs(int s){
    queue<int> q;
    q.push(s);
    vis[s]=1;
    while(!q.empty()){
        int x=q.front();
        q.pop();
        for(int i:arr[x]){
            if(!vis[i])
                q.push(i),vis[i]=1;
        }
    }
}
```

```
void bfs(int s) {
  queue<int> q;
```

```
  q.push(s);
  vis[s] = s;
  while (!q.empty()) {
    int x = q.front();
    q.pop();
    for (int i = 0; i < node[x].size(); i++) {
      if (!vis[node[x][i]]) {
        q.push(node[x][i]), vis[node[x][i]] = 1;
      }
    }
  }
}
```

---

tree

定義

根(root)：任一點皆可作為根，有指定根的樹稱作有根樹，通常會以根將樹吊起來，樹是往下長的
葉(leaf)：度數為1的點稱作葉(有根樹的根例外)
子樹(subtree)：移除一個點/邊後產生的樹為子樹
父子(parent,child)：有根樹中，對於兩相鄰點，靠近根者為父，遠離根者為子
祖先/後代(ancestor,descendant)：多層父子關係
距離(distance)：兩點間邊的數量(或邊權總合)
深度(depth)：有根樹中一個點到根的距離
高度(height)：有根樹中一個點到最遠的葉的距離

DFS(tree)

```
void dfs(int x,int p)//p為父節點
{
    for(int i:arr[x])
    {
        if(i!=p)
            dfs(i,x);
    }
}
```

求深度

```
void dfs(int x,int p)
{
    for(int i:arr[x])
    {
        if(i!=p)
            d[i]=d[x]+1;
            dfs(i,x);
    }
}
```

求高度

```
void dfs(int x,int p)
{
    h[x]=0;
    for(int i:arr[x])
    {
        if(i!=p)
            dfs(i,x);
            h[x]=max(h[x],h[i]+1);
    }
}
```

樹直徑

分治

```
int ans;
int dfs(int x,int p)
{
    int mh1=0,mh2=0,cur;
    for(auto i:arr[x])
    {
        if(i!=p)
        {
            cur=dfs(i,x);
            if(cur>mh1)
                mh2=mh1,mh1=cur;
            else if(cur>mh2)
                mh2=cur;
        }
    }
    ans=max(ans,mh1+mh2);
    return mh1+1;
}
```

先找最深再從最深找最遠

```
int ans;
int farnode;
int u, v, l;
struct edge
{
    int to, len;
};
vector<edge> node[MXN];
while (getline(cin, s)) {
            if (s == "") { break; }
            stringstream str(s);
            str >> u >> v >> l;
            tmp.len = l;
            tmp.to = v;
            node[u].push_back(tmp);
            tmp.to = u;
            node[v].push_back(tmp);
}

void dfs(int x, int p, int d) {
    if (d > ans) {
        ans = d;
        farnode = x;
    }
    for (int i = 0; i < node[x].size(); i++)
    {
        int temp = node[x][i].to;
        if (temp == p)
            continue;
        dfs(temp, x, d + node[x][i].len);
    }
}
```

最短路

Dijkstra單源最短路徑

w[a][b] 為點 a 到點 b 的邊權重
將所有點到起點的距離設為無限大 (用 dis 陣列紀錄)
起點自己本身的距離設為 0
重複以下步驟，直到全部點都走過為止
選從還沒走過的點中，離起點最近的點

將此點設定為走過
窮舉此點所有連到的點，進行鬆弛

dijkstra 演算法只能跑邊權重≥0的圖(負環88)

鬆弛

```
void relaxation(int u, int v, int w){
    if(dis[u] > dis[v] + w){
        dis[u] = dis[v] + w;
    }
}
```

初始化

```
#define INF 1e18
for(int i=0;i<n;i++){
    if(i == start)
        dis[i] = 0;
    else
        dis[i] = INF;
}
```

主code

