[1.基础算法 2](#_Toc21078)

[1.1快读快写 2](#_Toc23173)

[1.2二分 2](#_Toc29668)

[1.3前缀和 3](#_Toc6196)

[1.4离散化 4](#_Toc1900)

[1.5高精度 5](#_Toc23798)

[2. 数据结构 10](#_Toc2995)

[2.1单调栈 10](#_Toc26266)

[2.2单调队列 10](#_Toc255)

[2.3 ST表 11](#_Toc20279)

[2.4树状数组 11](#_Toc28742)

[2.5线段树 12](#_Toc32291)

[2.5.1线段树结构 12](#_Toc1246)

[2.5.2线段树优化建图 13](#_Toc5479)

[2.5.3维护GCD 18](#_Toc5261)

[2.5.4区间第K小 20](#_Toc26636)

[2.6 LCA 21](#_Toc4390)

[2.7莫队 24](#_Toc10179)

[2.8树链剖分（重链） 28](#_Toc30751)

[2.9 dsu on tree 32](#_Toc11427)

[3. 数论 33](#_Toc14272)

[3.1质数 34](#_Toc4124)

[3.2约数 35](#_Toc32475)

[3.3组合数 36](#_Toc23639)

[3.3欧拉函数 38](#_Toc21291)

[4. 动态规划 40](#_Toc23433)

[4.1背包问题 40](#_Toc27599)

[4.1.1 01背包 40](#_Toc12049)

[4.1.2完全背包 41](#_Toc30844)

[4.1.3多重背包 42](#_Toc21593)

[4.1.4分组背包 43](#_Toc857)

[5. 图论 44](#_Toc11719)

[5.1最短路 44](#_Toc19107)

[5.2拓扑排序 47](#_Toc6219)

[5.3二分图 48](#_Toc6283)

[5.4网络流 53](#_Toc6734)

[5.4.1最大流dinic 53](#_Toc4847)

[5.4.2最小费用最大流 54](#_Toc24322)

## 1.基础算法

### 1.1快读快写

template <typename T> inline void read(T& t) {

int f = 0, c = getchar(); t = 0;

while (!isdigit(c)) f |= c == '-', c = getchar();

while (isdigit(c)) t = t \* 10 + c - 48, c = getchar();

if (f) t = -t;

}

template <typename T> void print(T x) {

if (x < 0) x = -x, putchar('-');

if (x > 9) print(x / 10);

putchar(x % 10 + 48);

}

### 1.2二分、三分

区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用

int bsearch\_1(int l, int r)

{

while (l < r)

{

int mid = l + r >> 1;

if (check(mid)) r = mid;

else l = mid + 1;

}

return l;

}

区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用

int bsearch\_2(int l, int r)

{

while (l < r)

{

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

if (check(mid)) l = mid;

else r = mid - 1;

}

return l;

}

double bsearch\_3(double l, double r)

{

const double eps = 1e-6; eps 表示精度，取决于题目对精度的要求

while (r - l > eps)

{

double mid = (l + r) / 2;

if (check(mid)) r = mid;

else l = mid;

}

return l;

}

三分法

double l, r;

const double eps

while(r - l > eps) {

double mid1 = l + (r - l) / 3.0;

double mid2 = r - (r - l) / 3.0;

double fx = check(mid1), fy = check(mid2);

if(fx > fy) l = mid1;

else r = mid2;

}

## 1.3前缀和

一维前缀和：快速查询区间和

s[i] = s[i-1]+a[i]

a[l] + ... + a[r] = s[r] - s[l - 1]

二维前缀和：子矩阵和

s[i, j] = 第i行j列格子左上部分所有元素的和

以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：

s[x2, y2] - s[x1 - 1, y2] - s[x2, y1 - 1] + s[x1 - 1, y1 - 1]

s[i]处理：

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

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

s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];

}

}

一维差分

区间[l, r]中的每个数加上c：b[l] += c, b[r + 1] -= c

前缀和的逆运算就是差分

void insert(int l, int r, int x) {

b[l] += x;

b[r + 1] -= x;

}

二维差分

以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵中的所有元素加上c：

s[x1, y1] += c, s[x2 + 1, y1] -= c, s[x1, y2 + 1] -= c, s[x2 + 1, y2 + 1] += c

void insert(int x1, int y1, int x2, int y2, int c) {

b[x1][y1] += c;

b[x2 + 1][y1] -= c;

b[x1][y2 + 1] -= c;

b[x2 + 1][y2 + 1] += c;

}

差分处理原数组

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

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

insert(i, j, i, j, a[i][j]);

}

}

还原

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

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

b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1];

}

}

### 1.4离散化

/\*

\*第一种方式：stl函数（绝对的大小关系）

\*n 原数组大小

\*num 原数组中的元素 lsh 离散化的数组 cnt 离散化后的数组大小

\*/

int lsh[MAXN], cnt, num[MAXN], n;

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

scanf("%d", &num[i]);

lsh[i] = num[i];

}

sort(lsh + 1, lsh + n + 1);

cnt = unique(lsh + 1, lsh + n + 1) - lsh - 1;

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

num[i] = lower\_bound(lsh + 1, lsh + cnt + 1, num[i]) - lsh;

/\*

\*第二种方式：记录下标，排序后放回原数组

\*/

#include<algorithm>

struct Node {

int data, id;

bool operator < (const Node &a) const {

return data < a.data;

}

};

Node num[MAXN];

int rank[MAXN], n;

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

scanf("%d", &num[i].data);

num[i].id = i;

}

sort(num + 1, num + n + 1);

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

rank[num[i].id] = i;

## 1.5高精度

/\*

\*加法

\*/

#include <bits/stdc++.h>

using namespace std;

vector<int> add(vector<int> &a, vector<int> &b) {

vector<int> c;

int t = 0;

for(int i = 0; i < a.size() || i < b.size(); i++) {

if(i < a.size()) t += a[i];

if(i < b.size()) t += b[i];

c.push\_back(t % 10);

t /= 10;

}

if(t) c.push\_back(1);

return c;

}

int main() {

vector<int> A, B;

string a, b;

cin >> a >> b;

for(int i = a.size() - 1; i >= 0; i--) A.push\_back(a[i] - '0');

for(int i = b.size() - 1; i >= 0; i--) B.push\_back(b[i] - '0');

auto ok = add(A, B);

for(int i = ok.size() - 1; i >= 0; i--) {

cout << ok[i];

}

return 0;

}

/\*

\*减法C = A - B, 满足A >= B, A >= 0, B >= 0

\*/

bool cmp(vector<int> &a, vector<int> &b) {

if(a.size() != b.size()) return a.size() > b.size();

for(int i = a.size() - 1; i >= 0; i--) {

if(a[i] != b[i]) return a[i] > b[i];

}

return true;

}

vector<int> sub(vector<int> &A, vector<int> &B) {

vector<int> C;

for (int i = 0, t = 0; i < A.size(); i ++ ) {

t = A[i] - t;

if (i < B.size()) t -= B[i];

C.push\_back((t + 10) % 10);

if (t < 0) t = 1;

else t = 0;

}

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

signed main() {

string a, b;

cin >> a >> b;

vector<int> A, B;

for(int i = a.size() - 1; i >= 0; i--) A.push\_back(a[i] - '0');

for(int i = b.size() - 1; i >= 0; i--) B.push\_back(b[i] - '0');

if(cmp(A, B)) {

auto c = sub(A, B);

for(int i = c.size() - 1; i >= 0; i--) {

cout << c[i];

}

} else {

auto c = sub(B, A);

cout << "-";

for(int i = c.size() - 1; i >= 0; i--) {

cout << c[i];

}

}

return 0;

}

/\*

\*除法 A / b = C ... r, A >= 0, b > 0

\*/

vector<int> div(vector<int> &A, int b, int &r) {

vector<int> C;

r = 0;

for (int i = A.size() - 1; i >= 0; i -- ) {

r = r \* 10 + A[i];

C.push\_back(r / b);

r %= b;

}

reverse(C.begin(), C.end());

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

signed main() {

string a;

int b;

int r = 0;

cin >> a >> b;

vector<int> A;

for(int i = a.size() - 1; i >= 0; i--) {

A.push\_back(a[i] - '0');

}

auto c = div(A, b, r);

for(int i = c.size() - 1; i >= 0; i--) {

cout << c[i];

}

puts("");

cout << r;

return 0;

}

/\*

\*乘法

\*/

#include <iostream>

#include <vector>

using namespace std;

vector<int> mul(vector<int> &A, vector<int> &B) {

vector<int> C(A.size() + B.size());

for (int i = 0; i < A.size(); i++)

for (int j = 0; j < B.size(); j++)

C[i + j] += A[i] \* B[j];

for (int i = 0, t = 0; i < C.size() || t; i++) {

t += C[i];

if (i >= C.size()) C.push\_back(t % 10);

else C[i] = t % 10;

t /= 10;

}

while (C.size() > 1 && C.back() == 0) C.pop\_back();

return C;

