# Header

#include <stdio.h>  
#include <stdarg.h>  
#include <stdlib.h>  
#include <math.h>  
#include <string.h>  
#include <time.h>  
#include <limits.h>  
#include <vector>  
#include <list>  
#include <set>  
#include <utility> // pair  
#include <map>  
#include <iostream>  
#include <sstream>  
#include <algorithm> // sort  
#include <functional>  
#include <string>  
#include <stack>  
#include <queue>  
#include <fstream>  
#include <bitset>  
  
//#include <unordered\_map>  
//#include <unordered\_set>  
  
using namespace std;  
  
#define ll long long  
#define lll \_\_int128  
#define uchar unsigned char  
#define ushort unsigned short  
#define uint unsigned int  
#define ulong unsigned long  
#define ull unsigned long long  
  
#define INT\_INF 0x7fffffff  
  
#define pi acos(-1)  
  
#define mem(a,b) memset(a,b,sizeof(a))  
#define memn(a,b,c,n) memset(a,b,sizeof(c)\*(n))  
#define fre(a) freopen(a,"r",stdin)  
  
#define cio ios::sync\_with\_stdio(false); // Do not use it with "scanf" and other c input!  
#define pb push\_back  
#define mpair make\_pair  
#define rep(i,a,b) for(int i=a;i<=b;i++)  
#define pre(i,a,b) for(int i=a;i>=b;i--)  
#define REP(i,a,b) for(int i=a;i<b;i++)  
  
#define reada(a,s,n) rep(i,s,n)scanf("%d",a + i);  
#define READa(a,s,n) REP(i,s,n)scanf("%d",a + i);  
  
#define readall(a,s,n) rep(i,s,n)scanf("%lld",a + i);  
#define READall(a,s,n) REP(i,s,n)scanf("%lld",a + i);  
  
//template <typename \_Tp> inline void read(\_Tp&x) {  
// char ch;bool flag=0;x=0;  
// ch=getchar();  
// while(!isdigit(ch)){if(ch=='-')flag=1;ch=getchar();}  
// while(isdigit(ch))x=x\*10+ch-'0',ch=getchar();  
// if(flag)x=-x;  
//}  
//inline void print\_lll(lll x) {  
// if(x<0) {x=-x;putchar('-');}  
// if(x>9) print\_lll(x/10);  
// putchar(x%10+'0');  
//}  
  
#define \_T\_(T) int T;scanf("%d",&T);while(T--)  
#define \_\_T \_T\_(TTESTCASES)  
#define \_E\_(T) while(~T)  
  
#define \_C(a) cout << a << endl;  
  
#define dsci(a) int a;scanf("%d",&a)  
#define dscii(a,b) int a,b;scanf("%d%d",&a,&b)  
#define dsciii(a,b,c) int a,b,c;scanf("%d%d%d",&a,&b,&c)  
#define dscl(a) ll a;scanf("%lld",&a)  
#define dscll(a,b) ll a,b;scanf("%lld%lld",&a,&b)  
#define dsclll(a,b,c) ll a,b,c;scanf("%lld%lld%lld",&a,&b,&c)  
#define dscd(a) double a;scanf("%lf",&a)  
#define dscdd(a,b) double a,b;scanf("%lf%lf",&a,&b)  
#define dscddd(a,b,c) double a,b,c;scanf("%lf%lf%lf",&a,&b,&c)  
  
#define sci(a) scanf("%d",&a)  
#define scii(a,b) scanf("%d%d",&a,&b)  
#define sciii(a,b,c) scanf("%d%d%d",&a,&b,&c)  
#define scl(a) scanf("%lld",&a)  
#define scll(a,b) scanf("%lld%lld",&a,&b)  
#define sclll(a,b,c) scanf("%lld%lld%lld",&a,&b,&c)  
#define scd(a) scanf("%lf",&a)  
#define scdd(a,b) scanf("%lf%lf",&a,&b)  
#define scddd(a,b,c) scanf("%lf%lf%lf",&a,&b,&c)  
  
#define lowbit(x) ((x)&(-(x)))  
  
#define Tprint(a,s,e) REP(i,s,e){if(i!=s)printf(" ");printf("%lld",a[i]);}  
  
#define endl '\n'  
  
#define itn int  
#define iny int  
#define nit int  
#define inr int  
#define mian main  
#define iman main  
#define mina main  
#define mian main  
#define ednl endl  
#define fro for  
#define fir for  
#define reutrn return  
#define retunr return  
#define reutnr return  
#define re0 return 0  
#define re1 return 1

# Utils

## Segment

### 差分

int p[1000] = {0};  
int a[1000];  
int n;  
  
void pls(int l,int r,int k)  
{  
 p[l] += k;  
 p[r + 1] -= k;  
}  
  
void init()  
{  
 REP(i,0,n) {  
 pls(i,i,a[i]);  
 }  
}  
  
int main()  
{  
 scanf("%d",&n);  
   
 READ(a,0,n);  
 init();  
   
 Tprint(p,0,n);  
 printf("\n");  
   
 int q;  
 scanf("%d",&q);  
 int l,r;  
 while (q --) {  
 scanf("%d%d",&l,&r);  
 pls(l,r,1);  
 }  
   
 int s[1000];  
   
 s[0] = p[0];  
 rep(i,1,n - 1) {  
 s[i] = s[i - 1] + p[i];  
 }  
   
 rep(i,0,n - 1) printf("%d ",s[i]);  
 return 0;  
}

### 二维差分

int p[1000][1000] = {0};  
int a[1000][1000];  
  
void pls(int a,int b,int x,int y,int k)  
{  
 p[a][b] += k;  
 p[x + 1][y + 1] += k;  
 p[x][y + 1] -= k;  
 p[x + 1][y] -= k;  
}  
  
int main()  
{  
 int n,m;  
 scanf("%d%d",&n,&m);  
 rep(i,1,n) {  
 rep(j,1,m) {  
 scanf("%d",&a[i][j]);  
 }  
 }  
   
 rep(i,1,n) {  
 rep(j,1,m) {  
 pls(i,j,i,j,a[i][j]);  
 }  
 }  
   
 rep(i,1,n) {  
 rep(j,1,m) {  
 printf("%d ",p[i][j]);  
 }  
 printf("\n");  
 }  
 printf("\n");  
   
 int sum[1000][1000] = {0};  
 sum[1][1] = p[1][1];  
   
 rep(i,1,n) {  
 rep(j,1,m) {  
 sum[i][j] = sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1] + p[i][j];  
 }  
 }  
   
 rep(i,1,n) {  
 rep(j,1,m) {  
 printf("%d ",sum[i][j]);  
 }  
 printf("\n");  
 }  
   
   
 return 0;  
}

### 二维前缀和

int main()  
{  
 int arr[1000][1000] = {0};  
 int sum[1000][1000] = {0};  
 int n,m;  
 scanf("%d%d",&n,&m);  
 rep(i,1,n) {  
 rep(j,1,m) {  
 scanf("%d",&arr[i][j]);  
 }  
 }  
   
 rep(i,1,n) {  
 rep(j,1,m) {  
 printf("%d ",arr[i][j]);  
 }  
 printf("\n");  
 }  
   
 puts("");  
   
 sum[1][1] = arr[1][1];  
// rep(i,1,m) sum[0][i] = sum[0][i - 1] + arr[0][i];  
// rep(i,1,n) sum[i][0] = sum[i - 1][0] + arr[i][0];  
   
 rep(i,1,n) {  
 rep(j,1,m) {  
 sum[i][j] = sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1] + arr[i][j];  
 }  
 }  
   
 // sum up  
 rep(i,1,n) {  
 rep(j,1,m) {  
 printf("%d ",sum[i][j]);  
 }  
 printf("\n");  
 }  
   
 // obtain  
 int q;  
 scanf("%d",&q);  
 int a,b,x,y;  
 while (q --) {  
 scanf("%d%d%d%d",&a,&b,&x,&y);  
 printf("%d\n",sum[x][y] - sum[a - 1][y] - sum[x][b - 1] + sum[a - 1][b - 1]);  
 }  
   
 return 0;  
}

### 树状数组

#### 单点修改 & 区间查询

// 树状数组：单点修改，区间和查询  
const int MAXN = 5e5 + 10;  
  
ll a[MAXN];  
ll c[MAXN];  
  
int n,m;  
  
inline void add(int i,ll x)  
{  
 // 从叶子结点一路向上更新  
 for (;i <= n;i += lowbit(i)) {  
 c[i] += x;  
 }  
}  
  
inline ll sum(int i)  
{  
 // 查询： 由于每个c结点相当于一小段前缀和，因此全+起来，最后求得的便是总共的前缀和  
 ll ans = 0;  
 for (;i > 0;i -= lowbit(i))  
 {  
 ans += c[i];  
 }  
 return ans;  
}  
  
inline ll query(int a,int b)   
{  
 return sum(b) - sum(a - 1);  
}  
  
  
int main()  
{  
 scii(n,m);  
 rep(i,1,n) scl(a[i]);  
 rep(i,1,n) add(i, a[i]);  
 int t,x,y;  
 while (m --) {  
 sciii(t,x,y);  
 if (t == 1) {  
 // add  
 add(x, y);  
 } else if (t == 2) {  
 // query  
 printf("%lld\n",query(x,y));  
 }  
 }  
 re0;  
}

#### (+差分) 区间修改 & 单点查询

// 树状数组+差分： 区间修改+单点查询  
const int MAXN = 5e5 + 10;  
  
int a[MAXN];  
int c[MAXN];  
  
int n,m;  
  
inline void add(int idx,ll x)  
{  
 // 从叶子结点一路向上更新  
 for (int i = idx;i <= n;i += lowbit(i))  
 {  
 c[i] += x;  
 }  
}  
  
ll sum(int idx)  
{  
 // 查询：由于每个c结点相当于一小段前缀和，因此全+起来，最后求得的便是总共的前缀和  
 ll ans = 0;  
 for (int i = idx;i > 0;i -= lowbit(i))  
 {  
 ans += c[i];  
 }  
 return ans;  
}  
  
void pls(int l,int r,int k)  
{  
 add(l, k);  
 add(r + 1, -k);  
}  
  
void init()  
{  
 rep(i,1,n) {  
 pls(i,i,a[i]);  
 }  
}  
  
int main()  
{  
 scii(n,m);  
 rep(i,1,n) sci(a[i]);  
 init();  
   
 int t;  
 int x,y,k;  
   
 while (m --) {  
 sci(t);  
 if (t == 1) {  
 sciii(x,y,k);  
 // modify  
 pls(x,y,k);  
 } else if (t == 2) {  
 sci(x);  
 // get\_ans: sum\_up  
 printf("%lld\n",sum(x));  
 }  
 }  
 return 0;  
}