```
while(1){
    int ind = -1, mx = INF;
    for(int i = 0; i < n; i++){ // 1. 選從還沒走過的點中，離起點最近的點
        if(!vis[i] && dis[i] < mx){
            ind = i, mx  = dis[i];
        }
    }
    if(ind == -1)  // 如果全部點都走過則結束演算法
        break;
    vis[ind] = 1; // 2. 將此點設定為走過
    for(int i = 0; i < edge[ind].size(); i++){  // 3. 窮舉此點所有連到的點，進行鬆弛
        int u = edge[ind].to, w = edge[ind].weight;
        if(dis[u] > dis[ind] + w){
            dis[u] = dis[ind] + w;
        }
    }
}
```

用pq U化:D

```
init();
priority_queue<pair<ll,int>,vector<pair<ll,int>>,greater<pair<ll,int>>> pq;

pq.push(make_pair(dis[start], start)); // 為了方便實作，用pair包起來會先比較距離大小

while(!pq.empty()){
    auto [d, u] = pq.top(); pq.pop();
    if(vis[u])    continue;  // 確保每個點最多只被走過一遍
    vis[u] = 1;
    for(int i = 0; i < edge[u].size(); i++){  // 窮舉此點所有連到的點
        int v = edge[u][i].to, w = edge[u][i].weight;
        if(dis[v] > dis[u] + w){
            dis[v] = dis[u] + w;  // 鬆弛
            pq.push(make_pair(dis[v], v)); // 如果有更新距離，則丟進 priority_queue
        }
    }
}
```

Bellman-ford單源最短路徑

鬆弛

```
for(int j = 0; j < n-1; j++){// n個點 共有n-1條邊
    for(int i = 0; i < m; i++){ // 對於所有邊都嘗試鬆弛
        if(dis[ edge[i].to ] > dis[ edge[i].from ] + edge[i].weight){
            dis[ edge[i].to ] = dis[ edge[i].from ] + edge[i].weight;
        }
    }
}
```

偵測負環

```
//假設我們要找一張圖有沒有負環，則可以使用 Bellman-ford在沒有負環的情況下
//我們最多只需要跑 n−1次後，則找不到點去鬆弛
//因此跑完 n−1次之後，我們再多跑一次，只要有點可以鬆弛，則代表此圖有負環
for(int j = 0; j < n-1; j++){// n個點 共有n-1條邊
    for(int i = 0; i < m; i樓上的U化版本:D
```

SPFA(Shortest Path Faster Algorithm)樓上的U化版本:D

偵測負環

```
//設一個 len 陣列，紀錄每個點從起點開始是第幾輪被更新到超過 N-1輪還可以更新，則代表有負環的出現
int len[N]; // 紀錄每個點是第幾輪被鬆弛到
bool inque[N];// 紀錄是否已經在 queue 裡面
queue<int> que;
que.push(start);
while(!que.empty()){
    int u = que.front(); que.pop();
    if(len[u] > n-1)    return -1; // 超過 n-1 輪，找到負環
    inque[u] = 0;// 從 queue 拿出來設成 0
    for(int i = 0; i < edge[u].size(); i++){
        int v = edge[u][i].to, w = edge[u][i].weight;
        if(!inque[v] && dis[v] > dis[u] + w){
            dis[v] = dis[u] + w;
            que.push(v);
            inque[v] = 1;// 放進 queue 裡面設成 1
            len[v] = len[u] + 1; // 從來的點 +1輪被鬆弛到
        }
    }
}
```

deque版本U化

```
deque<int> deq; // 改成 deque
deq.push_back(start);
while(!que.empty()){
    // 取頭尾離起點距離較小的點
    int u = (dis[deq.front()] < dis[deq.back()] ? deq.front():deq.back());
    dis[deq.front()] < dis[deq.back()] ? deq.pop_front():deq.pop_back();
    if(len[u] > n-1)    return -1;
    inque[u] = 0;
    for(int i = 0; i < edge[u].size(); i++){
        int v = edge[u][i].to, w = edge[u][i].weight;
        if(!inque[v] && dis[v] > dis[u] + w){
            dis[v] = dis[u] + w;
            que.push(v);
            inque[v] = 1;
            len[v] = len[u] + 1;
        }
    }
}
```

Floyd Warshall全點對最短路徑

初始化

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        if(i != j)
            dis[i][j] = INF;
        else
            dis[i][j] = 0;
    }
}
cin>> u >> v >> w;
dis[u][v] = min(dis[u][v], w);
```

