

ACM 常用算法模板



1. 头文件模板 3

2. KMP改进算法 3

3. Lucas算法（计算C(n,m)%mod） 4

4. A^B快速幂（计算A^B%C的结果） 5

5. GCD(最小公约数)与LCM（最小公倍数） 6

6. 二分图匹配 6

7. 求(n!%p的值) 7

8. RMQ区间查询 7

9. 素数区间筛选 8

10. 并查集 9

11. 最长回文子串 10

12. zkw费用流 11

13. 最长上升子序列 12

14. 拓扑排序 13

15. 线段树维护 14

16. Tarjan强连通图分量 16

17. Dij算法 18

18. Dinic算法 20

19. 全排列 21

20. 二分搜索 21

21. 背包问题 22

22. LCS（最长公共子序列） 23

23. 向量用法 24

24. 求多边形面积 25

25. 区间更新 25

26. 树状数组 27

# 头文件模板

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <algorithm>

#include <string>

#include <vector>

#include <cmath>

#include <queue>

using namespace std;

typedef long long ll;

const int INF = 1e9;

const int maxn = 1e6;

const int mod = (1e9+7);

# KMP改进算法

void makeLPS(int \*lps , int m , string pattern)

{

lps[0] = 0 ;

int len = 0 ;

int  i = 1;

while(i < m)

{

if(pattern[i] == pattern[len])

{

len ++ ;

lps[i] = len ;

i ++ ;

}

else

{

if(len != 0)

{

len = lps[len - 1];

}

else

{

lps[i] = 0;

i++;

}

}

}

}

string data ;

int kmpSearch(string pattern)

{

int n = data.size() ;

int m = pattern.size() ;

int lps[m] ;

makeLPS(lps , pattern.size() , pattern) ;

int i = 0 , j = 0 , ans = 0 ;

while(i < n)

{

if(pattern[j] == data[i])

{

i++;

j++;

}

if(j == m)

{

ans++;

j = lps[j - 1];

}

else if(i < n  && data[i] != pattern[j])

{

if(j != 0)

{

j = lps[j - 1];

}

else

{

i++;

}

}

}

return ans;

}

# Lucas算法（计算C(n,m)%mod）

ll ans;

ll quick\_mod(ll a, ll b)

{

ll ans = 1;

a %= mod;

while(b)

{

if(b & 1)

{

ans = ans \* a % mod;

b--;

}

b >>= 1;

a = a \* a % mod;

}

return ans;

}

ll C(ll n, ll m)

{

if(m > n) return 0;

ll ans = 1;

for(int i=1; i<=m; i++)

{

ll a = (n + i - m) % mod;

ll b = i % mod;

ans = ans \* (a \* quick\_mod(b, mod-2) % mod) % mod;

}

return ans;

}

ll Lucas(ll n, ll m)

{

if(m == 0) return 1;

return C(n % mod, m % mod) \* Lucas(n / mod, m / mod) % mod;

}

# A^B快速幂（计算A^B%C的结果）

ll PowerMod(ll a,ll b,ll c)

{

ll ans = 1;

a %= c;

while(b>0)

{

if(b % 2 == 1)

ans = ans \* a % c;

b /= 2;

a = a \* a % c;

}

return ans;

}

# GCD(最小公约数)与LCM（最小公倍数）

ll gcd(ll a,ll b)

{

return a % b ? gcd(b,a%b):b;

}

ll lcm(ll a,ll b)

{

return a\*b/gcd(a,b);

}

# 二分图匹配

//顶点编号从0开始的

const int MAXN = 510;

int uN,vN;//u,v的数目，使用前面必须赋值

int g[MAXN][MAXN];//邻接矩阵

int linker[MAXN];

bool used[MAXN];

bool dfs(int u)

{

for(int v = 0; v < vN;v++)

if(g[u][v] && !used[v])

{

used[v] = true;

if(linker[v] == -1 || dfs(linker[v]))

{

linker[v] = u;

return true;

}

}

return false;

}

int hungary()

{

int res = 0;

memset(linker,-1,sizeof(linker));

for(int u = 0;u < uN;u++)