}

int main() {

string a, b;

cin >> a >> b;

vector<int> A, B;

for (int i = a.size() - 1; i >= 0; i--) A.push\_back(a[i] - '0');

for (int i = b.size() - 1; i >= 0; i--) B.push\_back(b[i] - '0');

vector<int> C = mul(A, B);

for (int i = C.size() - 1; i >= 0; i--) cout << C[i];

return 0;

}

### 1.6尺取法

/\*

\*尺取法 适用于l增大时侯r跟着增大取到一些合法的区间时使用

\*/

int ans = INF, l = 1, r = 1, cnt = 0;

for(; ;)

{

while(r <= n && cnt < sum) cnt += a[r++]; //直到符合总数

if(cnt < sum) break; //再也无法满足

ans = min(ans, r - l); //更新答案

cnt -= a[l++]; //尝试进步

}

### 1.7归并排序

int n, a[N], b[N]; //b为辅助数组

void merge\_dfs(int l, int mid, int r)

{

int i = l, j = mid + 1, k = l;

while(i <= mid && j <= r)

{

b[k++] = (a[i] < a[j] ? a[i++] : a[j++]); //排序

}

while(i <= mid) b[k++] = a[i++];

while(j <= r) b[k++] = a[j++];

RE(i, l, r) a[i] = b[i]; //还原

}

void dfs(int l, int r)

{

if(l == r) return;

int mid = l + r >> 1;

dfs(l, mid);

dfs(mid + 1, r);

merge\_dfs(l, mid, r);

}

## 数据结构

## 2.1单调栈

/\*

\*top = 0 栈空

\*st[++top] = x 插入

\*st[top] 栈顶

\*/

int top, st[N]; //指针 数组栈

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

if(top == 0) st[++top] = i;//为空入栈

//维护单调性 b[]存储答案

hile(top != 0 && a[i] > a[st[top]]) b[st[top]] = i, top--;

st[++top] = i;

}

## 2.2单调队列

/\*

\*head 队头 tail队尾

\*head <= tail 队列不为空

\*head++ 出队

\*q[++tail] = x 入队

\*/

int head, tail, q[N], idx[N];

head = 1, tail = 0;

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

while(head <= tail && a[i] > q[tail]) tail--;

q[++tail] = a[i];

idx[tail] = i;

if(idx[head] <= i - k) head++;//队头元素过时

if(i >= k) cout << q[head] << " ";

}

## 2.3 ST表

struct rmq {

int f[N][50], lg[N] = {-1};

void init() {

for(int i = 1; i <= n; ++i) f[i][0] = a[i];

for(int i = 1; i <= n; ++i) lg[i] = lg[i >> 1] + 1;

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

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

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

}

int query(int x, int y) {

int s = lg[y - x + 1];

return max(f[x][s], f[y - (1 << s) + 1][s]);

}

} st;

## 2.4树状数组

/\*

\*单点修改 区间查询

\*/

int lowbit(int x) {

return x & -x;

}

void change(int i, int x) {

for(; i <= n; i += lowbit(i)) c[i] += x;

}

//查询[1, i],

int query(int i) {

int res = 0;

for(; i; i -= lowbit(i)) res += c[i];

return res;

}

/\*

\*单点查询 区间修改

\*函数模板均跟第一个一样，读入a[]数组，以及区间修改时使用到差分

\*/

add(i, a[i] - a[i - 1]); //初始化

cin >> l >> r >> x;

add(l, x)， add(r + 1, -x); //差分更新

/\*

\*树状数组上二分求第k大：

\*假设有n个数, 求第k大等价于求(n - k + 1)小

\*可将权值用树状数组维护，二分查找第k大。

\*/

scanf("%d", &k);

k = num - k + 1;//num是实际存在的数的个数

int l = 1, r = n;//值域

while(l < r) {

int mid = l + r >> 1;

if(query(mid) >= k) r = mid;

else l = mid + 1;

}

cout << l;

## 2.5线段树

### 2.5.1线段树结构

/\*

\*线段树主要考虑区间维护的信息

\*以及如何设计lazy

\*区间修改区间查询模板

\*/

struct tnode {

int l, r; //区间边界

int sum, lazy; //需要维护的信息

};

struct segment\_tree {

tnode t[N << 2];

void build(int k, int l, int r) { //建树

t[k].l = l, t[k].r = r;

if(l == r) {

t[k].sum = a[l];

return;

}

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

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

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

t[k].sum = t[k << 1].sum + t[k << 1 | 1].sum; //push\_up

}

void push\_down(int k, int l, int r) { //下放懒标记

if(t[k].lazy) { //lazy值存在

int mid = l + r >> 1;

t[k << 1].lazy += t[k].lazy; //union\_lazy

t[k << 1 | 1].lazy += t[k].lazy;

t[k << 1].sum += t[k].lazy \* (mid - l + 1); //cal\_lazy

t[k << 1 | 1].sum += t[k].lazy \* (r - mid);

t[k].lazy = 0;

}

}

void change(int k, int l, int r, int val) {

if(t[k].l >= l && t[k].r <= r) {

t[k].lazy += val; //union\_lazy

t[k].sum += (t[k].r - t[k].l + 1) \* val; //cal\_lazy

return;

}

push\_down(k, t[k].l, t[k].r);

int mid = (t[k].l + t[k].r) >> 1;

if(r <= mid) change(k << 1, l, r, val);

else if(l > mid) change(k << 1 | 1, l, r, val);

else change(k << 1, l, mid, val), change(k << 1 | 1, mid + 1, r, val);

t[k].sum = t[k << 1].sum + t[k << 1 | 1].sum; //push\_up

}

int query(int k, int l, int r) {

if(t[k].l >= l && t[k].r <= r) {

return t[k].sum;

}

push\_down(k, t[k].l, t[k].r);

int mid = t[k].l + t[k].r >> 1;

if(r <= mid) return query(k << 1, l, r);

else if(l > mid) return query(k << 1 | 1, l, r);

else return query(k << 1, l, mid) + query(k << 1 | 1, mid + 1, r);

}

};

### 2.5.2线段树优化建图

/\*

\*某点到区间建边，不需要虚点

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 2e6 + 10;

int cnt, head[N];

struct edge {

int to, nex, w;

} e[N << 1];

inline void add(int u, int v, int w) {

e[++cnt].to = v;

e[cnt].w = w;

e[cnt].nex = head[u];

head[u] = cnt;

}

int n, m, s;

int tot, Ls[N], Rs[N], In, Out;

//入树 实现 u -> [L, R] 自上往下连边

void buildIn(int &k, int l, int r) {

if(l == r) {

k = l;

return;

}

k = ++tot;

int mid = l + r >> 1;

buildIn(Ls[k], l, mid);

buildIn(Rs[k], mid + 1, r);

add(k, Ls[k], 0);

add(k, Rs[k], 0);

}

//出树 实现 [L, R] -> u 自下往上连边

void buildOut(int &k, int l, int r) {

if(l == r) {

k = l;

return;

}

k = ++tot;

int mid = l + r >> 1;

buildOut(Ls[k], l, mid);

buildOut(Rs[k], mid + 1, r);

add(Ls[k], k, 0);

add(Rs[k], k, 0);

}

void addIn(int k, int l, int r, int L, int R, int tag, int w) {

if(l >= L && r <= R) {

add(tag, k, w);

return;

}

int mid = l + r >> 1;

if(L <= mid) addIn(Ls[k], l, mid, L, R, tag, w);

if(R > mid) addIn(Rs[k], mid + 1, r, L, R, tag, w);

}

void addOut(int k, int l, int r, int L, int R, int tag, int w) {

if(l >= L && r <= R) {

add(k, tag, w);

return;

}

int mid = l + r >> 1;

if(L <= mid) addOut(Ls[k], l, mid, L, R, tag, w);

if(R > mid) addOut(Rs[k], mid + 1, r, L, R, tag, w);

}

signed main() {

cin >> n >> m >> s; //n点m条边源点

tot = n;//线段树结点标号从n+1开始

buildIn(In, 1, n);

buildOut(Out, 1, n);

int op, l, r, u, v, w;

while(m-- && cin >> op) {

if(op == 1) {

cin >> u >> v >> w;

add(u, v, w);

} else if(op == 2) {

cin >> u >> l >> r >> w;

addIn(In, 1, n, l, r, u, w);

} else {

cin >> u >> l >> r >> w;

addOut(Out, 1, n, l, r, u, w);

}

}

dijstra();

return 0;

}

/\*

\*区间到区间建图，需要虚点p，q

\*/

#include <bits/stdc++.h>

#define ll long long

using namespace std;

template <typename T> inline void read(T& t) {

int f = 0, c = getchar();

t = 0;

while (!isdigit(c)) f |= c == '-', c = getchar();

while (isdigit(c)) t = t \* 10 + c - 48, c = getchar();

if (f) t = -t;

}

const int N = 5e6 + 10;

int cnt, head[N];

struct edge {

int to, nex, w;