#### 求最值

int n;  
  
const int MAXN = 1e5 + 10;  
  
int a[MAXN]; // 原数组  
ll c[MAXN]; // 求和树状数组  
  
void modify(int i,ll x)  
{  
 // 从叶子结点一路向上更新  
 for (;i <= n;i += lowbit(i)) {  
 c[i] += x;  
 }  
}  
  
ll sum(int i)  
{  
 // 查询： 由于每个c结点相当于一小段前缀和，因此全+起来，最后求得的便是总共的前缀和  
 ll ans = 0;  
 for (;i > 0;i -= lowbit(i))  
 {  
 ans += c[i];  
 }  
 return ans;  
}  
  
int mx[MAXN]; // 最大值数状数组  
  
void modify\_m(int i,int x)  
{  
 int low;  
 a[i] = x; // 会直接修改数组的值  
 for (;i <= n;i += lowbit(i)) {  
 mx[i] = a[i];  
 low = lowbit(i);  
 for (int j = 1;j < low;j <<= 1) {  
 mx[i] = max(mx[i], mx[i - j]);  
 }  
 }  
}  
  
int query\_max(int l,int r)  
{  
 int ans = max(a[l],a[r]);  
 while(true)  
 {  
 ans = max(ans, a[r]);  
 if (l == r) break;  
 r --;  
 for (;r - l > lowbit(r);r -= lowbit(r))  
 {  
 ans = max(ans,mx[r]);  
 }  
 }  
 return ans;  
}  
  
int main()  
{  
 int m;  
 scii(n,m);  
 rep(i,1,n) {  
 sci(a[i]);  
 modify\_m(i, a[i]); // 修改最大值  
 modify(i, a[i]);  
 }  
 int t,x,y;  
 while (m --) {  
 sciii(t,x,y);  
 if (t == 1) {  
 modify(x, y - a[x]);  
 modify\_m(x, y);  
 } else if (t == 2) {  
 printf("%lld\n",sum(y) - sum(x - 1));  
 } else printf("%d\n",query\_max(x, y));  
 }  
 return 0;  
}

#### 求逆序数

// 树状数组求逆序数  
const int MAXN = 5e4 + 10;  
  
int a[MAXN];  
int b[MAXN];  
ll c[MAXN];  
  
int n,m;  
  
void modify(int idx,ll x)  
{  
 // 从叶子结点一路向上更新  
 for (int i = idx;i <= n;i += lowbit(i)) {  
 c[i] += x;  
 }  
}  
  
ll sum(int idx)  
{  
 // 查询： 由于每个c结点相当于一小段前缀和，因此全+起来，最后求得的便是总共的前缀和  
 ll ans = 0;  
 for (int i = idx;i > 0;i -= lowbit(i))  
 {  
 ans += c[i];  
 }  
 return ans;  
}  
  
int main()  
{  
 int \*x;  
 ll ans;  
 int idx;  
 while (~sci(n)) {  
   
 rep(i,1,n) {  
 sci(a[i]);  
 b[i] = a[i];  
 c[i] = 0;  
 }  
 sort(a + 1, a + 1 + n);  
 x = unique(a + 1, a + 1 + n);  
// for (int \*i = a + 1;i != x;i ++) printf("%d ",\*i);  
// puts("");  
 ans = 0;  
 pre(i,n,1) {  
 idx = (int) (lower\_bound(a + 1, x, b[i]) - a);  
   
 ans += sum(idx);  
 modify(idx, 1);  
 }  
 printf("%lld\n",ans);  
 }  
 re0;  
}

### 线段树

修改和增加不能同时使用

#### 添加树(add)

// 线段树 - 二叉树，节点存的是一个 l，r，区间的内容n  
const int MAXN = 1e5 + 10;  
  
struct Node {  
 int l,r;  
 ll mx;  
 ll mn;  
 ll sum;  
 int lazy;  
 ll lzn;  
} tree[MAXN << 2];  
ll a[MAXN];  
  
void push\_up(int i)  
{  
 tree[i].sum = tree[i << 1].sum + tree[i << 1 | 1].sum;  
 tree[i].mn = min(tree[i << 1].mn,tree[i << 1 | 1].mn);  
 tree[i].mx = max(tree[i << 1].mx,tree[i << 1 | 1].mx);  
}  
  
// 下推标记(Add)  
void push\_down(int i)  
{  
 if (tree[i].lazy) {  
 tree[i << 1].sum += (tree[i << 1].r - tree[i << 1].l + 1) \* tree[i].lzn;  
 tree[i << 1 | 1].sum += (tree[i << 1 | 1].r - tree[i << 1 | 1].l + 1) \* tree[i].lzn;  
   
 tree[i << 1].mx += tree[i].lzn;  
 tree[i << 1 | 1].mx += tree[i].lzn;  
   
 tree[i << 1].mn += tree[i].lzn;  
 tree[i << 1 | 1].mn += tree[i].lzn;  
   
 tree[i << 1].lzn += tree[i].lzn;  
 tree[i << 1 | 1].lzn += tree[i].lzn;  
   
 tree[i << 1].lazy = tree[i].lazy;  
 tree[i << 1 | 1].lazy = tree[i].lazy;  
   
 tree[i].lazy = 0;  
 tree[i].lzn = 0;  
 }  
}  
  
// i - 二叉树节点编号，调用时取1  
// l，r 区间左右端下标，调用的时候取最大范围即可 build(1,n,1);  
void build(int l,int r,int i)  
{  
 tree[i].l = l;  
 tree[i].r = r;  
 tree[i].lazy = 0;  
 tree[i].lzn = 0;  
 if (l == r) {  
 tree[i].sum = a[l]; // a原数组，把原来的数值给叶子结点  
 tree[i].mx = a[l];  
 tree[i].mn = a[l];  
 return;  
 }  
 int m = (l + r) >> 1;  
 build(l,m,i << 1);  
 build(m + 1,r,i << 1 | 1);  
 push\_up(i);  
}  
  
void add(int l,int r,ll x,int i) // 将区间[l,r]整个加上x，调用(l,r,x,1)  
{  
 if (l <= tree[i].l && r >= tree[i].r) {  
 tree[i].sum += (tree[i].r - tree[i].l + 1) \* x;  
 tree[i].mx += x;  
 tree[i].mn += x;  
 tree[i].lzn += x;  
   
 tree[i].lazy = 1;  
 return;  
 }  
 push\_down(i);  
 int m = (tree[i].l + tree[i].r) >> 1;  
 if (l <= m) add(l,r,x,i << 1);  
 if (r > m) add(l,r,x,i << 1 | 1);  
 push\_up(i);  
}  
  
ll query(int l,int r,int i) //查询  
{  
 if (l <= tree[i].l && r >= tree[i].r){  
 return tree[i].sum;  
// return tree[i].mx;  
// return tree[i].mn;  
 }  
 push\_down(i);  
 int m = (tree[i].l + tree[i].r) >> 1;  
 ll sum = 0;  
// ll mx = 0;  
// ll mn = INT\_INF;  
 if (l <= m) {  
 sum += query(l,r,i << 1);  
// mx = max(mx,query(l,r,i << 1));  
// mn = min(mn,query(l,r,i << 1));  
 }  
 if (r > m) {  
 sum += query(l,r,i << 1 | 1);  
// mx = max(mx,query(l,r,i << 1 | 1));  
// mn = min(mn,query(l,r,i << 1 | 1));  
 }  
 return sum;  
// return mx;  
// return mn;  
}  
  
int main()  
{  
 dscii(n,m);  
 rep(i,1,n) scl(a[i]);  
 build(1, n, 1);  
 int k,a,b;  
 ll c;  
 while (m --) {  
 sciii(k,a,b);  
 if (k == 1) {  
 printf("%lld\n", query(a, b, 1));  
 } else if (k == 2) {  
 scl(c);  
 add(a, b, c, 1);  
 rep(i,1,n) printf("a[%d]=%lld\n",i,query(i, i, 1));  
 }  
 }  
 re0;  
}

#### 修改树(Modify)

// 下推标记(Modify)  
void push\_down(int i)  
{  
 if (tree[i].lazy) {  
 tree[i << 1].sum = (tree[i << 1].r - tree[i << 1].l + 1) \* tree[i].lzn;  
 tree[i << 1 | 1].sum = (tree[i << 1 | 1].r - tree[i << 1 | 1].l + 1) \* tree[i].lzn;  
   
 tree[i << 1].mx = tree[i].lzn;  
 tree[i << 1 | 1].mx = tree[i].lzn;  
   
 tree[i << 1].mn = tree[i].lzn;  
 tree[i << 1 | 1].mn = tree[i].lzn;  
   
 tree[i << 1].lzn = tree[i].lzn;  
 tree[i << 1 | 1].lzn = tree[i].lzn;  
   
 tree[i << 1].lazy = tree[i].lazy;  
 tree[i << 1 | 1].lazy = tree[i].lazy;  
   
 tree[i].lazy = 0;  
 tree[i].lzn = 0;  
 }  
}  
  
void modify(int l,int r,ll x,int i) // 将区间[l,r]直接变成x，调用(l,r,x,1)  
{  
 if (l <= tree[i].l && r >= tree[i].r) {  
 tree[i].sum = (tree[i].r - tree[i].l + 1) \* x;  
 tree[i].mx = x;  
 tree[i].mn = x;  
 tree[i].lzn = x;  
  
 tree[i].lazy = 1;  
 return;  
 }  
 push\_down(i);  
 int m = (tree[i].l + tree[i].r) >> 1;  
 if (l <= m) modify(l,r,x,i << 1);  
 if (r > m) modify(l,r,x,i << 1 | 1);  
 push\_up(i);  
}

### 主席树

求区间第k小

// 主席树求区间第k小  
  
const int maxn 1000007  
const int MAXN 18600007  
  
int L[MAXN],R[MAXN],Sum[MAXN],T[maxn],TP; // 左右子树，总和，树根，指针  
void Add(int &rt,int l,int r,int x) { // 建立新树  
 ++TP;L[TP]=L[rt];R[TP]=R[rt];Sum[TP]=Sum[rt]+1;rt=TP; // 复制&新建  
 if(l==r) return;  
 int m=(l+r)>>1;  
 if(x <= m) Add(L[rt],l,m,x);  
 else Add(R[rt],m+1,r,x);  
}  
int Search(int TL,int TR,int l,int r,int k){ // 区间查询第k大  
 if(l==r) return l; // 返回第k大的下标  
 int m=(l+r)>>1;  
 if(Sum[L[TR]]-Sum[L[TL]]>=k) return Search(L[TL],L[TR],l,m,k);  
 else return Search(R[TL],R[TR],m+1,r,k-Sum[L[TR]]+Sum[L[TL]]);  
}  
  
// 常规  
  
int a[maxn];  
// 离散化  
struct A{  
 int x,id;  
 bool operator<(const A&B)const{return x<B.x;} // TODO: <是区间第k小，>是区间第k大  
}ID[maxn];  
map<int,int> mp;  
int Rank[maxn];  
  
void build(int n) {  
 rep(i, 1, n) {  
 ID[i].x = a[i];  
 ID[i].id = i;  
 }  
 sort(ID+1,ID+n+1);  
 mp.clear();  
 rep(i,1,n) {  
 Rank[i]=ID[i].x;  
 mp[ID[i].x]=i;  
 }  
 // 初始化主席树  
 L[0]=R[0]=Sum[0]=T[0]=TP=0;  
 // 建主席树  
 rep(i,1,n) Add(T[i]=T[i-1],1,n,mp[a[i]]);  
}  
  
inline int query(int l,int r,int k,int n)  
{  
 return Rank[Search(T[l-1],T[r],1,n,k)];  
}  
  
int main()  
{  
 int q;  
 int l,r,k;  
 int n;  
 while(~scanf("%d%d",&n,&q)){  
 // 读取输入+离散化  
 rep(i,1,n) {  
 scanf("%d",a + i);  
 }  
  
 build(n);  
  
 // 开始计算  
 while (q --) {  
 scanf("%d%d%d",&l,&r,&k);  
 printf("%d\n",query(l,r,k,n));  
 }  
 }  
 return 0;  
}

## 并查集

### 路径压缩

const int MAXN = 1e5 + 10;  
  
int find\_set[MAXN];  
  
int find(int a)  
{  
 if (find\_set[a] == a) return a;  
 return find\_set[a] = find(find\_set[a]);  
}  
  
inline void bind(int a,int b)  
{  
 find\_set[find(a)] = find(b);  
}  
  
void init(int n)  
{  
 rep(i,0,n) {  
 find\_set[i] = i; // 每个种类初始状态只有自己一个点  
 }  
}