{

memset(used,false,sizeof(used));

if(dfs(u))

res++;

}

return res;

}

# 求(n!%p的值)

ll func(ll n,ll p)

{

ll result = 1;

for(int i=1;i<=n;i++)

{

result = result \* i % p;

}

return result;

}

# RMQ区间查询

//初始化RMQ,b数组下标从1开始，从0开始简单修改

//RMQ用途：

//对于长度为n的数列A，回答若干询问RMQ（A,i,j）(i,j<=n)，

//返回数列A中下标在i，j之间的最小/大值。

/\*一维

求最大值，数组下标从1开始

求最小值，或者最大最小值下标，或者数组从0开始对应修改即可

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

const int maxn = 1e6;

using namespace std;

const int MAXN = 50010;

int dp[MAXN][20];

int mm[MAXN];

//初始化RMQ,b数组下标从1开始，从0开始简单修改

void initrmq(int n,int b[])

{

mm[0] = -1;

for(int i = 1; i <= n;i++)

{

mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];

dp[i][0] = b[i];

}

for(int j = 1; j <= mm[n];j++)

for(int i = 1;i + (1<<j) -1 <= n;i++)

dp[i][j] = max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);

}

//查询最大值

int rmq(int x,int y)

{

int k = mm[y-x+1];

return max(dp[x][k],dp[y-(1<<k)+1][k]);

}

int a[maxn];

int main()

{

ios::sync\_with\_stdio(false);

int n,q;

cin>>n;

for(int i=1;i<=n;i++)

cin>>a[i];

initrmq(n,a);

cin>>q;

while(q--)

{

int l,r;

cin>>l>>r;

cout<<rmq(l,r)<<endl;

}

return 0;

}

# 素数区间筛选

/\*

素数筛选，筛选出小于等于maxn的素数

prime[0]存的是素数的个数

\*/

const int maxn = 1e4;

int prime[maxn+1];

void getPrime()

{

memset(prime,0,sizeof(prime));

for(int i=2;i<=maxn;i++)

{

if(!prime[i]) prime[++prime[0]] = i;

for(int j=1;j<=prime[0] && prime[j]<=maxn/i;j++)

{

prime[prime[j]\*i] = 1;

if(i%prime[j]==0) break;

}

}

}

# 并查集

//并查集板子

int par[maxn];//父亲

int rank[maxn];//树的高度

//初始化n个元素

void init(int n)

{

for(int i=0;i<n;i++)

{

par[i] = i;

rank[i] = 0;

}

}

//查询树的根

int find(int x)

{

if(par[x]==x)

return x;

else

return par[x] = find(par[x]);

}

//合并x和y所属的集合

void unite(int x,int y)

{

x = find(x);

y = find(y);

if(x==y)return;

if(rank[x]<rank[y])

par[x] = y;

else

{

par[y] = x;

if(rank[x]==rank[y])

rank[x]++;

}

}

//判断x和y是否属于一个集合

bool same(int x,int y)

{

return find(x) == find(y);

}

# 最长回文子串

/\*

求最长回文串

\*/

const int MAXN=110010;

char Ma[MAXN\*2];

int Mp[MAXN\*2];

void Manacher(char s[],int len)

{

int l = 0;

Ma[l++]='$';i

Ma[l++]='#';

for(int i=0;i<len;i++)

{

Ma[l++]=s[i];

Ma[l++]='#';

}

Ma[l]=0;

int mx=0,id=0;

for(int i=0;i<l;i++)

{

Mp[i] = mx > i ? min(Mp[2\*id-i],mx-i):1;

while(Ma[i+Mp[i]]==Ma[i-Mp[i]]) Mp[i]++;

if(i+Mp[i]>mx)

{

mx=i+Mp[i];

id=i;

}

}

}

char s[MAXN];

int main()

{

while(scanf("%s",s)==1)

{

int len=strlen(s);

Manacher(s,len);

int ans=0;

for(int i=0;i<2\*len+2;i++)

ans=max(ans,Mp[i]-1);

printf("%d\n",ans);

}

return 0;

}

# zkw费用流

bool vis[200001];int dist[200001];