} e[N << 1];

inline void add(int u, int v, int w) {

e[++cnt].to = v;

e[cnt].w = w;

e[cnt].nex = head[u];

head[u] = cnt++;

}

int n, m, s;

int tot, Ls[N], Rs[N], In, Out;

//入树 自上而下建边

void buildIn(int &k, int l, int r) {

if(l == r) {

k = l;

return;

}

k = ++tot;

int mid = l + r >> 1;

buildIn(Ls[k], l, mid);

buildIn(Rs[k], mid + 1, r);

add(k, Ls[k], 0);

add(k, Rs[k], 0);

}

//出树 自下而上建边

void buildOut(int &k, int l, int r) {

if(l == r) {

k = l;

return;

}

k = ++tot;

int mid = l + r >> 1;

buildOut(Ls[k], l, mid);

buildOut(Rs[k], mid + 1, r);

add(Ls[k], k, 0);

add(Rs[k], k, 0);

}

void addOut(int k, int l, int r, int L, int R, int idx) {

if(l >= L && r <= R) {

add(k, idx, 0);

return;

}

int mid = l + r >> 1;

if(L <= mid) addOut(Ls[k], l, mid, L, R, idx);

if(R > mid) addOut(Rs[k], mid + 1, r, L, R, idx);

}

void addIn(int k, int l, int r, int L, int R, int idx) {

if(l >= L && r <= R) {

add(idx, k, 0);

return;

}

int mid = l + r >> 1;

if(L <= mid) addIn(Ls[k], l, mid, L, R, idx);

if(R > mid) addIn(Rs[k], mid + 1, r, L, R, idx);

}

void insertt(int a, int b, int c, int d) {

int p = ++tot;

addOut(Out, 1, n, a, b, p);

int q = ++tot;

addIn(In, 1, n, c, d, q);

add(p, q, 1);

}

signed main() {

read(n), read(m), read(s);

tot = n;

buildIn(In, 1, n);

buildOut(Out, 1, n);

int a, b, c, d;

while(m--) {

read(a);

read(b);

read(c);

read(d);

insertt(a, b, c, d);

insertt(c, d, a, b);

}

dijstra();

return 0;

}

### 2.5.3维护GCD

#include <bits/stdc++.h>

#define go continue

#define int long long

#define IOS ios::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);

#define fory(i,a,b) for(int i = a; i <= b; ++i)

using namespace std;

const int N = 5e5 + 10;

int n, m, a[N];

inline int gcd(int a, int b) {

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

}

struct node {

int l, r, sum, d;

};

struct segment\_tree {

node t[N << 2];

int mp[N];

inline void help\_push(node& a, node& b, node& c) {

a.sum = b.sum + c.sum;

a.d = gcd(b.d, c.d);

}

inline void push\_up(int root) {

int ch = root << 1;

help\_push(t[root], t[ch], t[ch + 1]);

}

inline void build(int root, int l, int r) {

t[root].l = l;

t[root].r = r;

if(l != r) {

int ch = root << 1;

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

build(ch, l, mid);

build(ch + 1, mid + 1, r);

push\_up(root);

} else {

int tmp = a[l] - a[l - 1];

t[root].d = tmp;

t[root].sum = tmp;

mp[l] = root;

}

}

inline void change(int root, int x, int y) {

x = mp[x];

t[x].d = t[x].sum + y;

t[x].sum += y;

while(x >>= 1) push\_up(x);

}

inline node query(int root, int l, int r) {

if(t[root].l >= l && t[root].r <= r) {

return t[root];

} else {

int ch = root << 1;

int mid = (t[root].l + t[root].r) >> 1;

if(mid >= r) return query(ch, l, r);

else if(l > mid) return query(ch + 1, l, r);

else {

node left = query(ch, l, mid);

node right = query(ch + 1, mid + 1, r);

node res;

help\_push(res, left, right);

return res;

}

}

}

} st;

int l, r, x;

char op[2];

signed main() {

read(n), read(m);

fory(i, 1, n) read(a[i]);

st.build(1, 1, n);

while(m--) {

scanf("%s%lld%lld", op, &l, &r);

if(\*op == 'C') {

read(x);

st.change(1, l, x);

if(r + 1 <= n) st.change(1, r + 1, -x);

} else {

node ll = st.query(1, 1, l);

node rr = {0, 0, 0, 0};

if(l + 1 <= r) rr = st.query(1, l + 1, r);

printf("%lld\n", abs(gcd(ll.sum, rr.d)));

}

}

return 0;

}

### 2.5.4区间第K小

#include <bits/stdc++.h>

using namespace std;

const int N = 2e5 + 10;

int cnt, root[N];

int sum[N << 5], lc[N << 5], rc[N << 5];

int n, m, a[N], b[N], p;

void build(int& t, int l, int r) {

t = ++cnt;

if(l == r) return;

int mid = l + r >> 1;

build(lc[t], l, mid);

build(rc[t], mid + 1, r);

}

int change(int pre, int l, int r) {

int now = ++cnt;

lc[now] = lc[pre];

rc[now] = rc[pre];

sum[now] = sum[pre] + 1;

if(l == r) return now;

int mid = l + r >> 1;

if(p <= mid) lc[now] = change(lc[now], l, mid);

else rc[now] = change(rc[now], mid + 1, r);

return now;

}

int query(int L, int R, int l, int r, int k) {

if(l == r) return l;

int mid = l + r >> 1;

int x = sum[lc[R]] - sum[lc[L]];

if(x >= k) return query(lc[L], lc[R], l, mid, k);

else return query(rc[L], rc[R], mid + 1, r, k - x);

}

signed main() {

ios::sync\_with\_stdio(false), cin.tie(0);

cin >> n >> m;

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

cin >> a[i];

b[i] = a[i];

}

sort(b + 1, b + 1 + n);

int l, r, k, q;

q = unique(b + 1, b + 1 + n) - b - 1;

build(root[0], 1, q);

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

p = lower\_bound(b + 1, b + 1 + q, a[i]) - b;

root[i] = change(root[i - 1], 1, q);

}

while(m--) {

cin >> l >> r >> k;

cout << b[query(root[l - 1], root[r], 1, q, k)] << "\n";

}

return 0;

}

## 2.6 LCA

/\*

\*倍增法求LCA

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 5e5 + 10;

int head[N], cnt;

struct node {

int to, nex;

} e[N << 1];

void add\_edge(int u, int v) {

e[++cnt].to = v;

e[cnt].nex = head[u];

head[u] = cnt;

}

int dep[N], f[N][30], clg[N];

void dfs(int u, int fa) {

f[u][0] = fa;

dep[u] = dep[fa] + 1;

for(int i = 1; i <= clg[dep[u]]; ++i) {

f[u][i] = f[f[u][i - 1]][i - 1];

}

for(int i = head[u]; i; i = e[i].nex) {

if(e[i].to != fa) dfs(e[i].to, u);

}

}

int lca(int x, int y) {

if(dep[x] < dep[y]) swap(x, y);

while(dep[x] > dep[y]) x = f[x][clg[dep[x] - dep[y]] - 1];

if(x == y) return x;

for(int i = clg[dep[x]] - 1; i >= 0; --i) {

if(f[x][i] != f[y][i]) {

x = f[x][i];

y = f[y][i];

}

}

return f[x][0];

}

int n, m, s;

signed main() {

cin >> n >> m >> s;

int x, y;

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

cin >> x >> y;

add\_edge(x, y);

add\_edge(y, x);

}

for(int i = 1; i <= n; ++i) clg[i] = clg[i - 1] + (1 << clg[i - 1] == i);

dfs(s, 0);

while(m--) {

cin >> x >> y;

cout << lca(x, y) << "\n";

}

return 0;

}

/\*

\*树上任意两点距离

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 4e4 + 100;

int cnt, head[N];

struct node {

int to, nex, w;

} e[N << 1];

void add\_edge(int u, int v, int w) {

e[++cnt].to = v;

e[cnt].w = w;

e[cnt].nex = head[u];

head[u] = cnt;

}

int dep[N], clg[N], f[N][30], dis[N];

void dfs(int u, int fa) {

f[u][0] = fa;

dep[u] = dep[fa] + 1;

for(int i = 1; i <= clg[dep[u]]; ++i) {

f[u][i] = f[f[u][i - 1]][i - 1];

}

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(v != fa) {

dis[v] = dis[u] + e[i].w;

dfs(v, u);

}

}

}

int lca(int x, int y) {

if(dep[x] < dep[y]) swap(x, y);

while(dep[x] > dep[y]) x = f[x][clg[dep[x] - dep[y]] - 1];

if(x == y) return x;

for(int i = clg[dep[x]] - 1; i >= 0; --i) {

if(f[x][i] != f[y][i]) {

x = f[x][i];

y = f[y][i];

}

}

return f[x][0];

}

//公式

int cal\_dis(int u, int v) {

return dis[u] + dis[v] - 2 \* dis[lca(u, v)];