主code

```
int dis[N][N];
for(int k = 0; k < n; k++){ // 窮舉中繼點 k
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){ // 窮舉點對 (i, j)
            dis[i][j] = min(dis[i][j], dis[i][k]+dis[k][j]);
        }
    }
}
```

Data Structure

線段樹segment tree

build

```
#define cl(x) (x*2)    //左子節點index
#define cr(x) (x*2+1)  //右子節點index
void build(int i,int l,int r){ //i為當前節點index, l,r為當前遞迴區間
    if(l == r){ // 遞迴到區間大小為1
        tree[i] = arr[l];
        return;
    }
    int mid=(l+r)/2; //往兩邊遞迴
    build(cl(i),l,mid);
    build(cr(i),mid+1,r);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);
    //將節點的值設成左右子節點的最大值
}
```

詢問

```
// i 為當前節點index, l, r當前區間左右界，ql, qr詢問左右界
int query(int i,int l,int r,int ql,int qr){
    if(ql <= l && r <= qr){ //若當前區間在詢問區間內，直接回傳區間最大值
        return tree[i];
    }
    int mid=(l+r)/2, ret=0;
    if(ql<=mid) // 如果左子區間在詢問區間內
        ret = max(ret, query(cl(i),l,mid,ql,qr));
    if(qr> mid) // 如果右子區間在詢問區間內
        ret = max(ret, query(cr(i),mid+1,r,ql,qr);
    return ret;
}
```

單節點更新

```
void update(int i,int l,int r,int pos,int val){
    if(l == r){ // 修改 a[pos] 的值為 val
        tree[i] = val;
        return;
    }
    int mid=(l+r)/2;
    if(pos <= mid) // 如果修改位置在左子節點，往左遞迴
        update(cl(i),l,mid,pos,val);
    else // 否則往右遞迴
        update(cr(i),mid+1,r,pos,val);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);
}
```

區間修改

```
void update(int i,int l,int r,int ql,int qr,int v){//將區間l,r都加上v
    push(i,l,r);
    if(ql<=l && r<=qr){
        tag[i] += v;
        return;
    }
    int mid=(l+r)/2;
    if(ql<=mid)  update(cl(i),l,mid,ql,qr,v);
    if(qr> mid)  update(cr(i),mid+1,r,ql,qr,v);
    pull(i,l,r);
}
```

push

```
#define NO_TAG 0
void push(int i,int l,int r){
    if(tag[i] != NO_TAG){ // 判斷是否有打標記
        tree[i] += tag[i];  // 有的話就更新當前節點的值
        if(l != r){  // 如果有左右子節點把標記往下打
            tag[cl(i)] += tag[i];
            tag[cr(i)] += tag[i];
        }
        tag[i] = NO_TAG; // 更新後把標記消掉
    }
}
```

pull

```
void pull(int i,int l,int r){
    int mid = (l+r)/2;
    push(cl(i),l,mid); push(cr(i),mid+1,r);
    tree[i] = max(tree[cl(i)], tree[cr(i)]);
}
```