//解释一下各数组的含义：vis两个用处：spfa里的访问标记，増广时候的访问标记，dist是每个点的距离标号

int n,m,s,t,ans=0;

//s是起点，t是终点，ans是费用答案

int nedge=-1,p[200001],c[200001],cc[200001],nex[200001],head[200001];

inline void addedge(int x,int y,int z,int zz){

p[++nedge]=y;c[nedge]=z;cc[nedge]=zz;nex[nedge]=head[x];head[x]=nedge;

}

inline bool spfa(int s,int t){

memset(vis,0,sizeof vis);

for(int i=0;i<=n;i++)dist[i]=1e9;dist[t]=0;vis[t]=1;

deque<int>q;q.push\_back(t);

while(!q.empty()){

int now=q.front();q.pop\_front();

for(int k=head[now];k>-1;k=nex[k])if(c[k^1]&&dist[p[k]]>dist[now]-cc[k]){

dist[p[k]]=dist[now]-cc[k];

if(!vis[p[k]]){

vis[p[k]]=1;

if(!q.empty()&&dist[p[k]]<dist[q.front()])q.push\_front(p[k]);else q.push\_back(p[k]);

}

}

vis[now]=0;

}

return dist[s]<1e9;

}

inline int dfs(int x,int low){

if(x==t){vis[t]=1;return low;}

int used=0,a;vis[x]=1;

for(int k=head[x];k>-1;k=nex[k])if(!vis[p[k]]&&c[k]&&dist[x]-cc[k]==dist[p[k]]){

a=dfs(p[k],min(c[k],low-used));

if(a)ans+=a\*cc[k],c[k]-=a,c[k^1]+=a,used+=a;

if(used==low)break;

}

return used;

}

inline int costflow(){

int flow=0;

while(spfa(s,t)){

vis[t]=1;

while(vis[t]){

memset(vis,0,sizeof vis);

flow+=dfs(s,1e9);

}

}

return flow;//这里返回的是最大流，费用的答案在ans里

}

int main()

{

memset(nex,-1,sizeof nex);memset(head,-1,sizeof head);

scanf("%d%d%d%d",&n,&m,&s,&t);

for(int i=1;i<=m;i++){

int x,y,z,zz;scanf("%d%d%d%d",&x,&y,&z,&zz);

addedge(x,y,z,zz);addedge(y,x,0,-zz);

}

printf("%d ",costflow());printf("%d",ans);

return 0;

}

# 最长上升子序列

int dp[maxn];//dp[i]表示递增数量i的最小值

int a[maxn];

int main()

{

ios::sync\_with\_stdio(false);

int n,len = 1;

cin>>n;

for(int i=1;i<=n;i++)

cin>>a[i];

dp[len] = a[1];

for(int i=2;i<=n;i++)

{

if(a[i]>dp[len])

dp[++len] = a[i];

else

{

int pos = lower\_bound(dp+1,dp+len,a[i]) - dp;

//在dp[]找第一个>=a[i]下标

dp[pos] = a[i];

}

}

for(int i=1;i<=len;i++)

cout<<dp[i]<<" ";

cout<<endl;

//cout<<len<<endl;

return 0;

}

# 拓扑排序

const int maxn=1e4+5;

const int maxm=1e5+5;

struct Node

{

int v,w,next;

}edge[maxm];

int no,head[maxn];

int t,n,m;

int deg[maxn],val[maxn];

queue<int>q;

void init()

{

no=0;

memset(head,-1,sizeof(head));

memset(deg,0,sizeof(deg));

memset(val,0,sizeof(val));

}

void topsort()

{

while(!q.empty()) q.pop();

for(int i=1;i<=n;i++)

if(!deg[i]) q.push(i);

int ans=0;

while(!q.empty())

{

int u=q.front();q.pop();

for(int k=head[u];k+1;k=edge[k].next){

int v=edge[k].v;

val[v]=max(val[v],val[u]+edge[k].w);

if(--deg[v]==0) q.push(v);

}

}

cout<<\*max\_element(val+1,val+1+n)<<endl;

}

int main()

{

int t;

cin>>t;

while(t--)