### 路径压缩+按秩合并

const int MAXN = 1e5 + 10;  
  
int find\_set[MAXN];  
int depth[MAXN];  
  
int find(int a)  
{  
 if (find\_set[a] == a) return a;  
 return find\_set[a] = find(find\_set[a]);  
}  
  
inline void bind(int a,int b)  
{  
 int x = find(a), y = find(b);  
 if (depth[x] >= depth[y]) { // 如果a的 根的子树深度 比b的 根的子树深度 大，那a的根继续做根  
 find\_set[y] = x; // 改变b节点的根的根为a的根  
 if (depth[x] == depth[y]) { // 俩根深度一样  
 if (x != y) depth[x] ++; // 作为a的根，自然子树的深度++  
 }  
 } else find\_set[x] = y;  
}  
  
void init(int n)  
{  
 rep(i,0,n) {  
 find\_set[i] = i; // 每个种类初始状态只有自己一个点  
 depth[i] = 1; // 初始化秩  
 }  
}

## 最长上升子序列LIS

### dp:

// 最长上升子序列LIS\_dp http://acm.hdu.edu.cn/showproblem.php?pid=1257  
const int MAXN = 1e5 + 10;  
  
int a[MAXN];  
int dp[MAXN];  
  
int LIS(int \*a,int n) // start from zero  
{  
 memset(dp,0,sizeof(dp));  
 dp[0] = 1;  
 int ans = 1;  
 REP(i,1,n) {  
 REP(j,0,i) {  
 if (a[i] > a[j]) {  
 dp[i] = max(dp[i],dp[j]); // 此处可以使用树状数组进行优化  
 }  
 }  
 ans = max(ans,++ dp[i]);  
 }  
 return ans;  
}  
  
int main()  
{  
 int n;  
 while (~scanf("%d",&n)) {  
 rep(i,1,n) scanf("%d",a + i);  
 printf("%d\n",LIS(a + 1,n));  
 }  
   
}

### 贪心

// 最长上升子序列LIS\_greedy http://acm.hdu.edu.cn/showproblem.php?pid=1257  
const int MAXN = 1e5 + 10;  
  
int a[MAXN];  
int low[MAXN];  
  
int LIS(int \*a,int n) // start from zero  
{  
 int cnt = 0;  
 int t;  
 REP(i,0,n) {  
 t = (int) (lower\_bound(low, low + cnt, a[i]) - low);  
 low[t] = a[i];  
 if (t == cnt) cnt ++;  
 }  
 return cnt;  
}  
  
int main()  
{  
 int n;  
 while (~scanf("%d",&n)) {  
 rep(i,1,n) scanf("%d",a + i);  
 printf("%d\n",LIS(a + 1,n));  
 }  
}

## ST表

const int MAXN = 100010;  
  
int st[MAXN][20];  
int a[MAXN];  
  
int n,m;  
  
inline int read\_int()  
{  
 int x = 0,f = 1;  
 char ch = getchar();  
 while (!isdigit(ch)) {  
 if (ch == '-') f = -1;  
 ch = getchar();  
 }  
 while (isdigit(ch))  
 {  
 x = x \* 10 + ch - 48;  
 ch = getchar();  
 }  
 return x \* f;  
}  
  
void init() {  
 // 定义 st[i][j] 是从i开始，到i + 2^j这一段，即[i,i + 2^j]这一段中的最大/小值  
 rep(i,1,n) st[i][0] = a[i];  
  
 for (int j = 1;(1 << j) <= n;j ++) { // 遍历所有的j，j是一个很小的数字，最大值=log2(n)  
 rep(i,1,n - (1 << j) + 1) { // 在[1,n]区间范围内，确定j的情况下，把所有的i都遍历求值一遍  
 st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]); // 套公式  
 }  
 }  
}  
  
int query(int l, int r)  
{  
 int x = log2(r - l + 1);  
 return max(st[l][x],st[r - (1 << x) + 1][x]);  
}  
  
int main()  
{  
 scanf("%d%d",&n,&m);  
 rep(i,1,n) {  
 a[i] = read\_int();  
 }  
 init();  
 int l,r;  
 while (m --) {  
 scanf("%d%d",&l,&r);  
 printf("%d\n",query(l, r));  
 }  
 return 0;  
}

## 三分

与二分不同的是三分，二分要求函数有单调性，而三分是用于确定一个有极值的函数的极值。

### 例题

HDU http://acm.hdu.edu.cn/showproblem.php?pid=2899  
Now, here is a fuction:

Can you find the minimum value when x is between 0 and 100.  
**Input:** The first line of the input contains an integer T(1<=T<=100) which means the number of test cases. Then T lines follow, each line has only one real numbers Y.(0 < Y <1e10)  
**Output:** Just the minimum value (accurate up to 4 decimal places),when x is between 0 and 100.

/// HDU http://acm.hdu.edu.cn/showproblem.php?pid=2899  
/// 三分  
inline double gpow(double x,int times) {  
 double ans = 1;  
 rep(i,1,times) ans \*= x;  
 return ans;  
}  
  
inline double f(double x,double y) {  
 return 6 \* gpow(x,7) + 8 \* gpow(x,6) + 7 \* gpow(x,3) + 5 \* gpow(x,2) - y \* x;  
}  
  
int main()  
{  
 double y;  
 double l,r,l3,r3;  
 \_\_T {  
 scanf("%lf",&y);  
 l = 0;  
 r = 100;  
 do {  
 l3 = (l \* 2 + r) / 3; // 三等分左半边  
 r3 = (l + r \* 2) / 3; // 三等分右半边  
 if (f(l3,y) > f(r3,y)) l = l3; // 计算两个函数值，题目要求最小值，可根据图像得出（👀假设两个点在单调区间上，比较容易确定怎么赋值）  
 else r = r3;  
 } while (fabs(l3 - r3) > 1e-7);  
 printf("%.4f\n",f(l3,y));  
 }  
}

## rope

**奇淫怪巧：rope是一种类似块状链表操作的东西，速度很快，底层是可持续化平衡树实现。**

*只有gnu编译器可以使用，clang不行。*

### 头文件加入

#include <ext/rope>  
using namespace \_\_gnu\_cxx;

### 操作

push\_back(x); //在末尾添加x  
  
insert(pos,x); //在pos插入x，自然支持整个char数组的一次插入  
  
erase(pos,x); //从pos开始删除x个  
  
copy(pos,len,x); //从pos开始到pos+len为止用x代替  
  
replace(pos,x); //从pos开始换成x  
  
substr(pos,x); //提取pos开始x个  
  
at(x); //访问第x个元素  
  
[x] //访问第x个元素

## 约瑟夫环

n个人围成一圈，第一个人从1开始报数，报m的将被杀掉，下一个人接着从1开始报。如此反复，最后剩下一个，求最后的胜利者。

int cir(int n,int m)  
{  
 int p = 0;  
 for (int i = 2;i <= n;i ++)  
 {  
 p = (p + m) % i;  
 }  
 return p + 1;  
}

# Math

## 杂论

* 任何一个正整数一定能分解成若干个不重复且不相邻的斐波那契数之和+贪心 https://www.zhihu.com/question/68360587?sort=created

## gcd

ll gcd(ll a,ll b)  
{  
 if (a % b == 0) return b;  
 return gcd(b, a % b);  
}

### quickgcd

ll quick\_gcd(ll a, ll b)  
{  
 if(a == 0) return b;  
 if(b == 0) return a;  
 if(!(a & 1) && !(b & 1)) // a % 2 == 0 && b % 2 == 0;  
 return quick\_gcd(a >> 1, b >> 1) << 1;  
 else if(!(b & 1))  
 return quick\_gcd(a, b >> 1);  
 else if(!(a & 1))  
 return quick\_gcd(a >> 1, b);  
 else  
 return quick\_gcd(abs(a - b), min(a, b));  
}

### exgcd

/// 欧几里得扩展exgcd  
/// ax + by = gcd(a,b)  
ll exgcd(ll a, ll b, ll &x, ll &y)  
{  
 if (b == 0)  
 {  
 x = 1;  
 y = 0;  
 return a;  
 }  
 ll r = exgcd(b, a % b, x, y);  
 ll t = y;  
 y = x - (a / b) \* y;  
 x = t;  
 return r;  
}

#### ax + by = gcd(a,b)

**ax + by = gcd(a,b)的通解为：**

## quickpow

* 可以求a的逆元:quickpow(a % mod,mod - 2)，注意a的范围，它也要%mod过才不会爆精度。

const int mod = 1000000007;  
  
ll quickpow(ll a, ll b)  
{  
 ll ans = 1;  
 while (b)  
 {  
 if (b & 1) ans = a \* ans % mod;  
 a = a \* a % mod;  
 b >>= 1;  
 }  
 return ans;  
}

## 组合数

### 理论

组合数公式：https://www.zybang.com/question/6c5da15707d5ddfaff3eff3dc012d24f.html

**n个不同元素,取m个可重复取**

* 考虑顺序且可重复：
* 考虑顺序不可重复：
* 不考虑顺序且可重复：
* 不考虑顺序不可重复：

### 快速求C(n,m)

const int MAXN = 2000010;  
const int mod = 998244353;  
  
ll fac[MAXN],invfac[MAXN],invn[MAXN];  
  
ll C(ll n,ll m){  
 if (n < 0 || m < 0 || n < m) return 0;  
 return fac[n] \* invfac[m] % mod \* invfac[n - m] % mod;  
}  
  
void init() {  
 fac[0] = fac[1] = invn[0] = invn[1] = invfac[0] = invfac[1] = 1;  
 for (int i = 2;i < MAXN;i ++){  
 fac[i] = fac[i - 1] \* i % mod;  
 invn[i] = (mod - mod / i) \* invn[mod % i] % mod;  
 invfac[i] = invfac[i - 1] \* invn[i] % mod;  
 }  
}

## 计算几何

### 基本

const double eps = 1e-8;  
  
int sign(double x)  
{  
 if (fabs(x) < eps) return 0;  
 if (x < 0) return -1;  
 return 1;  
}  
  
struct Point  
{  
 double x,y;  
 Point() {}  
 Point(double \_x,double \_y):x(\_x),y(\_y) {}  
 Point(const Point &o):x(o.x),y(o.y) {}  
   
 Point operator-(const Point &b) const  
 {  
 return Point(x - b.x,y - b.y);  
 }  
   
 // 叉积  
 double operator^(const Point &b) const  
 {  
 return x \* b.y - y \* b.x;  
 }  
   
 // 点积  
 double operator\*(const Point &b) const  
 {  
 return x \* b.x + y \* b.y;  
 }  
   
 // 变成单位向量  
 void identity() {  
 double k = sqrt(x \* x + y \* y);  
 x /= k;  
 y /= k;  
 }  
   
 // 绕原点旋转角度B（弧度值），后x,y的变化  
 void transXY(double B)  
 {  
 double tx = x,ty = y;  
 x = tx \* cos(B) - ty \* sin(B);  
 y = tx \* sin(B) + ty \* cos(B);  
 }  
};  
  
  
struct Line  
{  
 Point s,e;  
 Line() {}  
 Line(Point \_s,Point \_e):s(\_s),e(\_e) {}  
   
 // 两直线相交求交点  
 // 第一个值为0表示直线重合，为1表示平行，为0表示相交,为2是相交  
 // 只有第一个值为2时，交点才有意义  
 pair<int,Point> operator &(const Line &b) const  
 {  
 Point res = s;  
 if (sign((s - e) ^ (b.s - b.e)) == 0)  
 {  
 if (sign((s - b.e) ^ (b.s - b.e)) == 0) return make\_pair(0,res); // 重合  
 else return make\_pair(1,res); // 平行  
 }  
 double t = ((s - b.s) ^ (b.s - b.e)) / ((s - e) ^ (b.s - b.e));  
 res.x += (e.x-s.x) \* t;  
 res.y += (e.y-s.y) \* t;  
 return make\_pair(2,res);  
 }  
};  
  
