目录

[string.h 1](#_Toc89595848)

[最小生成树模板（Prim） 4](#_Toc89595849)

[最小生成树（Kruskal） 5](#_Toc89595850)

[线段树模板（可进行加和乘法） 6](#_Toc89595851)

[线段树模板（加法） 9](#_Toc89595852)

[二叉苹果树 12](#_Toc89595853)

[多重背包（二进制拆分） 13](#_Toc89595854)

[最大网络流（增广路算法） 14](#_Toc89595855)

[石子合并 16](#_Toc89595856)

[高斯消元法 18](#_Toc89595857)

[维护数列（splay模板） 28](#_Toc89595858)

[偷天换日 35](#_Toc89595859)

[树状数组 36](#_Toc89595860)

[数学知识 45](#_Toc89595861)

[//组合数 45](#_Toc89595862)

[//欧几里得算法 45](#_Toc89595863)

[//拓展欧几里得算法 45](#_Toc89595864)

[//逆元 45](#_Toc89595865)

[//lucas定理 45](#_Toc89595866)

[//快速幂 46](#_Toc89595867)

[//中国剩余定理 46](#_Toc89595868)

[//大步小步算法 46](#_Toc89595869)

[//快速筛法求素数表 47](#_Toc89595870)

[//欧拉函数 47](#_Toc89595871)

[//莫比乌斯函数 48](#_Toc89595872)

[//高斯消元 48](#_Toc89595873)

[//高精度 49](#_Toc89595874)

[//矩阵乘法 52](#_Toc89595875)

# string.h

函数名: strcpy

功 能: 拷贝一个字符串到另一个

用 法: char \*strcpy(char \*destin, char \*source);

函数名: strcat

功 能: 字符串拼接函数

用 法: char \*strcat(char \*destin, char \*source);

函数名: strchr

功 能: 在一个串中查找给定字符的第一个匹配之处\

用 法: char \*strchr(char \*str, char c);

函数名: strcmp

功 能: 串比较

用 法: int strcmp(char \*str1, char \*str2);

看Asic码，str1>str2，返回值 > 0；两串相等，返回0

函数名: strncmpi

功 能: 将一个串中的一部分与另一个串比较, 不管大小写

用 法: int strncmpi(char \*str1, char \*str2, unsigned maxlen);

函数名: strcpy

　　功 能: 串拷贝

用 法: char \*strcpy(char \*str1, char \*str2);

函数名: strcspn

　　功 能: 在串中查找第一个给定字符集内容的段

用 法: int strcspn(char \*str1, char \*str2);

函数名: strdup

　　功 能: 将串拷贝到新建的位置处

用 法: char \*strdup(char \*str);

函数名: stricmp

　　功 能: 以大小写不敏感方式比较两个串

用 法: int stricmp(char \*str1, char \*str2);

函数名: strerror

　　功 能: 返回指向错误信息字符串的指针

用 法: char \*strerror(int errnum);

函数名: strcmpi

　　功 能: 将一个串与另一个比较, 不管大小写

用 法: int strcmpi(char \*str1, char \*str2);

函数名: strncmp

　　功 能: 串比较

用 法: int strncmp(char \*str1, char \*str2, int maxlen);

函数名: strncpy

　　功 能: 串拷贝

用 法: char \*strncpy(char \*destin, char \*source, int maxlen);

函数名: strnicmp

　　功 能: 不注重大小写地比较两个串

用 法: int strnicmp(char \*str1, char \*str2, unsigned maxlen);

函数名: strnset

　　功 能: 将一个字符串前n个字符都设为指定字符

用 法: char \*strnset(char \*str, char ch, unsigned n);

函数名: strpbrk

　　功 能: 在串中查找给定字符集中的字符

用 法: char \*strpbrk(char \*str1, char \*str2);

函数名: strrchr

　　功 能: 在串中查找指定字符的最后一个出现

用 法: char \*strrchr(char \*str, char c);

函数名: strrev

　　功 能: 串倒转

用 法: char \*strrev(char \*str)

函数名: strset

　　功 能: 将一个串中的所有字符都设为指定字符

用 法: char \*strset(char \*str, char c);

函数名: strspn

　　功 能: 在串中查找指定字符集的子集的第一次出现

用 法: int strspn(char \*str1, char \*str2);

函数名: strstr

　　功 能: 在串中查找指定字符串的第一次出现

用 法: char \*strstr(char \*str1, char \*str2)

函数名: strtod

　　功 能: 将字符串转换为double型值

用 法: double strtod(char \*str, char \*\*endptr);

函数名: strtok

　　功 能: 查找由在第二个串中指定的分界符分隔开的单词

用 法: char \*strtok(char \*str1, char \*str2);