{

cin>>n>>m;

init();

for(int i=1;i<=m;i++){

int u,v,w;

cin>>u>>v>>w;

edge[no].v=v;edge[no].w=w;

edge[no].next=head[u],head[u]=no++;

}

topsort();

}

return 0;

}

# 线段树维护

struct node

{

int left, right;

int max, sum;

};

node tree[maxn << 2];

int a[maxn];

int n;

int k = 1;

int p, q;

string str;

**void build(int m, int l, int r)//m 是 树的标号**

{

tree[m].left = l;

tree[m].right = r;

if (l == r){

tree[m].max = a[l];

tree[m].sum = a[l];

return;

}

int mid = (l + r) >> 1;

build(m << 1, l, mid);

build(m << 1 | 1, mid + 1, r);

tree[m].max = max(tree[m << 1].max, tree[m << 1 | 1].max);

tree[m].sum = tree[m << 1].sum + tree[m << 1 | 1].sum;

}

**void update(int m, int a, int val)//a 是 节点位置， val 是 更新的值（加减的值）**

{

if (tree[m].left == a && tree[m].right == a){

tree[m].max += val;

tree[m].sum += val;

return;

}

int mid = (tree[m].left + tree[m].right) >> 1;

if (a <= mid){

update(m << 1, a, val);

}

else{

update(m << 1 | 1, a, val);

}

tree[m].max = max(tree[m << 1].max, tree[m << 1 | 1].max);

tree[m].sum = tree[m << 1].sum + tree[m << 1 | 1].sum;

}

**int querySum(int m, int l, int r)**

{

if (l == tree[m].left && r == tree[m].right){

return tree[m].sum;

}

int mid = (tree[m].left + tree[m].right) >> 1;

if (r <= mid){

return querySum(m << 1, l, r);

}

else if (l > mid){

return querySum(m << 1 | 1, l, r);

}

return querySum(m << 1, l, mid) + querySum(m << 1 | 1, mid + 1, r);

}

**int queryMax(int m, int l, int r)**

{

if (l == tree[m].left && r == tree[m].right){

return tree[m].max;

}

int mid = (tree[m].left + tree[m].right) >> 1;

if (r <= mid){

return queryMax(m << 1, l, r);

}

else if (l > mid){

return queryMax(m << 1 | 1, l, r);

}

return max(queryMax(m << 1, l, mid), queryMax(m << 1 | 1, mid + 1, r));

}

build(1,1,n);

update(1,a,b);

query(1,a,b);

# Tarjan强连通图分量

/\*

\* Tarjan算法

\* 复杂度O(N+M)

\*/

const int MAXN = 20010;//点数

const int MAXM = 50010;//边数

struct Edge

{

int to,next;

}edge[MAXM];

int head[MAXN],tot;

int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong数组的值是1~scc

int Index,top;

int scc;//强连通分量的个数

bool Instack[MAXN];

int num[MAXN];//各个强连通分量包含点的个数，数组编号1~scc

//num数组不一定需要，结合实际情况

void addedge(int u,int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

}

void Tarjan(int u)

{

int v;

Low[u] = DFN[u] = ++Index;

Stack[top++] = u;

Instack[u] = true;

for(int i = head[u];i != -1;i = edge[i].next)

{

v = edge[i].to;

if( !DFN[v] )

{

Tarjan(v);

if( Low[u] > Low[v] )Low[u] = Low[v];

}

else if(Instack[v] && Low[u] > DFN[v])

Low[u] = DFN[v];

}

if(Low[u] == DFN[u])

{

scc++;

do

{

v = Stack[--top];

Instack[v] = false;

Belong[v] = scc;

num[scc]++;

}while( v != u);

}

}

void solve(int N)

{

memset(DFN,0,sizeof(DFN));

memset(Instack,false,sizeof(Instack));

memset(num,0,sizeof(num));

Index = scc = top = 0;

for(int i = 1;i <= N;i++)

if(!DFN[i])

Tarjan(i);

}

void init()

{

tot = 0;

memset(head,-1,sizeof(head));

}

int main()