// \*两点间距离  
double dis(Point a,Point b)  
{  
 return sqrt((a - b) \* (a - b));  
}

### 判断两线(段)相交

// \*判断线段相交  
bool inter(Line a,Line b)  
{  
 return max(a.s.x,a.e.x) >= min(b.s.x,b.e.x) &&  
 max(b.s.x,b.e.x) >= min(a.s.x,a.e.x) &&  
 max(a.s.y,a.e.y) >= min(b.s.y,b.e.y) &&  
 max(b.s.y,b.e.y) >= min(a.s.y,a.e.y) &&  
 sign((b.s - a.e) ^ (a.s - a.e)) \* sign((b.e - a.e) ^ (a.s - a.e)) <= 0 &&  
 sign((a.s - b.e) ^ (b.s - b.e)) \* sign((a.e - b.e) ^ (b.s - b.e)) <= 0;  
}  
  
// 判断直线和线段相交  
bool isLineAcrossSegment(Line a,Line b) // 判断直线a和线段b是否相交  
{  
 return sign((b.s - a.e) ^ (a.s - a.e)) \* sign((b.e - a.e) ^ (a.s - a.e)) <= 0;  
}

### 点到线(段)距离

// 点到直线距离  
// 返回为result,是点到直线最近的点  
Point pointToLine(Point P,Line L)  
{  
 Point result;  
 double t = ((P - L.s) \* (L.e - L.s)) / ((L.e - L.s) \* (L.e - L.s));  
 result.x = L.s.x + (L.e.x - L.s.x) \* t;  
 result.y = L.s.y + (L.e.y - L.s.y) \* t;  
 return result;  
}  
  
// 点到线段的距离  
// 返回点到线段最近的点  
Point nearestPointToLineSeg(Point P,Line L)  
{  
 Point result;  
 double t = ((P - L.s) \* (L.e - L.s)) / ((L.e - L.s) \* (L.e - L.s));  
 if (t >= 0 && t <= 1)  
 {  
 result.x = L.s.x + (L.e.x - L.s.x) \* t;  
 result.y = L.s.y + (L.e.y - L.s.y) \* t;  
 } else {  
 if (dis(P,L.s) < dis(P,L.e))  
 result = L.s;  
 else result = L.e;  
 }  
 return result;  
}

### 计算多边形面积

// 计算多边形面积  
// 点的编号从0~n-1  
double calcArea(Point p[],int n)  
{  
 double res = 0;  
 for (int i = 0;i < n;i++)  
 res += (p[i] ^ p[(i + 1) % n]) / 2;  
 return fabs(res);  
}

### 点是否在线段上

// \*判断点在线段上  
bool isOnSeg(Point P,Line L)  
{  
 return sign((L.s - P) ^ (L.e - P)) == 0 &&  
 sign((P.x - L.s.x) \* (P.x - L.e.x)) <= 0 &&  
 sign((P.y - L.s.y) \* (P.y - L.e.y)) <= 0;  
}

## 质数相关

### 欧拉筛

const int MAXN = 1e8+5;  
  
int num[MAXN], primes[MAXN]; // primes数组存的是所有的质数, num[数字]判断 数字 是否为质数  
int p\_cnt = 0;  
void Euler\_Sieve(int n) {  
 memset(num, -1, sizeof(num));  
 for (int i = 2;i <= n;i ++){  
 if (num[i]) primes[p\_cnt ++] = i;  
 for (int j = 0;j < p\_cnt && 1LL \* i \* primes[j] <= n;j ++){  
 num[i \* primes[j]] = 0;  
 if (i % primes[j] == 0) break;  
 }  
 }  
}

### 区间筛

const int MAXN = 1e7 + 10;  
  
int is\_prime[MAXN];  
int is\_prime\_small[MAXN];  
  
void segment\_sieve(ll a,ll b)  
{  
 for (ll i = 0;i \* i <= b;i ++) is\_prime\_small[i] = 1;  
 for (ll i = 0;i <= b - a;i ++) is\_prime[i] = 1;  
 for (ll i = 2;i \* i <= b;i ++)  
 {  
 if (is\_prime\_small[i])  
 {  
 for (ll j = 2 \* i;j \* j <= b;j += i) is\_prime\_small[j] = 0;  
 for (ll j = max(2LL,(a + i - 1) / i) \* i; j <= b;j += i) is\_prime[j - a] = 0;  
 }  
 }  
}  
  
ll primes[MAXN];  
  
int main()  
{  
 ll l,r;  
 scll(l,r);  
 segment\_sieve(l, r);  
 int cnt = 0;  
 for (ll i = 0;i <= r - l;i ++) {  
 if (i + l != 1 && is\_prime[i]) { // 注意特判断1  
 primes[cnt ++] = i + l; // 所有质数  
 }  
 }  
}

### 随机筛（超大数质因子分解）

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Miller\_Rabin 算法进行素数测试  
 \* 速度快，可以判断一个 < 2^63 的数是不是素数  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
const int S = 9; //随机算法判定次数，一般8~10就够了  
  
// 计算ret = (a\*b)%c a,b,c < 2^63  
ll mult\_mod(ll a,ll b,ll c)  
{  
 a %= c;  
 b %= c;  
 ll ret = 0;  
 ll tmp = a;  
 while (b) {  
 if (b & 1) {  
 ret += tmp;  
 if (ret > c)ret -= c;//直接取模慢很多  
 }  
 tmp <<= 1;  
 if (tmp > c) tmp -= c;  
 b >>= 1;  
 }  
 return ret;  
}  
  
// 计算 ret = (a^n)%mod  
ll pow\_mod(ll a,ll n,ll mod)  
{  
 ll ret = 1;  
 ll temp = a % mod;  
   
 while (n)  
 {  
 if (n & 1) ret = mult\_mod(ret,temp,mod);  
 temp = mult\_mod(temp,temp,mod);  
 n >>= 1;  
 }  
 return ret;  
}  
   
// 通过 a^(n-1)=1(mod n)来判断n是不是素数  
// n-1 = x\*2^t 中间使用二次判断  
// 是合数返回true, 不一定是合数返回false  
bool check(ll a,ll n,ll x,ll t)  
{  
 ll ret = pow\_mod(a,x,n);  
 ll last = ret;  
 for (int i = 1;i <= t;i ++)  
 {  
 ret = mult\_mod(ret,ret,n);  
 if (ret == 1 && last != 1 && last != n-1) return true; //合数  
 last = ret;  
 }  
 if (ret != 1) return true;  
 else return false;  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Miller\_Rabin算法  
// 是素数返回true,(可能是伪素数)  
// 不是素数返回false  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
bool Miller\_Rabin(ll n)  
{  
 if (n < 2) return false;  
 if (n == 2) return true;  
 if ((n & 1) == 0) return false; //偶数  
 ll x = n - 1;  
 ll t = 0;  
 while ((x & 1) ==0) {  
 x >>= 1;  
 t ++;  
 }  
 srand((uint) time(NULL));  
 ll a;  
 for (int i = 0;i < S;i ++)  
 {  
 a = rand() % (n - 1) + 1;  
 if (check(a,n,x,t)) return false;  
 }  
 return true;  
}  
   
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// pollard\_rho 算法进行质因素分解  
//  
//  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
   
ll factor[100]; //质因素分解结果(刚返回时时无序的)  
int tol; //质因素的个数，编号0~tol-1  
  
ll gcd(ll a,ll b) {  
 ll t;  
 while (b)  
 {  
 t = a;  
 a = b;  
 b = t % b;  
 }  
 if (a >= 0) return a;  
 else return -a;  
}  
  
// 找出一个因子  
ll pollard\_rho(ll x,ll c) {  
 ll i = 1, k = 2;  
 srand((uint) time(NULL));  
 ll x0 = rand() % (x - 1) + 1;  
 ll y = x0;  
 ll d;  
 while (1)  
 {  
 i ++;  
 x0 = (mult\_mod(x0,x0,x) + c) % x;  
 d = gcd(y - x0,x);  
 if (d != 1 && d != x) return d;  
 if (y == x0) return x;  
 if (i == k) {  
 y = x0;  
 k += k;  
 }  
 }  
}  
  
// 对n进行素因子分解，存入factor. k设置为107左右即可  
void findfac(ll n,int k)  
{  
 if (n == 1) return;  
 if (Miller\_Rabin(n)) {  
 factor[tol ++] = n;  
 return;  
 }  
 ll p = n;  
 int c = k;  
 while (p >= n) p = pollard\_rho(p,c --); // 值变化，防止死循环k  
 findfac(p,k);  
 findfac(n / p,k);  
}  
  
  
  
// 找出一个数字所有的质因子，若有重复的质因子，则输出yes CCPC2020威海  
int main()  
{  
 ll n;  
 int ok = 0;  
 \_\_T {  
 scl(n); // 读取一个ll数  
 tol = 0; // 质数的个数=0  
 findfac(n, 107); // 找出所有的因子  
 sort(factor, factor + tol); // 排序  
   
 ok = 0;  
 REP(i,1,tol) {  
 if (factor[i - 1] == factor[i]) { // 找出了一样的了  
 ok = 1;  
 break;  
 }  
 }  
 puts(ok ? "yes" : "no");  
 }  
}

### Min25筛

#### 求和

**快速求的质数和**

const int MAXN = 1000010;  
  
namespace Min25 {  
 int prime[MAXN], id1[MAXN], id2[MAXN], flag[MAXN], ncnt, m;  
  
 ll g[MAXN], sum[MAXN], a[MAXN], T, n;  
  
 inline int ID(ll x) {  
 return x <= T ? id1[x] : id2[n / x];  
 }  
  
 inline ll calc(ll x) {  
 return x \* (x + 1) / 2 - 1;  
 }  
  
 inline ll f(ll x) {  
 return x;  
 }  
  
 inline void init() {  
   
 rep(i,0,MAXN - 1) {  
 prime[i] = id1[i] = id2[i] = flag[i] = 0;  
 g[i] = sum[i] = a[i] = 0;  
 }  
   
 ncnt = 0;  
 m = 0;  
 T = sqrt(n + 0.5);  
 for (int i = 2; i <= T; i++) {  
 if (!flag[i]) {  
 prime[++ncnt] = i;  
 sum[ncnt] = sum[ncnt - 1] + i;  
 }  
 for (int j = 1; j <= ncnt && i \* prime[j] <= T; j++) {  
 flag[i \* prime[j]] = 1;  
 if (i % prime[j] == 0) break;  
 }  
 }  
 for (ll l = 1; l <= n; l = n / (n / l) + 1) {  
 a[++m] = n / l;  
 if (a[m] <= T) id1[a[m]] = m; else id2[n / a[m]] = m;  
 g[m] = calc(a[m]);  
 }  
 for (int i = 1; i <= ncnt; i++)  
 for (int j = 1; j <= m && (ll)prime[i] \* prime[i] <= a[j]; j++)  
 g[j] = g[j] - (ll)prime[i] \* (g[ID(a[j] / prime[i])] - sum[i - 1]);  
 }  
  
 inline ll solve(ll x) {  
 if (x <= 1) return x;  
 n = x;  
 init();  
 return g[ID(n)];  
 }  
  
}

## Euler Function

是x的所有质因子。

欧拉函数是小于或等于n的正整数中与n互质的数的数目。

ll eular(ll x)  
{  
 ll ans = x;  
 for (int i = 2;i \* i <= x;i ++)  
 {  
 if (x % i == 0)  
 {  
 ans = ans / i \* (i - 1);  
 while (x % i == 0) x /= i;  
 }  
 }  
 if (x > 1) ans = ans / x \* (x - 1);  
 return ans;  
}