}

int n, q, u, v, w;

signed main() {

for(int i = 1; i < N; ++i) clg[i] = clg[i - 1] + (1 << clg[i - 1] == i);

int t;

cin >> t;

while(t--) {

cnt = 0;

memset(head, -1, sizeof head);

dis[1] = dep[1] = 0;

cin >> n >> q;

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

cin >> u >> v >> w;

add\_edge(u, v, w);

add\_edge(v, u, w);

}

dfs(1, 0);

while(q-- && cin >> u >> v) cout << cal\_dis(u, v) << "\n";

}

return 0;

}

## 2.7莫队

/\*

\*询问[l, r]之间有多少个不同的数字

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 1e6 + 10;

int n, m;

int a[N], pos[N], ans[N], cnt;

map<int, int> mp;

struct qnode {

int l, r, id;

} q[N];

void add(int n) {

if(mp[a[n]]) mp[a[n]]++;

else {

mp[a[n]] = 1;

cnt++;

}

}

void sub(int n) {

mp[a[n]]--;

if(mp[a[n]] == 0) cnt--;

}

signed main() {

ios::sync\_with\_stdio(false), cin.tie(0);

cin >> n;

int siz = sqrt(n);

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

cin >> a[i];

pos[i] = i / siz;

}

cin >> m;

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

cin >> q[i].l >> q[i].r;

q[i].id = i;

}

sort(q + 1, q + 1 + m, [](qnode x, qnode y) {

return pos[x.l] == pos[y.l] ? x.r < y.r : pos[x.l] < pos[y.l];

});

int l = 1, r = 0;

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

while(q[i].l < l) add(--l);

while(q[i].r > r) add(++r);

while(q[i].l > l) sub(l++);

while(q[i].r < r) sub(r--);

ans[q[i].id] = cnt;

}

for(int i = 1; i <= m; ++i) cout << ans[i] << "\n";

return 0;

}

/\*

\*离线区间查询mex

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 2e5 + 100;

int n, m, a[N], mp[N];

int val[N], cnt[N], L[N], R[N], pos[N], tot;

int ok[N];

struct qnode {

int l, r, id;

} q[N];

bool cmp(qnode& a, qnode& b) {

return (pos[a.l] == pos[b.l]) ? a.r < b.r : pos[a.l] < pos[b.l];

}

int Q() {

int idx = 1;

while(val[idx] == R[idx] - L[idx] + 1) idx++;

for(int i = L[idx]; i <= R[idx]; ++i) if(!cnt[i]) return i;

}

void add(int x) {

if(cnt[x] == 0) val[pos[x]]++;

cnt[x]++;

}

void sub(int x) {

if(cnt[x] == 1) val[pos[x]]--;

cnt[x]--;

}

signed main() {

read(n), read(m);

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

read(a[i]);

if(a[i] > n) a[i] = n + 1;

}

int siz = sqrt(n);

for(int i = 0; i <= n; ++i) pos[i] = i / siz + 1;

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

L[++tot] = i, R[tot] = i + siz - 1;

}

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

read(q[i].l), read(q[i].r);

q[i].id = i;

}

sort(q + 1, q + 1 + m, cmp);

int l = 1, r = 0;

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

while(l < q[i].l) sub(a[l++]);

while(l > q[i].l) add(a[--l]);

while(r > q[i].r) sub(a[r--]);

while(r < q[i].r) add(a[++r]);

ok[q[i].id] = Q();

}

for(int i = 1; i <= m; ++i) printf("%d\n", ok[i]);

return 0;

}

/\*

\*带修莫队

\*询问[L, R]多少种不同颜色

\*修改某个位置的颜色

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 1e6 + 10;

int n, m, ans;

int a[N], ok[N], mp[N];

struct qnode {

int l, r, pre, id, bkl, bkr;

} q[N];

int qnum, cnum;

struct cnode {

int pos, val;

} c[N];

void change(int now, int i) {

if(c[now].pos >= q[i].l && c[now].pos <= q[i].r) {

if(--mp[a[c[now].pos]] == 0) ans--;

if(++mp[c[now].val] == 1) ans++;

}

swap(c[now].val, a[c[now].pos]);

}

void add(int x) {

if(++mp[x] == 1) ans++;

}

void sub(int x) {

if(--mp[x] == 0) ans--;

}

inline bool cmp(register qnode a, register qnode b) {

return a.bkl != b.bkl ? a.bkl < b.bkl : (a.bkr != b.bkr ? a.bkr < b.bkr : a.pre < b.pre);

}

void moqueue() {

int l = 1, r = 0, now = 0;

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

while(l < q[i].l) sub(a[l++]);

while(l > q[i].l) add(a[--l]);

while(r < q[i].r) add(a[++r]);

while(r > q[i].r) sub(a[r--]);

while(now < q[i].pre) now++, change(now, i);

while(now > q[i].pre) change(now, i), now--;

ok[q[i].id] = ans;

}

for(int i = 1; i <= qnum; ++i) cout << ok[i] << "\n";

}

signed main() {

read(n), read(m);

int siz = pow(n, 2.0 / 3.0);

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

read(a[i]);

}

char op[N];

while(m--) {

scanf("%s", op);

if(\*op == 'Q') {

read(q[++qnum].l);

read(q[qnum].r);

q[qnum].pre = cnum;

q[qnum].id = qnum;

q[qnum].bkl = (q[qnum].l - 1) / siz + 1;

q[qnum].bkr = (q[qnum].r - 1) / siz + 1;

} else {

read(c[++cnum].pos);

read(c[cnum].val);

}

}

sort(q + 1, q + 1 + qnum, cmp);

moqueue();

return 0;

}

## 2.8树链剖分（重链）

/\*

\*1 x y z 树从 x 到 y 结点最短路径上所有节点的值都加上 z

\*2 x y 求树从 x 到 y 结点最短路径上所有节点的值之和

\*3 x z 表示将以 x 为根节点的子树内所有节点值都加上 z

\*4 x 表示求以 x 为根节点的子树内所有节点值之和

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 2e5 + 10;

int cnt, head[N];

struct edge {

int to, nex;

} e[N << 1];

inline void add\_edge(int u, int v) {

e[++cnt].to = v;

e[cnt].nex = head[u];

head[u] = cnt;

}

int n, m, root, mod;

int a[N], w[N]; //初始点权, dfs序后的点权

int dfn[N], siz[N], son[N], f[N], top[N], dep[N], tim; //时间戳 子树大小 重儿子 父亲 当前链顶端节点

struct segment\_tree {

struct tnode {

int l, r, sum, lazy;

};

tnode t[N << 2];

inline void push\_down(int root) {

if(t[root].lazy != 0) {

t[root].sum += (t[root].lazy \* (t[root].r - t[root].l + 1)) % mod;

if(t[root].l != t[root].r) {

int ch = root << 1;

t[ch].lazy += t[root].lazy;

t[ch + 1].lazy += t[root].lazy;

}

t[root].lazy = 0;

}

}

inline void push\_up(int root) {

int ch = root << 1;

push\_down(ch);

push\_down(ch + 1);

t[root].sum = (t[ch].sum + t[ch + 1].sum) % mod;

}

inline void build(int root, int l, int r) {

t[root].l = l, t[root].r = r;

if(l != r) {

int ch = root << 1;

int mid = l + r >> 1;

build(ch, l, mid);

build(ch + 1, mid + 1, r);

push\_up(root);

} else {

t[root].lazy = 0;

t[root].sum = w[l] % mod;

}

}

void change(int root, int l, int r, int k) {

push\_down(root);

if(t[root].l >= l && t[root].r <= r) {

t[root].lazy += k;

return;

}

int ch = root << 1;

int mid = t[root].l + t[root].r >> 1;

if(r <= mid) change(ch, l, r, k);

else if(l > mid) change(ch + 1, l, r, k);

else change(ch, l, mid, k), change(ch + 1, mid + 1, r, k);

push\_up(root);

}

int query(int root, int l, int r) {

push\_down(root);

if(t[root].l >= l && t[root].r <= r) {

return t[root].sum % mod;

}

int ch = root << 1;

int mid = t[root].l + t[root].r >> 1;

if(r <= mid) return query(ch, l, r) % mod;

else if(l > mid) return query(ch + 1, l, r) % mod;

else return (query(ch, l, mid) % mod + query(ch + 1, mid + 1, r) % mod) % mod;

}

} st;

void dfs1(int u, int fa) {

f[u] = fa;

dep[u] = dep[fa] + 1;

siz[u] = 1;

int maxsize = -1;

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(v == fa) continue;

dfs1(v, u);

siz[u] += siz[v];

if(siz[v] > maxsize) {

maxsize = siz[v];

son[u] = v;

}

}

}

void dfs2(int u, int t) {

dfn[u] = ++tim;

top[u] = t;

w[tim] = a[u];

if(!son[u]) return;

dfs2(son[u], t);

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(v == f[u] || v == son[u]) continue;

dfs2(v, v);