{

ios::sync\_with\_stdio(false);

int n,m,x,y;

while(cin>>n>>m)

{

init();

for(int i=1;i<=m;i++)

{

cin>>x>>y;

addedge(x, y);

}

solve(n);

int ans = scc;

cout<<ans<<endl;

}

return 0;

}

# Dij算法

struct qnode

{

int v,c;

qnode(int \_v=0,int \_c=0):v(\_v),c(\_c){}

bool operator <(const qnode &r)const

{

return c>r.c;

}

};

struct Edge

{

int v,cost;

Edge(int \_v=0,int \_cost=0):v(\_v),cost(\_cost){}

};

vector<Edge> E[maxn];

bool vis[maxn];

int dist[maxn];

void Dij(int n,int start)//点的编号从1开始

{

memset(vis,false,sizeof(vis));

for(int i=1;i<=n;i++)

dist[i] = INF;

priority\_queue<qnode> que;

dist[start] = 0;

que.push(qnode(start,0));

qnode tmp;

while(!que.empty())

{

tmp = que.top();

que.pop();

int u = tmp.v;

if(vis[u])

continue;

vis[u] = true;

for(int i=0;i<E[u].size();i++)

{

int v = E[tmp.v][i].v;

int cost = E[u][i].cost;

if(!vis[v]&&dist[v]>dist[u]+cost)

{

dist[v] = dist[u] + cost;

que.push(qnode(v,dist[v]));

}

}

}

}

void addedge(int u,int v,int w)

{

E[u].push\_back(Edge(v,w));

}

void init()

{

memset(E, 0, sizeof(E));

memset(dist,0,sizeof(dist));

}

int main()

{

ios::sync\_with\_stdio(false);

int n,m,from,to,len;

while(cin>>n>>m&&(n||m))

{

init();

for(int i=0;i<m;i++)

{

cin>>from>>to>>len;

addedge(from, to, len);

addedge(to, from, len);

}

Dij(n, 1);

cout<<dist[n]<<endl;

}

return 0;

}

# Dinic算法

Int tot;

Int nex[maxn],first[maxn],w[maxn],v[maxn],n,m,ch[maxn];

void add(int from,int to,int weight){

v[tot]=to;w[tot]=weight;

nex[tot]=first[from];

first[from]=tot++;

}

bool tell(){

memset(ch,-1,sizeof(ch));

queue<int>q;

q.push(1);ch[1]=0;

while(!q.empty()){

int t=q.front();q.pop();

for(int i=first[t];~i;i=nex[i])

if(w[i]&&ch[v[i]]==-1)

q.push(v[i]),ch[v[i]]=ch[t]+1;

}

return ch[n]!=-1;

}

int zeng(int a,int b){

if(a==n)return b;

int r=0;

for(int i=first[a];~i&&b>r;i=nex[i])

if(ch[a]+1==ch[v[i]]&&w[i]){

int t=zeng(v[i],min(b-r,w[i]));

w[i]-=t;w[i^1]+=t;r+=t;

}

if(!r)ch[a]=-1;

return r;

}

int dinic(){

int ans=0,jy;

while(tell())

while(jy=zeng(1,0x3fffffff))

ans+=jy;

return ans;

}

int main(){

while(cin>>m>>n){

memset(first,-1,sizeof(first));

int xx,yy,zz;

tot=0;

for(int i=1;i<=m;i++){

cin>>xx>>yy>>zz;

add(xx,yy,zz);add(yy,xx,0);

}

cout<<dinic()<<endl;

}

}

# 全排列

void Pern(int list[], int k, int n) { // k表示前k个数不动仅移动后面n-k位数

if (k == n - 1) {

for (int i = 0; i < n; i++) {

printf("%d", list[i]);

}

printf("\n");

}else {

for (int i = k; i < n; i++) { // 输出的是满足移动条件所有全排列

swap(list[k], list[i]);

Pern(list, k + 1, n);

swap(list[k], list[i]);

}

}

}

# 二分搜索

/\*

|二分搜索|

|要求：先排序|

\*/

// left为最开始元素, right是末尾元素的下一个数，x是要找的数

int bsearch(int \*A, int left, int right, int x){

int m;

while (left < right){

m = left + (right - left) / 2;

if (A[m] >= x) right = m; else left = m + 1;