區間查詢，單點修改，線段樹

```
#include<iostream>
#include<vector>
#include<algorithm>
#define cl(idx) (2*idx)
#define cr(idx) (2*idx+1)

using namespace std;

int a[100005] = {};
int tree[400005] = {};

void build(int idx, int l, int r) {
    if (l == r) { tree[idx] = a[l]; return; }

    int mid = (l + r) / 2;
    build(cl(idx), l, mid);
    build(cr(idx), mid + 1, r);
    if (tree[cl(idx)] * tree[cr(idx)] > 0) tree[idx] = 1;
    else if (tree[cl(idx)] * tree[cr(idx)] < 0) tree[idx] = -1;
    else tree[idx] = 0;
}

int query(int idx, int l, int r, int ql, int qr) {
    if (ql <= l && qr >= r)return tree[idx];
```

```
    int mid = (l + r) / 2;
    int ret = 1;

    if (ql <= mid) { ret = query(cl(idx), l, mid, ql, qr)*ret; }
    if (qr > mid) { ret = query(cr(idx), mid + 1, r, ql, qr)*ret; }

    return ret;
}

void update(int idx, int l, int r, int pos, int val) {
    if (l == r) {tree[idx] =  val; return; }

    int mid = (l + r) / 2;

    if (mid >= pos) { update(cl(idx), l, mid, pos, val); }
    if (mid < pos) { update(cr(idx), mid+1, r, pos, val); }

    if (tree[cl(idx)] * tree[cr(idx)] > 0) tree[idx] = 1;
    else if (tree[cl(idx)] * tree[cr(idx)] < 0) tree[idx] = -1;
    else tree[idx] = 0;

}


int main() {
    int n, k;
    while (cin >> n >> k) {

        for (int i = 1; i <= n; i++) {
            cin >> a[i];
        }

        build(1, 1, n);

        for (int i = 1; i <= k; i++) {
            char c;
            cin >> c;
            if (c == 'C') {
                int pos, val;
                cin >> pos >> val;
                update(1, 1, n, pos, val);
            }
            if (c == 'P') {
                int ql, qr;
                cin >> ql >> qr;
                int ans = query(1, 1, n, ql, qr);
                if (ans > 0)cout << '+';
                else if (ans < 0)cout << '-';
                else cout << 0;
            }
        }

        cout << endl;
```

```
    }


}
```

樹狀數組Binary Indexed Tree

lowbit(x),**求出正整數 x 換成二進位最右邊的 1 的值**

```
#define lowbit(x) x&-x
```

詢問

```
int query(int x){
    int ret=0;
    while(x){ // 當不為 0 時
        ret += BIT[x];  // 回傳值加上BIT[x]
        x -= lowbit(x); // 每次減掉自己的lowbit
    }
    return ret;
}
```

離散化 樹狀數組

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

int a[500005] = {};
int bit[500005] = {};
int n;
vector<int>v;

int lowbit(int x) {
    return x & (-x);
}

long long query(int x) {
    long long ret = 0;
    while (x) {
        ret += bit[x];
        x -= lowbit(x);
    }
```

```
    return ret;
}

void update(int x, int v) {
    while (x <= n) {
        bit[x] += v;
        x += lowbit(x);
    }
}

int main() {
    while (cin >> n && n) {
        long long ans = 0;
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
            v.push_back(a[i]);
            bit[i] = 0;
        }

        sort(v.begin(), v.end());
        v.resize(unique(v.begin(), v.end())-v.begin());
        for (int i = 1; i <= n; i++) {
            a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;
        }


        for (int i = n; i >= 1; i--) {
            ans += query(a[i] - 1);
            update(a[i], 1);
        }

        cout << ans << endl;

        v.clear();

    }
```

快速冪

快速冪

計算x^y mod (p)

```
long long mypow(long long x,long long y){
    long long ans = 1;
    while(y){
        if( y&1 )    ans = ans * x % p;
        x = x * x % p;    //每次把自己平方
```

```
        y >>= 1;    //每次右移一格
    }
    return ans;
}
```

矩陣快速冪

```
int base[2][2]={
    {1, 1},
    {1, 0}
};
int ans[2][2]={
    {1, 0},
    {0, 1}
};

int mypow(int y){
    while(y){
        if( y&1 )    ans = mul(ans, base);//實作矩陣乘法
        base = mul(base, base);//實作矩陣乘法
        y >>= 1;
    }
    return ans[0][0];
}

//直行橫列
  for (int i = 0; i < a; i++) {
    for (int j = 0; j < d; j++) {
      for (int k = 0; k < b; k++) {
        ans[i][j] = ans[i][j] + array1[i][k] * array2[k][j];
      }
  }
}
```

生成樹

最小生成樹