}

}

inline void op1(int x, int y, int k) {

k %= mod;

while(top[x] != top[y]) {

if(dep[top[x]] < dep[top[y]]) swap(x, y);

st.change(1, dfn[top[x]], dfn[x], k);

x = f[top[x]];

}

if(dep[x] > dep[y]) swap(x, y);

st.change(1, dfn[x], dfn[y], k);

}

inline int op2(int x, int y) {

int cnt = 0;

while(top[x] != top[y]) {

if(dep[top[x]] < dep[top[y]]) swap(x, y);

cnt += st.query(1, dfn[top[x]], dfn[x]);

x = f[top[x]];

}

if(dep[x] > dep[y]) swap(x, y);

cnt += st.query(1, dfn[x], dfn[y]) % mod;

return cnt % mod;

}

inline void op3(int x, int k) {

st.change(1, dfn[x], dfn[x] + siz[x] - 1, k);

}

inline int op4(int x) {

return st.query(1, dfn[x], dfn[x] + siz[x] - 1);

}

signed main() {

ios::sync\_with\_stdio(false), cin.tie(0);

cin >> n >> m >> root >> mod;

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

int x, y;

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

cin >> x >> y;

add\_edge(x, y);

add\_edge(y, x);

}

dfs1(root, root);

dfs2(root, root);

st.build(1, 1, n);

while(m--) {

int op, z;

cin >> op;

if(op == 1) {

cin >> x >> y >> z;

op1(x, y, z);

} else if(op == 2) {

cin >> x >> y;

cout << op2(x, y) << "\n";

} else if(op == 3) {

cin >> x >> y;

op3(x, y);

} else {

cin >> x;

cout << op4(x) << "\n";

}

}

return 0;

}

## 2.9 dsu on tree

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

int cnt, head[N];

struct E {

int to, nex;

} e[N << 1];

inline void add\_edge(int u, int v) {

e[++cnt].to = v, e[cnt].nex = head[u], head[u] = cnt;

}

int siz[N], son[N];

void dfs(int u, int fa) {

siz[u] = 1;

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(v == fa) continue;

dfs(v, u);

siz[u] += siz[v];

if(siz[v] > siz[son[u]]) son[u] = v;

}

}

int a[N], ok[N], cot[N], sum;

void cal(int u, int fa, int val) { //统计函数

cot[a[u]] += val;

if(val == 1 && cot[a[u]] == 1) sum++;

if(val == -1 && cot[a[u]] == 0) sum--;

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(v != fa) cal(v, u, val);

}

}

void dsu(int u, int fa, bool flag) {

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(v != fa && v != son[u]) dsu(v, u, true);//往轻儿子方向走

}

if(son[u] != 0) dsu(son[u], u, false);//往重儿子方向走

cot[a[u]]++;

if(cot[a[u]] == 1) sum++;//计算自己的贡献

for(int i = head[u]; i; i = e[i].nex) { //计算轻儿子的贡献

int v = e[i].to;

if(v != fa && v != son[u]) cal(v, u, 1);

}

ok[u] = sum;

if(flag) cal(u, fa, -1);//释放掉cot数组 清空其他信息

}

int n, m;

signed main() {

ios::sync\_with\_stdio(false), cin.tie(0);

cin >> n;

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

cin >> u >> v;

add\_edge(u, v), add\_edge(v, u);

}

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

dfs(1, 0);

dsu(1, 0, 0);

cin >> m;

int x;

while(m-- && cin >> x) cout << ok[x] << "\n";

return 0;

}

# 数论

## 3.1质数

/\*

\*质数：严格大于1 的整数，只有自己和1两个因子。

\*算术基本定理：任何一个大于 1的自然数N，N不为质数，可以唯一被分解有

\*限个质数的乘积：N = p1^a1 \* p2^a2 \* .. \* pn^an

\*且最多有一个大于sqrt(n)的质因子

\*试除法判断质数：o(sqrt(n))

\*/

bool is\_prime(int x) {

if (x < 2) return false;

for (int i = 2; i <= x / i; i ++ )

if (x % i == 0)

return false;

return true;

}

/\*

\*分解质因数

\*/

void divide(int x) {

for (int i = 2; i <= x / i; i ++ )

if (x % i == 0) {

int s = 0;

while (x % i == 0) x /= i, s++;

cout << i << ' ' << s << endl;

}

if (x > 1) cout << x << ' ' << 1 << endl;

cout << endl;

}

/\*

\***线性筛法**

\*/

int primes[N], cnt;

bool st[N];

void get\_primes(int n) {

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

if (!st[i]) primes[cnt++] = i;

for (int j = 0; primes[j] <= n / i; ++j) {

st[primes[j] \* i] = true;

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

}

}

}

## 3.2约数

/\*

\*一个数的约数集合

\*/

vector<int> get\_divisors(int x) {

vector<int> res;

for (int i = 1; i <= x / i; ++i)

if (x % i == 0) {

res.push\_back(i);

if (i != x / i) res.push\_back(x / i);

}

sort(res.begin(), res.end());

return res;

}

/\*

\*约数个数：

\*由算术基本定理、组合数原理

\*N = p1^a1 + p2^a2 + .. + pn^an

\*设K为约数，sum为约数个数

\*K = p1^b1 \* p2^b2 \* ... \* pn ^bn (0 <= bi <= ai)

\*sum = (a1 + 1) \* (a2 + 1) \* .. \* (an + 1)

\*/

//多个数相乘得到的约数个数，统计指数最后再计算即可

unordered\_map<int, int> primes;

while (n--) {

int x;

cin >> x;

for (int i = 2; i <= x / i; ++i)

while (x % i == 0) {

x /= i;

primes[i]++;

}

if (x > 1) primes[x]++;

}

LL res = 1;

for (auto p : primes) res = res \* (p.second + 1) % mod;

cout << res << endl;

/\*

\*约数之和：

\*(p1^0 + p1^1 + ... + p1^a1) \* (p2^0 + p2^1 + ... + p2^a2) \* ...

\*(pn^0 + pn^1 + ... + pn^an)

\*/

unordered\_map<int, int> primes;

while (n--) {

int x;

cin >> x;

for (int i = 2; i <= x / i; i++)

while (x % i == 0) {

x /= i;

primes[i]++;

}

if (x > 1) primes[x]++;

}

LL res = 1;

for (auto p : primes) {

LL a = p.first, b = p.second;

LL t = 1;

while (b--) t = (t \* a + 1) % mod;

res = res \* t % mod;

}

cout << res << endl;

## 3.3组合数

/\*

\*打表方式组合数，数据范围较小

\*c[i][j] 代表i个里面选j个

\*/

int c[N][N]; //c[i][j] 代表i个里面选j个

inline void init(){

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

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

if(!j) c[i][j] = 1;

else c[i][j] = (c[i - 1][j - 1] + c[i - 1][j]) % mod;

}

}

/\*

\*数据范围预处理on到1e7，计算o1

\*/

const int N = 2e5 + 10;

const int mod = 1e9 + 7;

int fac[N], inv[N];

int qk(int a, int b) {

int ans = 1;

while(b) {

if(b & 1) ans = ans \* a % mod;

a = a \* a % mod;

b >>= 1;

}

return ans;

}

void init() {

fac[0] = 1;

for(int i = 1; i < N; ++i) fac[i] = fac[i - 1] \* i % mod;

inv[N - 1] = qk(fac[N - 1], mod - 2);

for(int i = N - 2; i >= 0; --i) inv[i] = inv[i + 1] \* (i + 1) % mod;

}

int C(int a, int b){

if(b < 0 || b > a) return 0;

return fac[a] \* inv[a - b] % mod \* inv[b] % mod; //组合数定义

}

/\*

\*卢卡斯定理 适合n， m1e18级别， p1e5

\*时间复杂度20\*N\*log(p)N\*lgP

\*/

int n, m, p;

int qk(int a, int b, int mod) {

int res = 1;

while(b) {

if(b & 1) res = res \* a % mod;

a = a \* a % mod;

b >>= 1;

}

return res;

}

int C(int a, int b, int p) {

int res = 1;

for(int i = 1, j = a; i <= b; i++, j--) {

res = res \* j % p;

res = res \* qk(i, p - 2, p) % p;

}

return res;

}

int lucas(int a, int b, int p) {

if(a < p && b < p) return C(a, b, p);

return C(a % p, b % p, p) \* lucas(a / p, b / p, p) % p;

}

## 3.3欧拉函数

/\*

f(n) : 1 - n 中与n互质的个数(gcd(a, b) = 1)

n = p1^a1.p2^a2...pn^an

f(n) = n \* (1 - 1/p1)(1 - 1/p2)...(1 - 1/pn)

\*/

int div\_ola(int x) {

int res = x;

for(int i = 2; i <= x / i; ++i) {

if(x % i == 0) {

res = res / i \* (i - 1);

while(x % i == 0) x /= i;

}

}

if(x > 1) res = res / x \* (x - 1);

return res;