// 如果要替换为 upper\_bound, 改为:

if (A[m] <= v) x = m+1;

else y = m;

}

return left;

}

/\*

最后left == right

如果没有找到135577找6，返回7

如果找有多少的x，可以用lower\_bound查找一遍，upper\_bound查找一遍，下标相减

C++自带的lower\_bound(a,a+n,x)返回数组中最后一个x的下一个数的地址

upper\_bound(a,a+n,x)返回数组中第一个x的地址

如果a+n内没有找到x或x的下一个地址，返回a+n的地址

lower\_bound(a,a+n,x)-upper\_bound(a,a+n,x)返回数组中x的个数

\*/

# 背包问题

//0 1背包：

void bag01(int cost,int weight) {

for(i = v; i >= cost; --i)

dp[i] = max(dp[i], dp[i-cost]+weight);

}

// 完全背包：

void complete(int cost, int weight) {

for(i = cost ; i <= v; ++i)

dp[i] = max(dp[i], dp[i - cost] + weight);

}

// 多重背包：

void multiply(int cost, int weight, int amount) {

if(cost \* amount >= v)

complete(cost, weight);

else{

k = 1;

while (k < amount){

bag01(k \* cost, k \* weight);

amount -= k;

k += k;

}

bag01(cost \* amount, weight \* amount);

}

}

// other

int dp[1000000];

int c[55], m[110];

int sum;

void CompletePack(int c) {

for (int v = c; v <= sum / 2; ++v){

dp[v] = max(dp[v], dp[v - c] + c);

}

}

void ZeroOnePack(int c) {

for (int v = sum / 2; v >= c; --v) {

dp[v] = max(dp[v], dp[v - c] + c);

}

}

void multiplePack(int c, int m） {

if (m \* c > sum / 2)

CompletePack(c);

else{

int k = 1;

while (k < m){

ZeroOnePack(k \* c);

m -= k;

k <<= 1;

}

if (m != 0){

ZeroOnePack(m \* c);

}

}

}

# LCS（最长公共子序列）

void solve()

{

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < m; ++j)

{

if (s1[i] == s2[j])

{

dp[i + 1][j + 1] = dp[i][j] + 1;

}

else

{

dp[i + 1][j + 1] = max(dp[i][j + 1], dp[i + 1][j]);

}

}

}

}

# 向量用法

struct node {

double x; // 横坐标

double y; // 纵坐标

};

typedef node Vector;

Vector operator + (Vector A, Vector B)

{ return Vector(A.x + B.x, A.y + B.y); }

Vector operator - (Point A, Point B)

{ return Vector(A.x - B.y, A.y - B.y); }

Vector operator \* (Vector A, double p)

{ return Vector(A.x\*p, A.y\*p); }

Vector operator / (Vector A, double p)

{ return Vector(A.x / p, A.y\*p); }

double Dot(Vector A, Vector B)

{ return A.x\*B.x + A.y\*B.y; } // 向量点乘

double Length(Vector A)

{ return sqrt(Dot(A, A)); } // 向量模长

double Angle(Vector A, Vector B)

{ return acos(Dot(A, B) / Length(A) / Length(B)); } // 向量之间夹角

double Cross(Vector A, Vector B)

{ // 叉积计算 公式

return A.x\*B.y - A.y\*B.x;

}

Vector Rotate(Vector A, double rad) // 向量旋转 公式

{

return Vector(A.x\*cos(rad) - A.y\*sin(rad), A.x\*sin(rad) + A.y\*cos(rad));

}

Point getLineIntersection(Point P, Vector v, Point Q, Vector w)

{ // 两直线交点t1 t2计算公式

Vector u = P - Q;

double t = Cross(w, u) / Cross(v, w); // 求得是横坐标

return P + v\*t; // 返回一个点

}

# 求多边形面积

node G[maxn];

int n;

double Cross(node a, node b) { // 叉积计算

return a.x\*b.y - a.y\*b.x;

}

int main()