### 欧拉降幂(指数取余)

const int mod = 1000000007;  
ll e = eular(mod);  
ll ans = quickpow(a % mod,b % e + e);

## 多点最小圆覆盖

const double EPS = 1e-12;  
  
const int MAXN = 100010;  
  
struct Node {  
 double x, y;  
} s[MAXN];  
  
Node o;//圆心坐标  
double r;//半径  
  
double dis(Node a, Node b) {  
 return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));  
}  
  
void getr(Node p1, Node p2, Node p3) {//三个点求三角形圆心坐标和半径  
 double a, b, c, d, e, f;  
 a = p2.y - p1.y;  
 b = p3.y - p1.y;  
 c = p2.x - p1.x;  
 d = p3.x - p1.x;  
 f = p3.x \* p3.x + p3.y \* p3.y - p1.x \* p1.x - p1.y \* p1.y;  
 e = p2.x \* p2.x + p2.y \* p2.y - p1.x \* p1.x - p1.y \* p1.y;  
 o.x = (a \* f - b \* e) / (2 \* a \* d - 2 \* b \* c);  
 o.y = (d \* e - c \* f) / (2 \* a \* d - 2 \* b \* c);  
 r = dis(o, p1);  
}  
  
int n;  
  
void cal()  
{  
 random\_shuffle(s + 1, s + n + 1);  
 o = s[1];  
 r = 0;  
 for (int i = 2; i <= n; i++) {  
 if (dis(s[i], o) > r + EPS) {  
 o = s[i];  
 r = 0;//第一个点为圆心  
 for (int j = 1; j < i; j++) {  
 if (dis(o, s[j]) > r + EPS) {  
 o.x = (s[i].x + s[j].x) / 2;  
 o.y = (s[i].y + s[j].y) / 2;  
 r = dis(o, s[j]);//第一个点和第二个点中点为圆心，距离为直径  
 for (int k = 1; k < j; k++) {  
 if (dis(o, s[k]) > r + EPS) {  
 getr(s[i], s[j], s[k]);//三点定圆  
 }  
 }  
 }  
 }  
 }  
 }  
}  
  
int main()  
{  
 scanf("%d",&n);  
 rep(i,1,n) {  
 scanf("%lf %lf",&s[i].x,&s[i].y);  
 }  
 cal();  
 printf("%.10f\n%.10f %.10f\n",r,o.x,o.y); // 半径，圆心坐标  
}

## 规律

### 卡特兰数

#### Sequence

1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...

#### Formula

变形：

递推：

#### 基本模型

* 有一个长度为的01序列，其中1,0各个，要求对于任意的整数，数列的前个数中，1的个数不少于0。
* 在n\*n的格子中，只在下三角行走，每次横或竖走一格，有多少中走法？其实向右走相当于进栈，向左走相当于出栈，本质就是n个数出栈次序的问题，所以答案就是卡特兰数。

### 超级卡特兰数（大施罗德数）

#### Sequence

#### Formula  
  
公式$S\_0$：  
$$F\_0=S\_0\\  
2F\_i=S\_i\\  
F\_n=\frac{(6n-3)F\_{n-1}-(n-2)F\_{n-2}}{n+1}$$  
  
$$S\_n=\frac1n\sum^n\_{k=1}2^kC\_n^kC\_n^{k-1},n\geq1$$  
  
#### 基本模型  
超级卡特兰数$S\_n$表示$(0,0)$走到$(n,n)$方案数，只能往右、上、右上走，且移动时始终满足$y\leq x$  
  
### 大乐透数  
  
#### Formula  
  
阶数$m$|公式|例子  
:--:|:--:|:--:  
1|$\frac{n}{1!}$|`1,2,3,4,5,...`  
2|$\frac{n(n+1)}{2!}$|`1,3,6,10,15,...`  
3|$\frac{n(n+1)(n+2)}{3!}$|`1,4,10,20,35,...`  
4|$\frac{n(n+1)(n+2)(n+3)}{4!}$|`1,5,15,35,70,...`  
...|...|...  
m|$$\frac{\prod\_{i=0}^{m-1}(n+i)}{m!}$$|`...`  
  
记$$P(n,m)=\frac{\prod\_{i=0}^{m-1}(n+i)}{m!}$$其中$m$是该公式的阶数，$n$为其数列的第$n$项。  
  
#### 大乐透数1  
  
##### Sequence  
```0,0,0,0,1,5,15,35,70,126,210,330,495,715,1001,1365,...```  
  
##### Formula  
  
公式：$$F\_n = \frac{n(n-1)(n-2)(n-3)}{24}=P(n-3,4)$$  
  
  
#### 大乐透数2  
  
##### Sequence  
  