函数名: strtol

　　功 能: 将串转换为长整数

用 法: long strtol(char \*str, char \*\*endptr, int base);

函数名: strupr

　　功 能: 将串中的小写字母转换为大写字母

用 法: char \*strupr(char \*str);

函数名: swab

　　功 能: 交换字节

　　用 法: void swab (char \*from, char \*to, int nbytes);

# 最小生成树模板（Prim）

#include<stdio.h>

#define MAXCOST 0x7fffffff

int graph[5005][5005];

int n, m;

void prim() {

int lowcost[5005];

int mst[5005];

int sum = 0;

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

lowcost[i] = graph[1][i];

mst[i] = 1;

}

mst[1] = 0;

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

int min = MAXCOST;

int minid = 0;

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

if (lowcost[j] < min && lowcost[j] != 0) {

min = lowcost[j];

minid = j;

}

}

sum += min;

lowcost[minid] = 0;

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

if (graph[minid][j] < lowcost[j]) {

lowcost[j] = graph[minid][j];

mst[j] = minid;

}

}

}

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

}

int main() {

int x, y, d;

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

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

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

graph[i][j] = MAXCOST;

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

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

graph[x][y] = d;

graph[y][x] = d;

}

prim();

}

# 最小生成树（Kruskal）

#include<iostream>

#include<algorithm>

using namespace std;

struct Edge {

long long a, b;

long long d;

}edge[1000005];

long long n, m, s=0;

long long root[9005];

bool cmp(Edge a, Edge b) {

return a.d < b.d;

}

long long find(long long a) {

if (root[a] == a) return a;

return root[a] = find(root[a]);

}

bool Union(int a, int b, int d) {

if (a == b)

return 0;

root[a] = b;

s += d;

return 1;

}

void krukal() {

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

root[i] = i;

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

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

if (Union(find(edge[i].a), find(edge[i].b), edge[i].d))

n--;

}

if (n == 1)

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

else

printf("orz\n");

}

int main() {

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

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

scanf("%lld%lld%lld", &edge[i].a, &edge[i].b, &edge[i].d);

krukal();

}

# 线段树模板（可进行加和乘法）

#include<cstdio>

#include<cstdlib>

#include<algorithm>

using namespace std;

typedef long long ll;

const int N = 1e5 + 5;

#define MOD 571373

#define lchild i << 1

#define rchild i << 1 | 1

#define mid ((l + r) >> 1)

int w[N];

struct tree

{

ll l;

ll r;

ll sum;

ll add;

ll mul;

}node[N << 2];

inline void update(int i) //修改子节点后，当前节点的值重新计算，前提是子节点已经计算了sum

{

node[i].sum = (node[lchild].sum + node[rchild].sum) % MOD;

}

inline void build(int i, ll l, ll r)

{

node[i].l = l;

node[i].r = r;

node[i].add = 0;

node[i].mul = 1;

if (l == r)

{

node[i].sum = w[l] % MOD;

return;

}

build(lchild, l, mid);

build(rchild, mid + 1, r);

update(i);

}

inline void pushdown(int i)//计算当前标记对子节点的影响，同时将当前节点标记清零，当前节点的sum通过相应的update更新

{

ll l = node[i].l;

ll r = node[i].r;

ll len = r - l + 1;

node[lchild].sum = (node[lchild].sum \* node[i].mul + node[i].add \* (len - (len >> 1))) % MOD;

node[rchild].sum = (node[rchild].sum \* node[i].mul + node[i].add \* (len >> 1)) % MOD;

node[lchild].mul = (node[lchild].mul \* node[i].mul) % MOD;

node[rchild].mul = (node[rchild].mul \* node[i].mul) % MOD;

node[lchild].add = (node[lchild].add \* node[i].mul + node[i].add) % MOD;

node[rchild].add = (node[rchild].add \* node[i].mul + node[i].add) % MOD;

node[i].add = 0;

node[i].mul = 1;

}

inline void modify1(int i, ll x, ll y, ll t)

{

ll l = node[i].l;

ll r = node[i].r;

if (x <= l && r <= y)

{

node[i].sum = (node[i].sum \* t) % MOD;

node[i].mul = (node[i].mul \* t) % MOD;

node[i].add = (node[i].add \* t) % MOD;

return;

}

pushdown(i);

if (x <= mid)

modify1(lchild, x, y, t);

if (mid < y)

modify1(rchild, x, y, t);

update(i);

}

inline void modify2(int i, ll x, ll y, ll t)

{

ll l = node[i].l;

ll r = node[i].r;

if (x <= l && r <= y)