```
#include<stdio.h>
int f[200005] = { 0 };
void init(int n) {
  for (int i = 0; i < n; i++)
    f[i] = i;
}
int find(int x) {
```

```
  if (f[x] == x)    return x;
  f[x] = find(f[x]);
  return f[x];
}
void marge(int a, int b) {
  a = find(a);
  b = find(b);

  if (a != b) {
    f[b] = a;
  }
}

struct MyStruct
{
  int u, v;
  long long int w;
}www[200005];

int cmp(const void* a, const void* b)
{
  struct MyStruct* c = (struct MyStruct*)a;
  struct MyStruct* d = (struct MyStruct*)b;
  if ((c->w - d->w) > 0) {
    return 1;
  }
  else if ((c->w - d->w) == 0) {
    return 0;
  }
  else {
    return -1;
  }
}

int main() {
  int n, m;
  while (scanf("%d %d", &n, &m)!=EOF) {
    init(n);
    for (int i = 0; i < m; i++) {
      scanf(" %d %d %lld", &www[i].u, &www[i].v, &www[i].w);
    }
    int count = 0;
    long long int ans = 0;
    qsort(www, m, sizeof(www[0]), cmp);
    for (int i = 0; i < m; i++) {
      int f1 = find(www[i].u);
      int f2 = find(www[i].v);
      if (f1 != f2) {
        marge(f1,f2);
        ans = ans + www[i].w;
        count++;
      }
      if (count == (n - 1)) {
```

```
        printf("%lld\n", ans);
        break;
      }
    }
    if (count != (n - 1)) {
      printf("-1\n");
    }
  }
  return 0;
}
```

質數判斷

```
bool isprime[1000005];
int primeSize = 0, prime[200005];
//先將所有大於1數字設為質數
for(int i=2;i<=n;i++)    isprime[i] = 1;

for(int i=2;i<=n;i++){
    if(isprime[i]){    //如果為質數
        prime[primeSize++] = i;
        for(long long j=i;i*j<=n;j++){
            //所有質數的倍數設成非質數
            isprime[i*j] = 0;
        }
    }
}
```

string

sstream(字串分割)

```
string s;
getline(cin,s)
stringstream str(s);
str >> u >> v >> l;
str.fail() == 1 //表示已都切割並輸出完畢
```

STL

vector/deque

宣告

```
#include<vector>//函式庫
vector<資料型態>num(大小);//名稱為num的vector容器
```

begin/end

```
for (vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it){
  cout << ' ' << *it;cout << '\n';
  return 0;
}
```

rbegin/rend

```
vector<int>::reverse_iterator rit;
for (rit = myvector.rbegin(); rit!= myvector.rend(); ++rit)*rit = ++i;
```

iterator(迭代器)/reverse_iterator(反向迭代器)

```
宣告方法 vector<int>::iterator it
取值用 *it
宣告發法 vector<int>::reverse_iterator rit;
取值用 *rit
```

size/resize/reserve

num.size() 表當前容器的大小

```
vector<int> myints;
cout << myints.size() << '\n';
myints.insert (myints.end(),10,100);//從 .end()開始 插入10次100
```

resize:重新給予新的size大小

```
myvector.resize(5);//size變為五
myvector.resize(8,100);//size變為8 剩下補100
```

```
myvector.resize(12);
myvector contains: 1 2 3 4 5 100 100 100 0 0 0 0
```

reserve:

```
a.reserve(65);  // 預留 65 個元素的位置，沒有初始化
```

empty 判斷size是不是0 回傳t/f

front/back/data

```
//回傳數值
int i = myvector.front();
int i = myvector.back();
//回傳指標
int* p = myvector.data();
```

push_back/pop_back/insert

```
num.push_back(數值)將數值加入容器最後端
pop移除同理
num.insert (num.end(),10,100);//從 .end()開始 插入10次100
it = myvector.begin();
it = myvector.insert ( it , 200 );從位置it 插入200
myvector.insert (it,2,300);從位置it 插入2次300
vector<int> anothervector (2,400);
myvector.insert (it+2,anothervector.begin(),anothervector.end());
int myarray [] = { 501,502,503 };
myvector.insert (myvector.begin(), myarray, myarray+3);
```

erase