```0,1,4,10,20,35,56,84,120,165,220,286,364,455,560,680,816,969,1140,...```  
  
##### Formula  
  
公式：$$F\_n = \frac{n(n+1)(n+2)}6=P(n,3)$$  
  
## Matrix  
  
```c++  
/// 矩阵模版 快速幂  
const int mod = 1e9 + 7;  
  
class Matrix  
{  
 const static int MAXN = 10;  
  
 ll a[MAXN][MAXN];  
  
public:  
 int w;  
 int h;  
   
 Matrix(int h,int w):w(w),h(h)  
 {  
 mem(a,0);  
 }  
   
 Matrix(const char format[], ...) {  
   
 va\_list args;  
 w = 0;  
 h = 0;  
   
 char buf[1000];  
   
 va\_start(args, format);  
 vsprintf(buf, format, args);  
 va\_end(args);  
   
 stringstream ss(buf);  
 stringstream num;  
 ll n;  
 string line;  
 while (getline(ss, line)) {  
 num.clear();  
 num << line;  
 w = 0;  
 while (num >> n) {  
 a[h + 1][++ w] = n;  
 }  
 h ++;  
 }  
 }  
   
 void E()  
 {  
 if (w == h) {  
 mem(a,0);  
 rep(i,1,w) a[i][i] = 1;  
 }  
 }  
   
 void print()  
 {  
 int f = 1;  
 rep(i,1,h) {  
 f = 1;  
 rep(j,1,w) {  
 if (f) f = 0;  
 else printf(" ");  
 printf("%lld",a[i][j]);  
 }  
 puts("");  
 }  
 }  
   
 void read\_in()  
 {  
 rep(i,1,h) {  
 rep(j,1,w) {  
 scanf("%lld",&a[i][j]);  
 }  
 }  
 }  
   
 Matrix operator\* (const Matrix &B) const  
 {  
 if (w != B.h) return Matrix(0,0); // invalid  
   
 Matrix ans(h,B.w);  
 rep(i,1,h) {  
 rep(j,1,B.w) {  
 rep(k,1,w) {  
// ans[i][j] = (ans[i][j] + a[i][k] \* B[k][j]); // Not Moduled  
 ans[i][j] = (ans[i][j] + a[i][k] \* B[k][j] % mod + mod) % mod; // Moduled  
 }  
 }  
 }  
 return ans;  
 }  
   
 const ll\* operator[] (int i) const {  
 return a[i];  
 }  
   
 ll\* operator[] (int i) {  
 return a[i];  
 }  
};  
  
Matrix quickpow(Matrix a, ll b)  
{  
 if (a.h != a.w) return Matrix(0,0); // invalid  
   
 Matrix ans(a.h,a.w);  
 rep(i,1,ans.h) ans[i][i] = 1; // Set ans matrix to E  
 while (b)  
 {  
 if (b & 1) ans = a \* ans;  
 a = a \* a;  
 b >>= 1;  
 }  
 return ans;  
}

**sample of use**:

https://www.luogu.com.cn/problem/P1939

/// Sample of build a Matrix to solve num sequent.  
  
int main()  
{  
 Matrix p("1 0 1\n1 0 0\n0 1 0"); // Construct a matrix  
 Matrix base("1\n1\n1");  
 int n;  
 \_\_T {  
 scanf("%d",&n);  
 if (n <= 3) {  
 printf("1\n");  
 continue;  
 }  
 printf("%lld\n",(quickpow(p, n - 3) \* base)[1][1]);  
 }  
 return 0;  
}

# String

## Hash

可以用来求前缀回文串，分别对正串和逆串计算哈希，exkmp也可以做，正反串匹配。

ull hs[MAXN]; // 哈希数组，定义成ull自动取模  
ull bin[MAXN]; // 记录p的次方数  
const int p = 131; // 进制数  
  
string str; // 字符串  
  
void getHash() // str是从0开始的下标  
{  
 bin[0] = 1;  
 for (int i = 1;str[i - 1];i ++) {  
 hs[i] = hs[i - 1] \* p + str[i - 1] - 'a'; // 算哈希值，把他当一个大数字来看待  
 bin[i] = bin[i - 1] \* p; // 计算p的次方数  
 }  
}  
  
ull getSub(int l,int r) // 获取其中的一段subString的哈希值hash[l...r]，l和r均为从1开始的下标  
{  
 return hs[r] - hs[l - 1] \* bin[r - l + 1];  
}

## kmp

nxt(标号从[1,m]，m是匹配串长度)数组意义：nxt[i]是取出从1到i的前缀，对于这个前缀s，存在一个它的子串t，并且t != s，同时t既是s的前缀又是s的后缀，nxt[i] = max(|t|)。

const int MAXN = 1e6 + 10;  
string str;  
string p;  
int n;  
int m;  
  
int nxt[MAXN];  
  
void make()  
{  
 // nxt(标号从[1,m]，m是匹配串长度)数组意义：nxt[i]是取出从1到i的前缀，对于这个前缀s，存在一个它的子串t，并且t != s，同时t既是s的前缀又是s的后缀，nxt[i] = max(|t|)。  
 nxt[0] = -1;  
 nxt[1] = 0;  
 for (int i = 0,k = -1;i < m;)  
 {  
 if (k == -1 || p[i] == p[k]) nxt[++ i] = ++ k;  
 else k = nxt[k];  
 }  
}  
  
int kmp()  
{  
 int ok = 0;  
 make();  
  
 //int f = 0;  
 for (int i = 0,j = 0;i < n;)  
 {  
 if (j == -1 || str[i] == p[j]) {  
 j ++;  
 i ++;  
 if (j == m) {  
 //printf("Found: %lu\n",i - m); // 找到一个p串在str串中的下标  
 ok ++;  
 //f = 1;  
 j = nxt[j];  
 }  
 }  
 else {  
 j = nxt[j];  
 }  
  
 }  
 //if (!f) printf("Not Found!\n");  
 return ok;  
}  
  
int main()  
{  
 cin >> str >> p;  
 n = (int) str.size();  
 m = (int) p.size();  
 printf("%d\n",kmp());  
 return 0;  
}

### exkmp (prefix)

问题定义：给定两个字符串S和T（长度分别为n和m），下标从0开始，定义extend[i]等于S[i]...S[n-1]与T的最长相同前缀的长度，求出所有的extend[i]。举个例子，看下表：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| S | a | a | a | a | a | b | b | b |
| T | a | a | a | a | a | c |  |  |
| extend[i] | 5 | 4 | 3 | 2 | 1 | 0 | 0 | 0 |

如果在S的某个位置i有extend[i]等于m，则可知在S中找到了匹配串T，并且匹配的首位置是i。扩展KMP算法可以找到S中所有T的匹配。

/// 扩展kmp https://segmentfault.com/a/1190000008663857  
const int MAXN = 1e5 + 10;  
  
int nxt[MAXN];  
int extend[MAXN];  
  
string T,S;  
  
// 求解 T 中 next[]，注释参考 exkmp()  
void make()  
{  
 int a = 0, p = 0;  
 int m = (int) T.size();  
 nxt[0] = m;  
  
 for (int i = 1; i < m; i++)  
 {  
 if (i >= p || i + nxt[i - a] >= p)  
 {  
 if (i >= p)  
 p = i;  
  
 while (p < m && T[p] == T[p - i])  
 p++;  
  
 nxt[i] = p - i;  
 a = i;  
 }  
 else  
 nxt[i] = nxt[i - a];  
 }  
}  
  
// 求解 extend[]  
void exkmp()  
{  
 make();  
   
 int a = 0, p = 0;  
 int n = (int) S.size();  
 int m = (int) T.size();  
   
 for (int i = 0; i < n; i++)  
 {  
 if (i >= p || i + nxt[i - a] >= p) // i >= p 的作用：举个典型例子，S 和 T 无一字符相同  
 {  
 if (i >= p) p = i;  
 while (p < n && p - i < m && S[p] == T[p - i]) p ++;  
 extend[i] = p - i;  
 a = i;  
 } else extend[i] = nxt[i - a];  
 }  
}

#### 例题

2021牛客寒假算法基础集训营4 https://ac.nowcoder.com/acm/contest/9984/B  
众所周知，武辰延很喜欢字符串。  
这天，他对着两个字符串和发呆，他发现这两个串的前缀有很多相似的地方，的两个前缀连接起来竟也是的前缀。  
武辰延想知道有多少对的非空前缀连接起来是的前缀。  
形式化地讲，我们把看作字符串长度为的前缀。  
对于一对前缀(允许 i=j)而言，当满足时，我们认为这两个的前缀拼接后等于的一个前缀。  
两对的前缀与不同当且仅当或。

/// 2021牛客寒假算法基础集训营4B  
/// https://ac.nowcoder.com/acm/contest/9984/B  
int main()  
{  
 cin >> T >> S;  
 exkmp();  
 int n = (int) S.size();  
 int k = extend[0];  
 ll ans = 0;  
 int tg = min(k,n-1);  
 rep(i,1,tg)  
 {  
 ans += extend[i];  
 }  
 printf("%lld\n",ans);  
 return 0;  
}

## AC automation

// AC自动机  
// 给定n个模式串s\_i和一个文本串t，求有多少个不同的模式串在文本串里出现过。两个模式串不同当且仅当他们编号不同。  
  
const int MAXN = 5e5 + 10;  
  
struct AC\_Auto {  
 int next[MAXN][26],fail[MAXN],end[MAXN];  
 int root,cnt;  
  
 inline int newNode() {  
 for (int i = 0;i < 26;i ++) next[cnt][i] = -1;  
 end[cnt ++] = 0;  
 return cnt - 1;  
 }  
 void init() {  
 cnt = 0;  
 root = newNode();  
 }  
  
 void insert(char str[]) {  
 int len = (int) strlen(str);  
 int now = root;  
 for (int i = 0;i < len;i++) {  
 if (next[now][str[i] - 'a'] == -1) next[now][str[i] - 'a'] = newNode();  
 now = next[now][str[i] - 'a'];  
 }  
 end[now] ++;  
 }  
  
 void build() {  
 queue<int> q;  
 fail[root] = root;  
 for(int i = 0;i < 26;i++)  
 {  
 if (next[root][i] == -1) next[root][i] = root;  
 else  
 {  
 fail[next[root][i]] = root;  
 q.push(next[root][i]);  
 }  
 }  
 while (!q.empty()) {  
 int now = q.front();  
 q.pop();  
 for (int i = 0;i < 26;i++)  
 {  
 if (next[now][i] == -1) next[now][i] = next[fail[now]][i];  
 else  
 {  
 fail[next[now][i]] = next[fail[now]][i];  
 q.push(next[now][i]);  
 }  
 }  
 }  
 }  
  
 int query(char \*s) {  
 int len = (int) strlen(s);  
 int now = 0,ans = 0;  
 for (int i = 0;i < len;i ++){  
 now = next[now][s[i] - 'a'];  
 for (int t = now;t && ~end[t];t = fail[t]) {  
 ans += end[t];  
 end[t] = -1;  
 }  
 }  
 return ans;  
 }  
  
 void debug() {  
 for (int i = 0;i < cnt;i ++) {  
 printf("id = %3d,fail = %3d,end = %3d,chi = [",i,fail[i],end[i]);  
 for (int j = 0;j < 26;j ++) printf("%2d",next[i][j]);  
 printf("]\n");  
 }  
 }  
};  
  
AC\_Auto ac;  
char str[MAXN << 1];  
  
int main()  
{  
 int n;  
 \_\_T {  
 scanf("%d",&n);  
 ac.init();  
 rep(i,1,n) {  
 scanf("%s",str);  
 ac.insert(str);  
 }  
 ac.build();  
 scanf("%s",str);  
 printf("%d\n",ac.query(str));  
 }  
 return 0;  
}

## Manacher

### 最长回文串

string Manacher(string &s)  
{  
 //改造字符串  
 int n = (int) s.size();  
 string res = "$#";  
 for (int i = 0;i < n;i ++)  
 {  
 res += s[i];  
 res += "#";  
 }  
  
 //数组  
 n = (int) res.size();  
 vector<int> P(n,0);  
 int mi = 0,right = 0; //mi为最大回文串对应的中心点，right为该回文串能达到的最右端的值  
 int maxLen = 0,maxPoint = 0; //maxLen为最大回文串的长度，maxPoint为记录中心点  
 for (int i = 1;i < n;i ++)  
 {  
 P[i] = right > i ? min(P[2 \* mi - i],right - i) : 1; //关键句，文中对这句以详细讲解  
 while (res[i + P[i]] == res[i - P[i]]) {  
 P[i] ++;  
 }  
 if (right < i + P[i]) //超过之前的最右端，则改变中心点和对应的最右端  
 {  
 right = i + P[i];  
 mi = i;  
 }  
 if (maxLen < P[i]) //更新最大回文串的长度，并记下此时的点  
 {  
 maxLen = P[i];  
 maxPoint = i;  
 }  
 }  
 return s.substr((maxPoint - maxLen) / 2,maxLen - 1);  
}

### 回文串个数

ll Manacher\_n(string &s)  
{  
 //改造字符串  
 int n = (int) s.size();  
 string res = "$#";  
 for (int i = 0;i < n;i ++)  
 {  
 res += s[i];  
 res += "#";  
 }  
  
 //数组  
 n = (int) res.size();  
 vector<ll> P(n,0);  
 ll mi = 0,right = 0; //mi为最大回文串对应的中心点，right为该回文串能达到的最右端的值  
 ll maxLen = 0,maxPoint = 0; //maxLen为最大回文串的长度，maxPoint为记录中心点  
   
 ll ans = 0;  
 for (ll i = 1;i < n;i ++)  
 {  
 P[i] = right > i ? min(P[2 \* mi - i],right - i) : 1; //关键句，文中对这句以详细讲解  
 while (res[i + P[i]] == res[i - P[i]]) {  
 P[i] ++;  
 }  
 if (right < i + P[i]) //超过之前的最右端，则改变中心点和对应的最右端  
 {  
 right = i + P[i];  
 mi = i;  
 }  
 if (maxLen < P[i]) //更新最大回文串的长度，并记下此时的点  
 {  
 maxLen = P[i];  
 maxPoint = i;  
 }  
 ans += P[i] / 2;  
 }  
 return ans;  
}

## 表达式求值

struct Node {  
 int type; // 0 - num,1 - op  
 char c;  
 ll num;  
};  
  
int priority[130];  
  
ll quickpow(ll a, ll b)  
{  
 ll ans = 1;  
 while (b)  
 {  
 if (b & 1) ans = a \* ans;  
 a = a \* a;  