{

node[i].sum = (node[i].sum + (r - l + 1) \* t) % MOD;

node[i].add = (node[i].add + t) % MOD;

return;

}

pushdown(i);

if (x <= mid)

modify2(lchild, x, y, t);

if (mid < y)

modify2(rchild, x, y, t);

update(i);

}

inline ll getsum(int i, ll x, ll y)

{

ll l = node[i].l;

ll r = node[i].r;

if (x <= l && r <= y)

return (node[i].sum + MOD) % MOD;

pushdown(i);

ll ans = 0;

if (x <= mid)

ans += getsum(lchild, x, y);

if (mid < y)

ans += getsum(rchild, x, y);

return (ans + MOD) % MOD;

}

int main()

{

ll n, m, p;

scanf("%lld%lld%lld", &n, &m, &p);

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

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

build(1, 1, n);

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

{

ll x, y, k;

int c;

scanf("%d", &c);

if (c == 1)

{

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

modify1(1, x, y, k % MOD);

}

else if (c == 2)

{

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

modify2(1, x, y, k % MOD);

}

else

{

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

printf("%lld\n", getsum(1, x, y));

}

}

return 0;

}

# 线段树模板（加法）

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define lchild i<<1

#define rchild (i<<1)| 1

#define mid ((l+r)>>1)

const int N = 1e5 + 5;

typedef long long ll;

ll w[N];

struct tree {

ll l;

ll r;

ll sum;

ll add;

}node[N << 2];

inline void update(int i) {

node[i].sum = node[lchild].sum + node[rchild].sum;

}

inline void pushdown(int i) {

ll len = node[i].r - node[i].l + 1;

node[lchild].sum += node[i].add \* (len-(len>>1));

node[rchild].sum += node[i].add \* (len >> 1);

node[lchild].add += node[i].add;

node[rchild].add += node[i].add;

node[i].add = 0;

}

inline void build(int i, ll l, ll r) {

node[i].l = l;

node[i].r = r;

if (l == r) {

node[i].sum = w[l];

node[i].add = 0;

return;

}

build(lchild, l, mid);

build(rchild, mid + 1, r);

update(i);

}

inline void modify(int i, ll x, ll y, ll t) {

ll l = node[i].l;

ll r = node[i].r;

if (x <= l && r <= y) {

node[i].sum += (r-l+1)\*t;

node[i].add += t;

return;

}

pushdown(i);

if (x <= mid)

modify(lchild, x, y, t);

if (mid < y)

modify(rchild, x, y, t);

update(i);

}

inline ll getsum(int i, ll x, ll y) {

ll l = node[i].l;

ll r = node[i].r;

if (x <= l && r <= y) {

return node[i].sum;

}

ll ans = 0;

pushdown(i);

if (x <= mid) {

ans += getsum(lchild, x, y);

}

if (mid < y)

ans += getsum(rchild, x, y);

return ans;

}

int main() {

ll n, m;

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

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

scanf("%lld", &w[i]);

}

build(1, 1, n);

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

int choose;

scanf("%d", &choose);

if (choose == 1) {

ll x, y, k;

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

modify(1, x, y, k);

}

if (choose == 2) {

ll x, y;

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

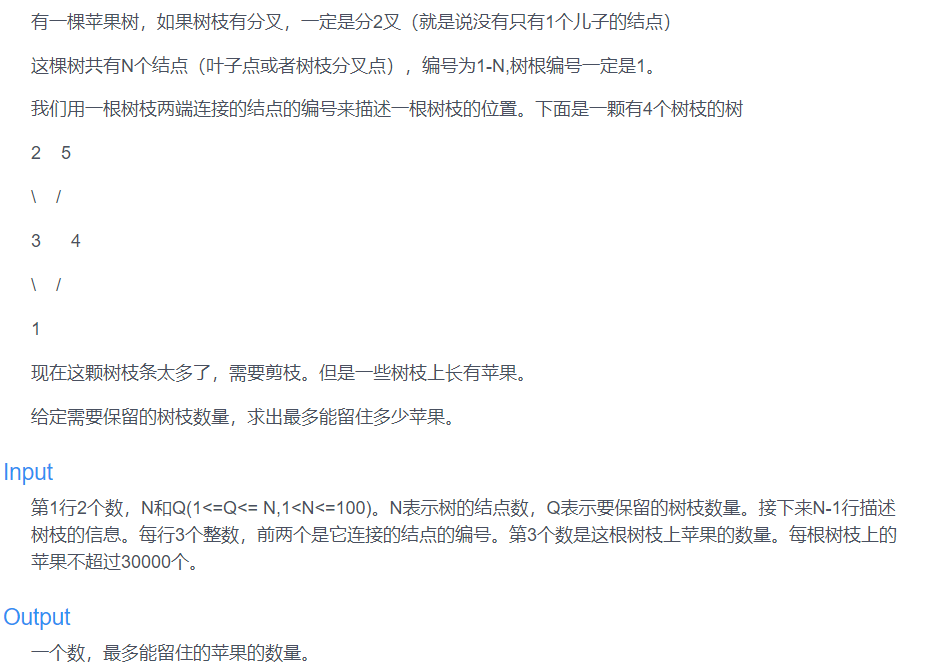
printf("%lld\n", getsum(1, x, y));