}

/\*

\*筛法

\*/

const int N = 1e5 + 10;

int cnt, p[N], st[N], e[N];

void div\_ola(int n) {

e[1] = 1;

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

if(!st[i]) {

p[cnt++] = i;

e[i] = i - 1;

}

for(int j = 0; p[j] <= n / i; ++j) {

int t = p[j] \* i;

st[t] = true;

if(i % p[j] == 0) {

e[t] = e[i] \* p[j];

break;

}

e[t] = e[i] \* (p[j] - 1);

}

}

}

signed main() {

div\_ola(100);

for(int i = 1; i <= 100; ++i) cout << e[i] << " ";

return 0;

}

## 3.4SG函数

/\*

\*打表版本

\*/

int sg[N], vis[N], f[N];

void cal\_sg(int n) {

for(int i = 1; i <= n; ++i) { //从1开始 sg[0] = 0

memset(vis, 0, sizeof vis); //清空后继状态

for(int j = 1; f[j] <= i; ++j) { //f[i]保证有序且预处理的最大权值>=n

vis[sg[i - f[j]]] = 1;

}

for(int j = 0; ; ++j) if(!vis[j]) {

sg[i] = j;

break;

}

}

}

/\*

\*dfs版本 用于不好打表的情况

\*/

int sg[N][N], f[N];

int dfs(int x) {

if(sg[x]) return sg[x];

int vis[N] = {0};

for(int i = 0; i < n; ++i) { //种类数

if(x >= f[i]) { //如果合法

dfs(x - f[i]);

vis[sg[x - f[i]] = 1

}

}

for(int i = 0; ; ++i) if(!vis[i]) {

return sg[x] = i;

}

}

# 动态规划

## 4.1背包问题

### 01背包

/\*

\*01背包基础写法

\*/

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

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

if(j >= v[i]) f[i][j] = max(f[i - 1][j], f[i - 1][j - v[i]] + w[i]);

else f[i][j] = f[i - 1][j];

}

}

/\*

\*回滚优化 优化空间

\*/

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

for(int j = m; j >= v[i]; --j) { //一定是从后往前跑

f[j] = max(f[j], f[j - v[i]] + w[i]);

}

}

/\*

\*有些01变形不好使用回滚，再加个辅助数组帮助优化空间

\*/

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

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

if(j < v[i]) f[j][0] = f[j][1];

else f[j][0] = max(f[j][1], f[j - v[i]][1] + w[i]);

}

for(int j = 1; j <= m; ++j) f[j][1] = f[j][0];

}

cout << f[m][0];

/\*

\*求刚好装满的最大值，初始化均为负值，f[0] = 0

\*/

for(int i = 0; i < mx; i++) f[i] = -inf;//一个极小负值

f[0] = 0;

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

cin >> v[i] >> w[i];

}

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

for (int j = m; j >= v[i]; j--) {

f[j] = max(f[j], f[j - v[i]] + w[i]);

if (f[j] < 0) f[j] = -inf;

}

}

if(f[m] > 0) cout << f[m];

else cout << "no";

/\*

\*求n个物品，具有重量和价值，价值max，不超过背包容量的方案总数。

\*/

cin >> n >> m;

for(int i = 0; i <= m; ++i) cnt[i] = 1;

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

cin >> v >> w;

for(int j = m; j >= v; --j) {

int val = f[j - v] + w;

if(val > f[j]) { //当前值更大，替换最优方案数

f[j] = val;

cnt[j] = cnt[j - v];

} else if(val == f[j]) cnt[j] = (cnt[j] + cnt[j - v]) % mod;

//出现了一个相同的最大值，叠加

}

}

cout << cnt[m];

/\*

\*n个整数，从中任意选出若干个组成m的方案数

\*/

f[0] = 1;

int x;

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

cin >> x;

for(int j = m; j >= x; --j) f[j] += f[j - x];

}

cout << f[m];

### 4.1.2完全背包

/\*

\*物品可重复选取

\*/

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

cin >> v >> w;

for(int j = v; j <= m; ++j) { //正着跑

f[j] = max(f[j], f[j - v] + w);

}

}

cout << f[m];

/\*

\*方案数

\*/

f[0] = 1;

int x;

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

cin >> x;

for(int j = m; j >= x; --j) f[j] += f[j - x];

}

cout << f[m];

### 4.1.3多重背包

/\*

\*每种物品数量不定，二进制枚举优化

\*/

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

cin >> a >> b >> s;

int k = 1;

while(s - k >= 0) {

s -= k;

v[++cnt] = a \* k;

w[cnt] = b \* k;

k <<= 1;

}

if(s > 0) {

v[++cnt] = a \* s;

w[cnt] = b \* s;

}

}

n = cnt;

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

for(int j = m; j >= v[i]; --j) {

f[j] = max(f[j], f[j - v[i]] + w[i]);

}

}

cout << f[m];

### 4.1.4分组背包

/\*

\*每组物品只能选一种

\*/

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

cin >> s[i];

for(int j = 1; j <= s[i]; ++j) cin >> v[i][j] >> w[i][j];

}

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

for(int j = m; j > 0; --j) {

for(int k = 1; k <= s[i]; ++k) {

if(v[i][k] <= j)

f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);

}

}

}

cout << f[m];

## 4.2数位DP

int dfs(int pos, bool limit, bool lead)

{

if (pos == cnt)

return 1;

int ans = 0;

if (dp != -1)return dp;

for (int v = 0; v <= (limit ? A[pos] : 9); v++) // 根据是否limit决定循环上界

{

ans += dfs(pos + 1, limit && v == A[pos], lead && v == 0);

}

return dp = ans;

}

int f(int x) {

if (x == 0)return 0;

cnt = 0;

memset(A, 0, sizeof(A));

memset(dp, -1, sizeof(dp)); // 初始化dp数组为-1

while (x)

A[cnt++] = x % 10, x /= 10;

reverse(A, A + cnt);

return dfs(0, true, true);

}

## 4.3区间DP

f[i][j]: [i, j]区间的dp值

len = (n << 1)

rep(L, 2, n) { //长度

for(int i = 1; i <= len - L + 1; ++i) { //起点

int j = i + L - 1; //终点

for(int k = i; k < j; ++k) { //枚举中点

//更新

}

//更新

}

}

# 图论

## 5.1最短路

/\*

\*弗洛伊德多源点最短路

\*/

RE(i, 1, n)

RE(j, 1, n) {

g[i][j] = i == j ? 0 : INF;

}

RE(k, 1, n)

RE(i, 1, n)

RE(j, 1, n) {

g[i][j] = min(g[i][j], g[i][k] + g[k][j]);

}

/\*\*\

单源最短路

输入: n m s t 接下来m行 u, v, w表示u -> v 有一条权值为w的无向边

input:

3 3 1 2

1 2 3

2 3 4

1 3 5

output:

3

\\*\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

struct node {

int to, w, next;

} e[N << 1];

int cnt = 0, head[N];

inline void add\_edge(int u, int v, int w) {

e[++cnt].to = v;

e[cnt].w = w;

e[cnt].next = head[u];

head[u] = cnt;

}

inline void init() {

cnt = 0;

memset(head, -1, sizeof head);

}

int n, m, s, t;

struct v {

int x, dis;

bool operator < (const v& a) const {

return dis > a.dis; //stl默认大顶堆

}

};

int dis[N];

bool vis[N];

/\*\*\

dijstra

1、从源点开始每次选取一个离点集距离最近的点t 添加到集合中

2、利用t点对集合中的点进行松弛操作，进行更新

3、使用堆进行1找点的操作降低时间复杂度

4、不能在有负边权的图中使用

\\*\*/

inline void dijstra(int s, int t) {

priority\_queue<v> q;

memset(dis, 0x3f, sizeof dis);

memset(vis, 0, sizeof vis);

dis[s] = 0;

q.push({s, 0});

while(!q.empty()) {

v now = q.top();

q.pop();

if(vis[now.x]) continue;

vis[now.x] = 1;

for(int i = head[now.x] ; i != -1; i = e[i].next) {

int y = e[i].to;

if(vis[y]) continue;

if(dis[y] > dis[now.x] + e[i].w) {

dis[y] = dis[now.x] + e[i].w;

q.push({y, dis[y]});

}

}

}

if(dis[t] >= 0x3f3f3f3f) cout << -1;

else cout << dis[t];

}

/\*\*\

spfa算法思路:

1、每次迭代,取出队头点v,依次枚举从v出发的边,v->u,设边的长度为w

判断dis[v] + w 是否小于dis[u], 小于则更新值,

2、由于s-u的距离变短了,有可能u能改变其他点,使用vis数组判断是否在队列,没有则放入

3、若一个点的入队次数超过n,则存在负环

\\*\*/

queue<int> q1;

inline void spfa(int s, int t) {

memset(dis, 0x3f, sizeof dis);

memset(vis, 0, sizeof vis);

dis[s] = 0;

vis[s] = 1;

q1.push(s);

while(!q1.empty()) {

int x = q1.front();

q1.pop();

vis[x] = 0;

for(int i = head[x]; i != -1; i = e[i].next) {

int y = e[i].to;

if(dis[y] > dis[x] + e[i].w) {

dis[y] = dis[x] + e[i].w;

if(!vis[y]) {

q1.push(y);

vis[y] = 1;