 b >>= 1;  
 }  
 return ans;  
}  
  
void getPost(string &str,vector<Node> &post) {  
 post.clear();  
 stack<char> op\_stk;  
 ll num = 0;  
 int has\_num = 0;  
   
 for (int i = 0;str[i];i ++) {  
 if (str[i] >= '0' && str[i] <= '9') {  
 num = num \* 10 + (str[i] ^ 48);  
 has\_num = 1;  
 } else {  
 if (has\_num) post.push\_back({0,0,num});  
 has\_num = num = 0;  
 if (str[i] != ')') {  
 if (str[i] != '(') while (!op\_stk.empty() && priority[op\_stk.top()] >= priority[str[i]]) {  
 post.push\_back({1,op\_stk.top(),0});  
 op\_stk.pop();  
 }  
 op\_stk.push(str[i]);  
 } else {  
 while (op\_stk.top() != '(') {  
 post.push\_back({1,op\_stk.top(),0});  
 op\_stk.pop();  
 }  
 op\_stk.pop();  
 }  
 }  
 }  
 if (has\_num) post.push\_back({0,0,num});  
 while (!op\_stk.empty()) {  
 post.push\_back({1,op\_stk.top(),0});  
 op\_stk.pop();  
 }  
}  
  
ll getAns(vector<Node> &post) {  
 ll ans = 0;  
 stack<ll> num\_stk;  
   
 for (auto i : post) {  
 if (!i.type) num\_stk.push(i.num);  
 else {  
 switch (i.c) {  
 case '+':  
 ans = num\_stk.top();  
 num\_stk.pop();  
 ans += num\_stk.top();  
 num\_stk.pop();  
 num\_stk.push(ans);  
 break;  
 case '-':  
 ans = num\_stk.top();  
 num\_stk.pop();  
 ans = num\_stk.top() - ans;  
 num\_stk.pop();  
 num\_stk.push(ans);  
 break;  
 case '\*':  
 ans = num\_stk.top();  
 num\_stk.pop();  
 ans \*= num\_stk.top();  
 num\_stk.pop();  
 num\_stk.push(ans);  
 break;  
 case '/':  
 ans = num\_stk.top();  
 num\_stk.pop();  
 ans = num\_stk.top() / ans;  
 num\_stk.pop();  
 num\_stk.push(ans);  
 break;  
 case '^':  
 ans = num\_stk.top();  
 num\_stk.pop();  
 ans = quickpow(num\_stk.top(),ans);  
 num\_stk.pop();  
 num\_stk.push(ans);  
 break;  
 default:  
 break;  
 }  
 }  
 }  
 return num\_stk.top();  
}  
  
int main() {  
 priority['('] = 1;  
 priority['^'] = 4;  
 priority['\*'] = priority['/'] = 3;  
 priority['+'] = priority['-'] = 2;  
   
 string str;  
 cin >> str;  
 vector<Node> post;  
   
 getPost(str,post);  
 printf("%lld\n",getAns(post));  
}  
  
/\*  
1+(3+2)\*(7^2+6\*9)/(2)  
\*/  
  
/\*  
258  
\*/

# Graphics

## Forward Star

### No weight

const int MAXN = 1e5 + 10;  
  
struct Edge {  
 int to;  
 int nxt;  
} e[MAXN \* 2];  
int g[MAXN]; // Please call init() to memset it to -1!  
int cnt = 0;  
  
void init(int n)  
{  
 cnt = 0;  
 memn(g,-1,int,n);  
}  
  
void add\_edge(int u,int v)  
{  
 e[cnt] = {v,g[u]};  
 g[u] = cnt ++;  
}

### With weight

const int MAXN = 1e5 + 10;  
  
struct Edge {  
 int to;  
 ll w;  
 int nxt;  
} e[MAXN \* 2];  
int g[MAXN]; // Please call init() to memset it to -1!  
int cnt = 0;  
  
void init(int n)  
{  
 cnt = 0;  
 memn(g,-1,int,n);  
}  
  
void add\_edge(int u,int v,ll w)  
{  
 e[cnt] = {v,w,g[u]};  
 g[u] = cnt ++;  
}

## Dijkstra

### Build Graphic from Vector

const int MAXN = 1e5 + 10;  
  
int dis[MAXN];  
int vis[MAXN];  
  
struct ST {  
 int n;  
 int w;  
   
 bool operator< (const ST &other) const {  
 return w > other.w;  
 }  
};  
  
vector<ST> g[MAXN];  
  
void dij(int s) {  
 mem(dis,-1);  
 mem(vis,0);  
  
 priority\_queue<ST> q;  
 q.push({s,dis[s] = 0});  
  
 ST current;  
 int k;  
 while (!q.empty()) {  
 current = q.top();  
 q.pop();  
 if (vis[current.n]) continue;  
 vis[current.n] = 1;  
  
 for (auto to : g[current.n]) {  
 k = current.w + to.w;  
 if (dis[to.n] == -1 || dis[to.n] > k) {  
 q.push({to.n,dis[to.n] = k});  
 }  
 }  
 }  
}

### Build Graphic from Forward Star

const int MAXN = 1e5 + 10;  
  
struct Edge {  
 int to;  
 ll w;  
 int nxt;  
} e[MAXN \* 2];  
int g[MAXN]; // Please call init() to memset it to -1!  
int cnt = 0;  
  
void init(int n)  
{  
 cnt = 0;  
 memn(g,-1,int,n);  
}  
  
void add\_edge(int u,int v,ll w)  
{  
 e[cnt] = {v,w,g[u]};  
 g[u] = cnt ++;  
}  
  
  
ll dis[MAXN];  
int vis[MAXN];  
  
struct Node {  
 int n;  
 ll w;  
  
 bool operator< (const Node &other) const {  
 return w > other.w;  
 }  
};  
  
void dij(int s) {  
 mem(dis,-1);  
 mem(vis,0);  
  
 priority\_queue<Node> q;  
 q.push({s,dis[s] = 0});  
  
 Node current;  
 ll k;  
 int to;  
 while (!q.empty()) {  
 current = q.top();  
 q.pop();  
 if (vis[current.n]) continue;  
 vis[current.n] = 1;  
  
 for (int i = g[current.n];~i;i = e[i].nxt) {  
 to = e[i].to;  
 k = dis[current.n] + e[i].w;  
 if (dis[to] == -1 || dis[to] > k) {  
 q.push({to,dis[to] = k});  
 }  
 }  
 }  
}  
  
int main()  
{  
 int n,m;  
 int u,v;  
 ll w;  
 int s,t;  
 \_\_T {  
 scanf("%d%d",&n,&m);  
 init(n + 5);  
 while (m --) {  
 scanf("%d%d%lld",&u,&v,&w);  
 add\_edge(u,v,w);  
 add\_edge(v,u,w);  
 }  
 scanf("%d%d",&s,&t);  
 dij(s);  
 printf("%lld\n",dis[t]);  
 }  
 return 0;  
}

### dij变式，多状态

const int MAXN = 1e5 + 10;  
  
struct ST {  
 int n;  
 double w;  
 int s;  
  
 bool operator< (const ST &other) const {  
 return w > other.w;  
 }  
};  
  
double dis[MAXN][3];  
int vis[MAXN][3];  
vector<ST> g[MAXN];  
  
void dij(int s) {  
 rep(i,1,MAXN) dis[i][0] = dis[i][1] = dis[i][2] = -1;  
 mem(vis,0);  
   
   
 priority\_queue<ST> q;  
 q.push({s,dis[s][0] = 0,0});  
   
 inr step;  
   
 ST current;  
 double k,cost;  
 while (!q.empty()) {  
 current = q.top();  
 q.pop();  
   
 if (vis[current.n][current.s]) continue;  
 vis[current.n][current.s] = 1;  
   
 for (auto to : g[current.n]) {  
   
 cost = to.w;  
  
 // update costs from step(mode)  
 //  
 // REP(i,0,current.s % 3) {  
 // cost = 1 / (1 - cost);  
 // }  
 // cost = fabs(cost);  
   
 // update step  
 // step = (current.s + 1) % 3;  
   
 k = dis[current.n][current.s] + cost;  
 if (dis[to.n][step] == -1 || dis[to.n][step] > k) {  
 q.push({to.n,dis[to.n][step] = k,step});  
 }  
 }  
 }  
}

## 定理

### Havel-Hakimi定理（度数序列判断是否能组成 无向简单图）

给定度数序列

首先降序排序得

然后删除第一个元素，在中对之后个元素进行-1。其实就是贪心地认为后面几个点都与这个点相连。

删完再排序，继续删。。。

只要一直这么删下去，序列中不出现-1就是能够成简单无向图。

**POJ1659**

struct Node {  
 int n;  
 int num;  
} d[100];  
  
int cmp(const Node &a,const Node &b) {  
 return a.num > b.num;  
}  
  
int main()  
{  
 int n;  
 int ok;  
 int ans[15][15];  
 \_\_T {  
 sci(n);  
 rep(i,1,n) {  
 sci(d[i].num);  
 d[i].n = i;  
 }  
 memset(ans,0,sizeof ans);  
 ok = 1;  
 rep(i,1,n) { // 要全部变成0，所以最后一个也要爬  
 sort(d + i,d + 1 + n,cmp); // 先排序（降序）  
 if (i + d[i].num > n) { // 如果已经超过了肯定也不行了  
 ok = 0;  
 break;  
 }  
 rep(j,i + 1,i + d[i].num) {  
 ans[d[i].n][d[j].n] = 1;  
 ans[d[j].n][d[i].n] = 1;  
 d[j].num --; // 开始减  
 if (d[j].num < 0) {  
 ok = 0;  
 goto end;  
 }  
 }  
 }  
 end:  
 if (ok) {  
 puts("YES");  
 rep(i,1,n) {  
 rep(j,1,n) {  
 if (j != 1) printf(" ");  
 printf("%d",ans[i][j]);  
 }  
 puts("");  
 }  
 } else puts("NO");  
 puts("");  
 }  
}

### Erdős–Gallai theorem(是否构成无向图，只能判断，不能建图)：

一个度数序列可以构成无向图，当且仅当将降序排序之后：

## LCA

### Tarjan 离线算法

### dfs+ST倍增 在线算法

## dfs序

可以将**树**线段化。

const int MAXN = 1e5 + 10;  
  
int dfn = 0;  
int in[MAXN],out[MAXN];  
  
void dfs(int k,int f)  
{  
 in[k] = ++ dfn;  
 for (int i = g[k];~i;i = e[i].nxt)  
 {  
 if (e[i].to != f) {  
 dfs(e[i].to,k);  
 }  
 }  
 out[k] = dfn;  
}

## 求树的重心

求法：删掉某一个点i，求剩余连通块的节点个数，求出一个w[i]=max(每个连通块的节点个数)，那么所有点中最小的w[i]，那么这个点i便是树的重心

性质： 1. 树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两个距离和，他们的距离和一样。 2. 把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。 3. 一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。 4. 一棵树最多有两个重心，且相邻

int num,m;  
  
vector<int> g[50010];  
int vis[50010] = {0};  
//int ch[50010] = {0};  
int w[50010] = {0};  
  
int gravity(int n)  
{  
 if (vis[n]) return 0;  
 vis[n] = 1;  
 int c = 0;  
 int x = -1;  
 int t;  
 for (auto i : g[n]) {  
 c += (t = gravity(i));  
 x = max(t,x);  
 }  
 x = max(x,num - c - 1);  
// ch[n] = c;  
 w[n] = x;  
 return c + 1;  
}  
  
int main()  
{  
   
 scanf("%d%d",&num,&m);  
 int u,v;  
 while (m --) {  
 scanf("%d%d",&u,&v);  
 g[u].pb(v);  
 g[v].pb(u);  
 }  
   
 gravity(1);  
// Tprint(ch,1,num + 1);  
// puts("");  
// Tprint(w,1,num + 1);  
// puts("");  
 int m = INT\_INF;  
 rep(i,1,num) m = min(m,w[i]);  
 int f = 1;  
 rep(i,1,num) {  
 if (m == w[i]) {  
 if (f) f = 0;  
 else printf(" ");  
 printf("%d",i);  
 }  
 }  
 puts("");  
   
}

## DSU on Tree 启发式树上合并

cf gym102431K: 统计一个子树中，节点标号连续的段数

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// \* \*  
// \* 启发式树上合并：用于统计树上子树中特定节点的个数 \*  
// \* \*  
// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
const int MAXN = 200010;  
  
vector<int> g[MAXN];  
  
int n;  
  
// -----统计重儿子（节点数最多的儿子）-----  
int weight[MAXN];  
  
int getMaxSon(int x,int f)  
{  
 int mx = -1;  
 int nmx = -1;  
 int t;  
 int ch = 1;  
 for (auto i : g[x]) {  
 if (i == f) continue;  
 t = getMaxSon(i,x);  
 if (t > mx) {  
 mx = t;  
 nmx = i;  
 }  
 ch += t;  
 }  
 weight[x] = nmx;  
 return ch;  
}  
// ----------------------------------  
  
  
int vis[MAXN];  
int ans[MAXN];  
int heavy;  
  
int count(int x,int f,int type) // type = 1统计，type = -1清空  
{  
 int t = 0;  
   
 // ---------------在此修改计数方式---------------  
 if (vis[x - 1] && vis[x + 1]) t --;  
 else if (!vis[x - 1] && !vis[x + 1]) t ++;  
 // -------------------------------------------  
   
 // -----根据题意改变vis什么，若是统计颜色的则vis[color[x]] += type-----  