```
// erase the 6th element
  myvector.erase (myvector.begin()+5);
```

swap

```
vector<int> foo (3,100);   // three ints with a value of 100
vector<int> bar (5,200);   // five ints with a value of 200
```

```
foo.swap(bar);//foo和bar整個交換
```

clear

```
num.clear();
size清零
```

deque

#include <deque>

多了**push_front和pop_front**

list

和deque一樣，多了**splice**

```
list1.splice(it,list2);//從位置it插入list2 list2 清空
```

pq

priority_queue<T> pq; 預設由大排到小

priority_queue<T, **vector**<T>, **greater**<T> > pq; 改成由小排到大

priority_queue<T, **vector**<T>, **cmp**> pq; 自行定義 cmp 排序

pq.**push**(x); 加入元素

x = pq.**top**();讀取優先權最高的值

pq.pop(); // 讀取後刪除

---

bs ACcode

给你N堆电池，总时间K，每堆电池有三个属性，电池的数量，还有一个bi、ci值，从第i堆拿一个电池放到第j堆，需要花费(bi+cj)时间，让你在K时间内，对电池进行移动，输出的是电池数量最多的那一堆的电池数目（最小值）。

```
#include <iostream>
#include <stdio.h>
```

```
#include <math.h>
#include <string.h>
#include <string>
#include <stdlib.h>
#include <algorithm>
#include <queue>

#define ll long long
#define P pair<int,int>
using namespace std;

const int maxn=100000+5;
int N;
ll k;

struct Point
{
   ll a,b,c;
   bool operator <(const Point &rhs)const
   {
       return c <rhs.c;
   }
}p[maxn];

int ok(int h)
{
    ll tot=0,sum=0;//tot: 大于平均值的电池数  sum:总花费
    for(int i=0; i<N; i++)
    {
        if(p[i].a<h)
            continue;
        tot+=p[i].a-h;
        sum+=(p[i].a-h)*p[i].b;
    }

    for(int i=0; i<N&&sum<=k&&tot; i++)
    {
        if(p[i].a>=h)
            continue;
        ll num=min(tot,h-p[i].a);
        tot-=num;
        sum+=num*p[i].c;
    }
    if(sum<=k&&!tot)
        return 1;
    return 0;
}
int main()
{
    while(scanf("%d %lld",&N, &k)!=EOF)
    {
        ll l=0,r=0,mid,ans;
        for(int i=0; i<N; i++)
```

```
          {
              scanf("%lld %lld %lld",&p[i].a,&p[i].b,&p[i].c);
              r=max(r,p[i].a);
              l+=p[i].a;
          }
        sort(p,p+N);
      l=(l+N-1)/N; //向上取整
      while(l<=r)
      {
          mid=(l+r)/2;
          if(ok(mid))
           r=mid-1,ans=mid;
          else
            l=mid+1;
      }
      printf("%lld\n",ans);
    }
    return 0;
}
```

糖果

```
#include <bits/stdc++.h>
using namespace std;
int bsearch(int a[],int n,int k)
{
    int lo;
    int hi;
    int mid;
    int i;
    long long int count;
    lo = 0;
    hi = a[n-1];

    while(lo < hi) {
        mid = lo + (hi-lo+1)/2;
        count = 0;
        for(i = 0; i < n; i++) {
            count = count + (a[i]/mid);
        }

        if(count >= k) {
            lo = mid;
        }
        else {
            hi = mid-1;
        }
    }
    return lo;
}
int main()
```

```
{
    long long int k;
    int n;
    int t;
    int a[50005];
    int i;
    int ans;
    scanf("%d",&t);
    while(t--) {
        scanf("%d%lld",&n,&k);
        for(i = 0; i < n; i++) {
            scanf("%d",&a[i]);
        }
        sort(a,a+n);
        ans = bsearch(a,n,k);
        printf("%d\n",ans);
    }
}
```