{

while (scanf("%d", &n) != EOF && n) {

for (int i = 0; i < n; i++)

scanf("%lf %lf", &G[i].x, &G[i].y);

double sum = 0;

G[n].x = G[0].x;

G[n].y = G[0].y;

for (int i = 0; i < n; i++) {

sum += Cross(G[i], G[i + 1]);

}

sum = sum / 2.0;

printf("%.1f\n", sum);

}

return 0;

}

# 区间更新

typedef long long ll;

const int maxn = 100010;

int t,n,q;

ll anssum;

struct node{

ll l,r;

ll addv,sum;

}tree[maxn<<2];

**void maintain(int id) {**

if(tree[id].l >= tree[id].r)

return ;

tree[id].sum = tree[id<<1].sum + tree[id<<1|1].sum;

}

**void pushdown(int id) {**

if(tree[id].l >= tree[id].r)

return ;

if(tree[id].addv){

int tmp = tree[id].addv;

tree[id<<1].addv += tmp;

tree[id<<1|1].addv += tmp;

tree[id<<1].sum += (tree[id<<1].r - tree[id<<1].l + 1)\*tmp;

tree[id<<1|1].sum += (tree[id<<1|1].r - tree[id<<1|1].l + 1)\*tmp;

tree[id].addv = 0;

}

}

**void build(int id,ll l,ll r) {**

tree[id].l = l;

tree[id].r = r;

tree[id].addv = 0;

tree[id].sum = 0;

if(l==r) {

tree[id].sum = 0;

return ;

}

ll mid = (l+r)>>1;

build(id<<1,l,mid);

build(id<<1|1,mid+1,r);

maintain(id);

}

**void updateAdd(int id,ll l,ll r,ll val) {**

if(tree[id].l >= l && tree[id].r <= r)

{

tree[id].addv += val;

tree[id].sum += (tree[id].r - tree[id].l+1)\*val;

return ;

}

pushdown(id);

ll mid = (tree[id].l+tree[id].r)>>1;

if(l <= mid)

updateAdd(id<<1,l,r,val);

if(mid < r)

updateAdd(id<<1|1,l,r,val);

maintain(id);

}

**void query(int id,ll l,ll r) {**

if(tree[id].l >= l && tree[id].r <= r){

anssum += tree[id].sum;

return ;

}

pushdown(id);

ll mid = (tree[id].l + tree[id].r)>>1;

if(l <= mid)

query(id<<1,l,r);

if(mid < r)

query(id<<1|1,l,r);

maintain(id);

}

int main() {

scanf("%d",&t);

int kase = 0 ;

while(t--){

scanf("%d %d",&n,&q);

**build(1,1,n);**

int id;

ll x,y;

ll val;

printf("Case %d:\n",++kase);

while(q--){

scanf("%d",&id);

if(id==0){

scanf("%lld %lld %lld",&x,&y,&val);

**updateAdd(1,x+1,y+1,val);**

}

else{

scanf("%lld %lld",&x,&y);

anssum = 0;

**query(1,x+1,y+1);**

printf("%lld\n",anssum);

} } }

return 0;

}

# 树状数组

int a[maxn];

int n;

**int lowbit(const int t) {**

return t & (-t);

}

**void insert(int t, int d) {**

while (t <= n){

a[t] += d;

t = t + lowbit(t);

}

}

**ll getSum(int t) {**

ll sum = 0;

while (t > 0){

sum += a[t];

t = t - lowbit(t);

}

return sum;

}

int main() {

int t, k, d;

scanf("%d", &t);

k= 1;

while (t--){

memset(a, 0, sizeof(a));

scanf("%d", &n);

for (int i = 1; i <= n; ++i) {

scanf("%d", &d);

**insert(i, d);**

}

string str;

printf("Case %d:\n", k++);

while (cin >> str) {

if (str == "End") break;

int x, y;

scanf("%d %d", &x, &y);

if (str == "Query")

**printf("%lld\n", getSum(y) - getSum(x - 1));**

else if (str == "Add")

**insert(x, y);**

else if (str == "Sub")

**insert(x, -y);**

}

}

return 0;

}