 vis[x] += type;  
 // -------------------------------------------------------------  
   
 for (auto i : g[x]) {  
 if (i == f) continue;  
 if (i != heavy) // 跳过重儿子  
 t += count(i,x,type);  
 }  
 return t;  
}  
  
int dfs(int x,int f,int keep) // keep 是否保留，调用直接(root,-1,1)  
{  
 int t = 0;  
 for (auto i : g[x]) {  
 if (f == i) continue;  
 if (weight[x] != i) {  
 dfs(i,x,0); // 遍历轻儿子，不保留  
 }  
 }  
 if (weight[x] != -1) {  
 t += dfs(weight[x],x,1); // 遍历重儿子，保留  
 heavy = weight[x]; // 记录当前重儿子  
 }  
   
 t += count(x, f, 1); // 计算答案  
 heavy = -1; // 清空重儿子记录  
 ans[x] = t; // 记录答案  
  
 if (!keep) count(x, f, -1); // 不保留直接清除vis  
 return t;  
}  
  
int main()  
{  
 int u,v;  
 int T = 1;  
 \_\_T {  
 sci(n);  
 rep(i,1,n + 5) {  
 g[i].clear();  
 ans[i] = vis[i] = 0; // 注意清空  
 }  
 heavy = -1; // 注意清空  
 rep(i,1,n - 1) {  
 scii(u,v);  
 g[u].pb(v);  
 g[v].pb(u);  
 }  
 getMaxSon(1, -1);  
 dfs(1, -1, 1);  
 printf("Case #%d:",T ++);  
 rep(i,1,n) printf(" %d",ans[i]);  
 puts("");  
 }  
 return 0;  
}  
/\*  
1  
7  
1 2  
2 4  
2 6  
1 3  
3 5  
3 7  
\*/  
  
/\*  
Case #1: 1 3 3 1 1 1 1  
\*/

## 求树上一条边两边点的个数

const int MAXN = 1e5 + 10;  
  
int all = 0;  
int nn;  
  
ll ch[MAXN];  
ll a[MAXN];  
  
struct Edge {  
 int to;  
 int nxt;  
} e[MAXN \* 2];  
int g[MAXN]; // Please memset it to -1!!!!!!!!  
int cnt = 0;  
  
void init(int n)  
{  
 cnt = 0;  
 memn(g,-1,int,n);  
}  
  
void add\_edge(int u,int v)  
{  
 e[cnt] = {v,g[u]};  
 g[u] = cnt ++;  
}  
  
  
// 方法：记录每个节点的子节点数，在用总结点数减去该边子节点数即可  
void dfs(int n, int f)  
{  
 ch[n] ++;  
   
 int to;  
 for (int i = g[n];~i;i = e[i].nxt) {  
 to = e[i].to;  
 if (to == f) continue;  
 dfs(to, n);  
 ch[n] += ch[to];  
 }  
   
 for (int i = g[n];~i;i = e[i].nxt) {  
 to = e[i].to;  
 if (to == f) continue;  
 a[all ++] = ch[to] \* (nn - ch[to]);  
 }  
   
}  
  
int main()  
{  
 int u,v;  
 scanf("%d",&nn);  
 init(nn + 5);  
 rep(i,1,nn - 1) {  
 scanf("%d%d",&u,&v);  
 add\_edge(u,v);  
 add\_edge(v,u);  
 }  
 mem(ch,0);  
 dfs(1,-1);  
 sort(a, a + all);  
 nn --;  
 ll sum = 0;  
 int i = 0;  
 while (nn) {  
 sum += (nn --) \* a[i ++];  
 }  
 printf("%lld\n",sum);  
 return 0;  
}

## Net Flow

### Edmond-Karp

/// 网络流-最大流模板 EK算法 前向星  
/// https://www.luogu.com.cn/problem/P3376  
const int MAXN = 5000 + 10;  
  
struct Edge {  
 int to;  
 ll w;  
 int nxt;  
} e[MAXN \* 2];  
int g[MAXN];  
int cnt = 0;  
  
void init(int n)  
{  
 cnt = 0;  
 memn(g,-1,int,n);  
}  
  
void add\_edge(int u,int v,ll w)  
{  
 e[cnt] = {v,w,g[u]};  
 g[u] = cnt ++;  
}  
  
  
int pre[MAXN]; // record the previous node index and indicating a node whether has been visited.  
  
ll flow[MAXN];  
int n,m;  
  
ll getAgtPath(int s,int t) // method to find an augmented path  
{  
 // bfs  
 queue<int> q;  
 q.push(s);  
   
 int current;  
 memn(pre,-1,int,n + 5);  
   
 flow[s] = INT\_INF;  
 pre[s] = 0; // sourse node has been visited  
   
 int to;  
 while (!q.empty()) {  
 current = q.front();  
 q.pop();  
   
 if (current == t) break;  
   
 for (int i = g[current];~i;i = e[i].nxt) {  
 to = e[i].to;  
 if (pre[to] != -1 || e[i].w <= 0) continue;  
   
 pre[to] = i; // record the index of current edge  
   
 flow[to] = min(flow[current],e[i].w);  
 q.push(to);  
 }  
 }  
   
 if (pre[t] != -1) return flow[t]; // return the terminal's flow  
 return -1; // 404 not found  
}  
  
ll EKmaxFlow(int s,int t) // source -> terminal  
{  
 if (s == t) return INT\_INF;  
   
   
 ll flow = 0;  
 ll agt\_flow;  
 int cur;  
 int edge\_idx;  
   
 while ((agt\_flow = getAgtPath(s, t)) != -1) { // find augmented path until all flows are gone.  
   
 // modify the w of path  
 cur = t;  
 while (cur != s) {  
 edge\_idx = pre[cur];  
   
 e[edge\_idx].w -= agt\_flow;  
 e[edge\_idx ^ 1].w += agt\_flow; // xor can get the reverse edge swiftly  
   
 cur = e[edge\_idx ^ 1].to;  
 }  
   
 flow += agt\_flow; // add to ans  
 }  
  
 return flow;  
}  
  
int vis[210][210];  
  
int main()  
{  
 int s,t;  
 scanf("%d%d%d%d",&n,&m,&s,&t);  
 init(n + 5);  
 mem(vis,-1);  
 int u,v,w;  
 rep(i,1,m) {  
 scanf("%d%d%d",&u,&v,&w);  
 if (vis[u][v] == -1) { // record the vised edge  
 add\_edge(u, v, w);  
 add\_edge(v, u, 0); // add reversed edge, which w=0;  
 vis[u][v] = cnt - 2;  
 }  
 else {  
 e[vis[u][v]].w += w; // increase the weight  
 }  
 }  
 printf("%lld\n",EKmaxFlow(s, t));  
 return 0;  
}

### Dinic

/// 网络流-最大流模板 Dinic算法 前向星  
/// https://www.luogu.com.cn/problem/P3376  
const int MAXN = 500010;  
  
struct Edge {  
 int to;  
 ll w;  
 int nxt;  
} e[MAXN << 1];  
int g[MAXN]; // Please call init() to memset it to -1!  
int cnt = 0;  
  
void init(int n)  
{  
 cnt = 0;  
 memset(g, -1, sizeof(int) \* n);  
}  
  
void add\_edge(int u,int v,ll w)  
{  
 e[cnt] = {v,w,g[u]};  
 g[u] = cnt ++;  
}  
  
int s,t,n,m;  
int dis[MAXN];  
int cur[MAXN]; // 替代g数组，记住上次dfs最后跑到的地方，优化，减少dfs的跑的次数  
  
int bfs()  
{  
 memset(dis, -1, sizeof(int) \* (n + 5));  
  
 queue<int> q;  
 q.push(s);  
 dis[s] = 0;  
  
 int to,current,k;  
  
 while (!q.empty()) {  
 current = q.front();  
 q.pop();  
  
 for (int i = g[current];~i;i = e[i].nxt) {  
 to = e[i].to;  
 k = dis[current] + 1;  
 if (dis[to] == -1 && e[i].w > 0) { // 只有没有访问过的，且该通路可以走(w > 0)  
 dis[to] = k;  
 if (to == t) return 1;  
 q.push(to);  
 }  
 }  
 }  
  
 return 0;  
}  
  
ll dfs(int node,ll flow)  
{  
 if (node == t) return flow;  
 int to;  
 ll d;  
 for (int &i = cur[node];~i;i = e[i].nxt) { // 改变i的同时，cur[node]的值也会被改变  
 to = e[i].to;  
 if (dis[node] + 1 == dis[to] && e[i].w > 0) {  
 d = dfs(to,min(e[i].w,flow));  
 if (d > 0) {  
 e[i].w -= d;  
 e[i ^ 1].w += d;  
 return d;  
 }  
 }  
 }  
 return 0;  
}  
  
ll dinic()  
{  
 ll ans = 0;  
 ll d;  
 while (bfs()) {  
 rep(i,1,n) cur[i] = g[i];  
 while ((d = dfs(s,INT\_INF)))  
 ans += d;  
 }  
 return ans;  
}  
  
int main()  
{  
 // 需要用到的变量有n（n个点），s（源点），t（汇点）  
 scanf("%d%d%d%d",&n,&m,&s,&t);  
 int u,v;  
 ll w;  
 init(n + 5);  
 while (m --) {  
 scanf("%d%d%lld",&u,&v,&w);  
 add\_edge(u, v, w);  
 add\_edge(v, u, 0);  
 }  
 printf("%lld\n",dinic());  
 return 0;  
}

### 二分图最大匹配

最大匹配：选取k条边，并且一个点只能被选到一次，k最大。

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// \* 二分图匹配 - 网络流算法 \*  
// \* 思路：建立超级源点到各个左部点的联系（权重为1）， \*  
// \* 建立右部点到超级汇点的联系，以及左右部分点之间的联系，\*  
// \* 最后直接最大流板子。 \*  
// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
int main()  
{  
 // \*\*注意：MAXN尽量开大点，不然很容易RE\*\*  
 int nn,mm,e;  
 scanf("%d%d%d",&nn,&mm,&e);  
 init(nn + mm + 10);  
 int u,v;  
 // nn + mm + 1 源点  
 // nn + mm + 2 汇点  
 while (e --) {  
 scanf("%d%d",&u,&v);  
 add\_edge(u, v + nn, 1); // 建立 左部点 到 右部点 的联系  
 add\_edge(v + nn, u, 0);  
 }  
 rep(i,1,nn) {  
 add\_edge(nn + mm + 1, i, 1); // 建立超级源点到各个左部点的联系  
 add\_edge(i, nn + mm + 1, 0);  
 }  
   
 rep(i,nn + 1,nn + mm) {  
 add\_edge(i, nn + mm + 2, 1); // 建立 右部点 到超级汇点的联系  
 add\_edge(nn + mm + 2, i, 0);  
 }  
   
 n = nn + mm + 2; // 设置总点量  
 s = nn + mm + 1; // 设置起点  
 t = nn + mm + 2; // 设置终点  
 printf("%lld\n",dinic());  
 return 0;  
}

## 一般图最大匹配(带花树)

const int MAXN = 50010; // 开比边大  
  
int n,m,father[MAXN],vst[MAXN],match[MAXN],pre[MAXN],type[MAXN];  
  
vector<int> g[MAXN];  
queue<int> q;  
  
void add\_edge(int u,int v) {  
 g[u].push\_back(v);  
}  
  
int LCA(int x,int y) {  
 static int times = 0;  
 times ++;  
 x = father[x];  
 y = father[y]; //已知环位置  
 while (vst[x] != times) {  
 if (x) {  
 vst[x] = times;  
 x = father[pre[match[x]]];  
 }  
 swap(x,y);  
 }  
 return x;  
}  
  
void blossom(int x,int y,int lca) {  
 while (father[x] != lca) {  
 pre[x] = y;  
 y = match[x];  
 if (type[y] == 1) {  
 type[y] = 0;  
 q.push(y);  
 }  
 father[x] = father[y] = father[lca];  
 x = pre[y];  
 }  
}  
  
int augument(int s) {  
 for (int i = 1; i <= n;i ++) father[i]=i;  
 memset(type,-1,sizeof(type));  
   
 q = queue<int>();  
 type[s] = 0;  
 q.push(s); //仅入队o型点  
   
 int cur;  
 while (!q.empty()) {  
 cur = q.front();  
 q.pop();  
 for (int next : g[cur]) {  
 if (type[next] == -1) {  
 pre[next] = cur;  
 type[next] = 1; //标记为i型点  
 if(!match[next]) {  
 for(int to = next,from = cur; to; from = pre[to]) {  
 match[to] = from;  
 swap(match[from],to);  
 }  
 return true;  
 }  
 type[match[next]] = 0;  
 q.push(match[next]);  
 } else if (type[next] == 0 && father[cur] != father[next]) {  
 int lca = LCA(cur,next);  
 blossom(cur,next,lca);  
 blossom(next,cur,lca);  
 }  
 }  
 }  
 return false;  
}  
  
int matching() {  
 int ans = 0;  
 for(int i=n; i>=1; i--)  
 if(!match[i]) ans += augument(i);  
 return ans;  
}  
  
int main() {  
 scii(n,m);  
 int u,v;  
 while (m --) {  
 scii(u,v);  
 add\_edge(u, v);  
 add\_edge(v, u);  
 }  
 int ans = matching();  
 printf("%d\n",ans);  
 for (int i = 1; i <= n;i ++) {  
 if (i != 1) printf(" ");  
 printf("%d",match[i]); // 表示 i号点 和 match[i]号点 相匹配  
 }  
 return 0;  
}

# 基础

## vector操作

**逆向迭代器删除**

vector<int> v;  
  
for (auto i = v.rbegin();i != v.rend();) {  
 v.erase((++ i).base()); // 逆向迭代器删除  
}

**正向迭代器删除**

vector<int> v;  
  
for (auto i = v.begin();i != v.end();) {  
 i = v.erase(i); // 正向迭代器删除  
}