}

}

}

}

if(dis[t] >= 0x3f3f3f3f) cout << -1;

else cout << dis[t];

}

inline void solve() {

init();

cin >> n >> m >> s >> t;

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

int x, y, z;

cin >> x >> y >> z;

add\_edge(x, y, z);

add\_edge(y, x, z);

}

//dijstra(s, t);

spfa(s, t);

}

signed main() {

solve();

return 0;

}

## 5.2拓扑排序

/\*\*\

拓扑排序:每次找到入度为0的点删掉

判断一个有向图中是否有环，无环的图都能进行拓扑排序

\\*\*/

const int N = 1e5 + 10;

int inr[N]; //记录入度的数组

struct node {

int t, next;

} e[N << 1];

int cnt = 0;

int head[N];

int n, m;

void add\_edge(int x, int y) {

e[++cnt].t = y;

e[cnt].next = head[x];

head[x] = cnt;

}

void topo() {

queue<int> q;

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

if(inr[i] == 0) q.push(i);

}

int idx = 0, ans[N];

while(!q.empty()) {

int x = q.front();

ans[++idx] = x;

q.pop();

for(int i = head[x]; i != -1; i = e[i].next) {

inr[e[i].t]--;

if(inr[e[i].t] == 0) q.push(e[i].t);

}

}

if(idx != n) cout << -1;

else {

for(int i = 1; i <= idx; ++i) cout << ans[i] << " ";

}

}

signed main() {

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

memset(head, -1, sizeof head);

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

int x, y;

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

inr[y]++;

add\_edge(x, y);

}

topo();

return 0;

}

## 5.3二分图

/\*

\*染色法判断二分图

\*二分图性质：至少两个点，回路长度均为偶数

\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

int cnt, head[N];

struct edge {

int to, nex;

} e[N << 1];

inline void add\_edge(int u, int v) {

e[++cnt].to = v;

e[cnt].nex = head[u];

head[u] = cnt;

}

int n, m, col[N];

vector<int> v1, v2;

bool dfs(int u) {

for(int i = head[u]; i; i = e[i].nex) {

int v = e[i].to;

if(!col[v]) {

col[v] = 3 - col[u];

if(!dfs(v)) return false;

}

if(col[v] == col[u]) return false;

}

return true;

}

signed main() {

ios::sync\_with\_stdio(false), cin.tie(0);

cin >> n >> m;

int u, v;

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

cin >> u >> v;

add\_edge(u, v);

add\_edge(v, u);

}

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

if(col[i]) continue;

if(!col[i]) {

col[i] = 1;

if(!dfs(i)) {

cout << "-1\n";

return 0;

}

}

}

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

if(col[i] == 1) v1.push\_back(i);

else v2.push\_back(i);

}

cout << v1.size() << "\n";

for(int i = 0; i < v1.size(); ++i) cout << v1[i] << " \n"[i == v1.size() - 1];

cout << v2.size() << "\n";

for(int i = 0; i < v2.size(); ++i) cout << v2[i] << " \n"[i == v2.size() - 1];

return 0;

}

/\*\*\

https://www.luogu.com.cn/problem/P3386

二分图最大匹配匈牙利算法:

假设左部是男孩,右部是女孩

1、对于一个男孩子,如果他喜欢女孩y,女孩y没有配偶,就在一起

2、如果y有配偶了,x还是想去挖一手墙脚,y也不是什么好女人,

如果y发现他的配偶z能换一个对象(这里dfs),y就主动抛弃z(z能换一个对象),跟x在一起

这样就多了一对配偶

\\*\*/

#include <bits/stdc++.h>

using namespace std;

const int N = 1010;

int n, m, e;//左部 右部 边数

int g[N][N], vis[N], link[N];

//dfs搜索的起点是左集合, link[]代表右集合

int dfs(int x) {

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

if(!g[x][i] || vis[i]) continue; //如果没有边 或者已经被邀请过了 就跳过

vis[i] = 1;

if(link[i] == 0 || dfs(link[i])) { //如果没有配偶,或者能让自己的配偶找到配偶(找到增广路),就在一起

link[i] = x;

return 1;

}

}

return 0;

}

signed main() {

int ok = 0;

cin >> n >> m >> e;

for(int i = 1, x, y; i <= e; ++i) {

cin >> x >> y;

g[x][y] = 1;

}

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

memset(vis, 0, sizeof vis);

if(dfs(i)) ok++;//如果能匹配成功,边数++

}

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

return 0;

}

/\*

\*二分图最大权完美匹配

\*/

#include<bits/stdc++.h>

using namespace std;

int n, m; //左右点均为n m条边

long long edge[505][505];

int matched[505];//记录右部点所匹配的左部点

int visy[505];

int lx[505], ly[505]; //lx表示左部点的顶标，ly表示右部点的顶标

int pre[505];

long long slack[505];

void bfs(int want\_match) {

int now, new\_match = 0;

int x = 0; //0点初始化

memset(pre, 0, sizeof pre); //重新建立交错树（因为每一次要重新找增广路）

memset(slack, 0x3f, sizeof slack); //初始化松弛量

matched[x] = want\_match; //初始化0点匹配到了u这个左部点

do {

now = matched[x]; //找到当前寻找匹配的左部点

long long delta = 1e18; //delta记录寻找的增广路路径的松弛量

visy[x] = true;

//标记本轮已经在这个点找过增广路，和匈牙利算法中vis数组同理

for(register int i = 1; i <= n; i++) { //枚举右部点

if(!visy[i]) { //找到了一个还没有尝试找增广路的右部点

if(slack[i] > lx[now] + ly[i] - edge[now][i]) {

//找到了一个匹配点

//优化DFS进行的find操作

slack[i] = lx[now] + ly[i] - edge[now][i];

pre[i] = x; //记录交错树(BFS对DFS的优化)

}

if(slack[i] < delta) {

delta = slack[i]; //更新delta

new\_match = i;

}

}

}

for(register int i = 0; i <= n; i++) { //因为初始化的是0，所以说必须从0开始修改顶标

if(visy[i]) {

lx[matched[i]] -= delta; //i点匹配到的左部点的顶标减去delta

ly[i] += delta; //i点的顶标加上delta

//和DFS版本同理

} else slack[i] -= delta; //进行松弛操作

}

x = new\_match; //下一个访问的左部点即刚才取出的右部点的匹配点

//此处x指代找到的右部点

} while(matched[x]);

while(x) matched[x] = matched[pre[x]], x = pre[x]; //交错树还原操作

}

long long KM() {

memset(matched, 0, sizeof matched);

memset(lx, 0, sizeof lx);

memset(ly, 0, sizeof ly);

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

memset(visy, 0, sizeof visy); //每次重新进行增广

bfs(i);

}

long long ans = 0;

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

ans += (long long)edge[matched[i]][i]; //正常的统计答案

return ans;

}

int main() {

n = read(), m = read();

memset(edge, -0x3f, sizeof edge); //赋初值为负无穷，因为题目边权有负数

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

int u = read(), v = read(), w = read();

edge[u][v] = max(edge[u][v], (long long)w);

}

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

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

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

return 0;

}

## 5.4网络流

### 5.4.1最大流dinic

#include <bits/stdc++.h>

using namespace std;

const int N = 1e4 + 10, M = 2e5 + 10;

int n, m, S, T;

int head[N], ver[M], edge[M], Next[M], idx;

int deep[N], cur[M];

void add(int a, int b, int c) {

ver[idx] = b, edge[idx] = c, Next[idx] = head[a], head[a] = idx++;

ver[idx] = a, edge[idx] = 0, Next[idx] = head[b], head[b] = idx++;

}

bool bfs() {

queue<int> q;

memset(deep, 0, sizeof deep);

q.push(S);

cur[S] = head[S];

deep[S] = 1;

while (q.size()) {

int t = q.front();

q.pop();

for (int i = head[t]; i != -1; i = Next[i]) {

int e = ver[i];

if (deep[e] == 0 && edge[i]) {

deep[e] = deep[t] + 1;

cur[e] = head[e];

if (e == T)

return 1;

q.push(e);

}

}

}

return 0;

}

int find(int fa, int limit) {

if (fa == T)

return limit;

int flow = 0;

for (int i = cur[fa]; i != -1 && flow < limit; i = Next[i]) {

cur[fa] = i;

int e = ver[i];

if (deep[e] == deep[fa] + 1 && edge[i]) {

int t = find(e, min(edge[i], limit - flow));

if (!t) {

deep[e] = 0;

continue;

}

edge[i] -= t, edge[i ^ 1] += t, flow += t;

}

}

return flow;

}

int dinic() {

int maxflow = 0, flow;

while (bfs())

while (flow = find(S, INT\_MAX))

maxflow += flow;

return maxflow;

}

int main() {

cin >> n >> m >> S >> T;

memset(head, -1, sizeof head);

while (m--) {

int a, b, c;

cin >> a >> b >> c;

add(a, b, c);

}

cout << dinic();

}

### 5.4.2最小费用最大流

#include <bits/stdc++.h>

using namespace std;

const int N = 5e3 + 10, M = 1e5 + 10;

int n, m, S, T;

int head[N], ver[M], edge[M], w[M], Next[M], idx;

int dist[N], pre[N], incf[N];

bool vis[N];

void add(int a, int b, int c, int d) {

ver[idx] = b, edge[idx] = c, w[idx] = d, Next[idx] = head[a], head[a] = idx++;

ver[idx] = a, edge[idx] = 0, w[idx] = -d, Next[idx] = head[b], head[b] = idx++;

}

bool spfa() {

queue<int> q;

memset(dist, 0x3f, sizeof dist);

memset(incf, 0, sizeof incf);

incf[S] = INT\_MAX, dist[S] = 0;

q.push(S);

while (q.size()) {

int t = q.front();

q.pop();

vis[t] = false;

for (int i = head[t]; i != -1; i = Next[i]) {

int e = ver[i];

if (edge[i] && dist[e] > dist[t] + w[i]) {

dist[e] = dist[t] + w[i];

pre[e] = i;

incf[e] = min(edge[i], incf[t]);

if (!vis[e]) {

q.push(e);

vis[e] = 1;

}

}

}

}

return incf[T] > 0;

}

void EK(int &flow, int &cost) {

flow = cost = 0;

while (spfa()) {

int t = incf[T];

flow += t, cost += t \* dist[T];

for (int i = T; i != S; i = ver[pre[i] ^ 1]) {

edge[pre[i]] -= t;

edge[pre[i] ^ 1] += t;

}

}

}

signed main() {

cin >> n >> m >> S >> T;

memset(head, -1, sizeof head);

while (m--) {

int a, b, c, d;

cin >> a >> b >> c >> d;

add(a, b, c, d);

}

int flow, cost;

EK(flow, cost);

cout << flow << " " << cost;

}

# 字符串

长度为n的字符串

1、有n(n+1)/2 +1个子串；

2、非空子串：n（n+1）/2；

3、非空真子串：n（n+1）/2– 1。

## 6.5哈希

/\*

\*双哈希统计多少个相同的字符串 可以利用ull的自动取模

\*/

#define ull unsigned long long

const int N = 1e5 + 10;

const int mod = 1e9 + 7;

const int P = 131, PP = 1331;

ull p[N], pp[N], hs1[N], hs2[N];

char s[N];

set<pair<ull, ull> > st;

ull get1(int l, int r) {

return (hs1[r] - hs1[l - 1] \* p[r - l + 1] % mod + mod) % mod;

}

ull get2(int l, int r) {

return (hs2[r] - hs2[l - 1] \* pp[r - l + 1] % mod + mod) % mod;

}

void cal\_hash(int l, int r) {

ull t1 = get1(l, r), t2 = get2(l, r);

st.insert({t1, t2});

}

signed main() {

int T;

scanf("%d", &T);

while(T-- && scanf("%s", s + 1)) {

int n = strlen(s + 1);

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

p[i] = (p[i - 1] \* P) % mod;

pp[i] = (pp[i - 1] \* PP) % mod;

hs1[i] = (hs1[i - 1] \* P % mod + s[i]) % mod;

hs2[i] = (hs2[i - 1] \* PP % mod + s[i]) % mod;

}

cal\_hash(1, n);

}

printf("%d\n", st.size());

return 0;

}

/\*

\*n,m的矩阵，查询有q个大小a,b的矩阵出现

\*/

#define ull unsigned long long

using namespace std;

const int N = 1e3 + 10;

char ch;

ull h[N][N];

ull p[N], pp[N];

unordered\_map<ull, int> mp;

ull get(int x1, int y1, int x2, int y2) {

return h[x2][y2] - h[x2][y1 - 1] \* p[y2 - y1 + 1] - h[x1 - 1][y2] \* pp[x2 - x1 + 1]

+ h[x1 - 1][y1 - 1] \* p[y2 - y1 + 1] \* pp[x2 - x1 + 1];

}

int n, m, a, b, q;

signed main() {

cin >> n >> m >> a >> b;

p[0] = pp[0] = 1;

for(int i = 1; i <= 1000; ++i) {

p[i] = p[i - 1] \* 131;

pp[i] = pp[i - 1] \* 233;

}

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

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

cin >> ch;

h[i][j] = h[i][j - 1] \* 131 + ch;

}

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

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

h[i][j] = h[i][j] + h[i - 1][j] \* 233;

}

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

for(int j = 1; j <= m - b + 1; ++j) {

ull z = get(i, j, i + a - 1, j + b - 1);

mp[z]++;

}

cin >> q;

while(q--) {

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

for(int j = 1; j <= b; ++j) {

cin >> ch;

h[i][j] = h[i][j - 1] \* 131 + ch;

}

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

for(int j = 1; j <= b; ++j) {

h[i][j] = h[i][j] + h[i - 1][j] \* 233;

}

cout << (mp[h[a][b]] ? 1 : 0) << endl;

}

}

## 6.5 Kmp

int n, m;

char s[N], p[N];

int ne[N];

signed main() {

cin >> s + 1 >> p + 1;

n = strlen(s + 1), m = strlen(p + 1);

for(int i = 2, j = 0; i <= m; ++i) {

while(j && p[i] != p[j + 1]) j = ne[j];

if(p[i] == p[j + 1]) ++j;

ne[i] = j; //更新ne值

}

for(int i = 1, j = 0; i <= n; ++i) {

while(j && s[i] != p[j + 1]) j = ne[j];

if(s[i] == p[j + 1]) ++j;

if(j == m) {

cout << i - m + 1 << "\n"; //第一个字符的位置

j = ne[j];

}

}

}

## 6.6 Manacher

/\*

\*p[i] 回文半径 p[i] - 1即是回文串长度

\*(i - p[i] + 1) / 2 为原回文串长度为p[i] - 1的起点

\*/

char s[N], t[N << 1];

int n, p[N << 1];

void manacher() {

int L = 0;

t[++L] = '$', t[++L] = '#';

for(int i = 1; i <= n; ++i) t[++L] = s[i], t[++L] = '#';

int mx = 0, id, ans = -1;

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

if(mx >= i) p[i] = min(mx - i, p[2 \* id - i]);

else p[i] = 1;

while(t[i - p[i]] == t[i + p[i]]) ++p[i];

if(i + p[i] > mx) mx = i + p[i], id = i;

if(ans < p[i]) ans = p[i];

}

cout << ans - 1;

}

/\*

\*判断某个区间是否是回文

\*/

bool check(int L, int R) {

int mid = L + R + 1;

return 2 \* p[mid] - 1 >= 2 \* (R - L);

}

## 6.7 后缀数组

/\*

\*sa[i]：排名为i的后缀开始

\*hel[i]：sa[i]与sa[i - 1] 的最长公共前缀lcp

\*某个字符串的本质不同的子串是n \* (n + 1) / 2 - sum(hel[i])

\*/

int n, m; //m 值域

char s[N];

int sa[N], hel[N];

int h1[N], h2[N], rk[N], c[N];

void SA() {

int \*x = h1, \*y = h2;

for(int i = 1; i <= n; ++i) c[x[i] = s[i]]++;

for(int i = 2; i <= m; ++i) c[i] += c[i - 1];

for(int i = n; i; --i) sa[c[x[i]]--] = i; //第一关键字排序

for(int k = 1; k <= n; k <<= 1) {

int num = 0;

for(int i = n - k + 1; i <= n; ++i) y[++num] = i;

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

if(sa[i] > k) y[++num] = sa[i] - k;

}

for(int i = 1; i <= m; ++i) c[i] = 0;

for(int i = 1; i <= n; ++i) c[x[i]]++;

for(int i = 2; i <= m; ++i) c[i] += c[i - 1];

for(int i = n; i; --i) sa[c[x[y[i]]]--] = y[i], y[i] = 0;

swap(x, y);

x[sa[1]] = 1, num = 1;

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

x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] + k]) == 1 ? num : ++num;

if(num == n) break;

m = num;

}

for(int i = 1; i <= n; i++) rk[sa[i]] = i;

for(int i = 1, k = 0; i <= n; i++) {

if(rk[i] == 1) continue;

if(k) k--;

int j = sa[rk[i] - 1];

while(i + k <= n && j + k <= n && s[i + k] == s[j + k]) k++;

hel[rk[i]] = k;

}

}

/\*

\*2个串求公共最长子串

\*len1为第一个串的长度

\*/

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

if((sa[i] <= len1) != (sa[i - 1] <= len1)) {

if(hel[i] > ans) ans = hel[i];

}

}