}

}

}

# 二叉苹果树



#include<stdio.h>

struct hhx {

int to, next;

}a[240];

int m, n, t = 0;;

int b[120][120];//保存苹果树

int f[120][120],head[120],p[120];

int max(int a,int b) {

if (a > b)

return a;

return b;

}

void add(int x, int y) {

a[++t].to = y;

a[t].next = head[x];

head[x] = t;

}

void dp(int d) {

p[d] = 1;

for (int i = head[d]; i; i = a[i].next) {

if (p[a[i].to] == 1)

continue;

dp(a[i].to);

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

for (int k = j - 1; k >= 0; k--)

f[d][j] = max(f[d][j], f[a[i].to][k] + f[d][j - k - 1] + b[d][a[i].to]);

}

}

}

int main() {

int x, y, z;

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

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

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

b[x][y] = z;

b[y][x] = z;

add(x, y);

add(y, x);

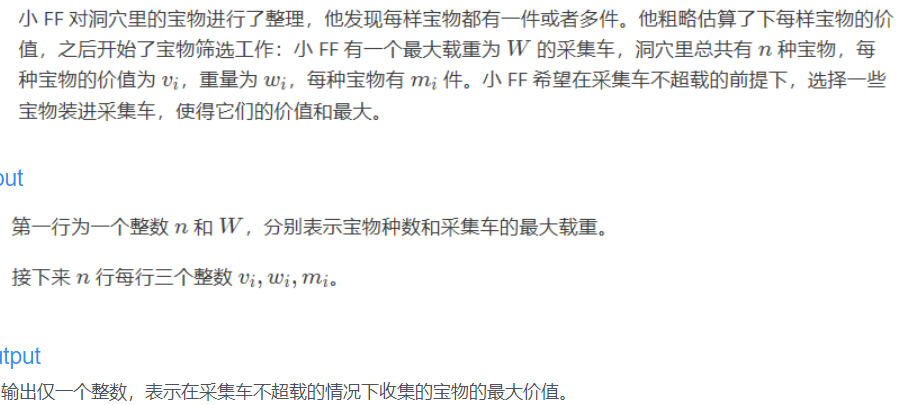
}

dp(1);

printf("%d\n", f[1][m]);

}

# 多重背包（二进制拆分）



#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#define N 100010

#define inf 1 << 30

using namespace std;

int W, n, f[N], ans, v[N], w[N], cnt, c;

int main()

{

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

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

{

int vv, ww, mm;

scanf("%d%d%d", &vv, &ww, &mm);

c = mm;

for (int j = 0; (1 << j) <= c; j++)

{

int x = 1 << j;

w[++cnt] = x \* ww, v[cnt] = x \* vv;

c -= x;

}

if (c) w[++cnt] = c \* ww, v[cnt] = c \* vv;

}

for (int i = 1; i <= W; i++) f[i] = -inf;

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

for (int j = W; j >= w[i]; j--)

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

for (int i = 0; i <= W; i++) ans = max(ans, f[i]);

printf("%d", ans);

return 0;

}

# 最大网络流（增广路算法）

#include <cstdio>

#include <cstring>

#include <iostream>

#include <string>

#include <algorithm>

#include <map>

#include <vector>

using namespace std;

const int N = 1100;

const int INF = 0x3f3f3f3f;

struct Node

{

int to;//终点

int cap; //容量

int rev; //反向边

};

vector<Node> v[N];

bool used[N];

void add\_Node(int from,int to,int cap) //重边情况不影响

{

v[from].push\_back((Node){to,cap,v[to].size()});

v[to].push\_back((Node){from,0,v[from].size()-1});

}

int dfs(int s,int t,int f)

{

if(s==t)

return f;

used[s]=true;

for(int i=0;i<v[s].size();i++)

{

Node &tmp = v[s][i]; //注意

if(used[tmp.to]==false && tmp.cap>0)

{

int d=dfs(tmp.to,t,min(f,tmp.cap));

if(d>0)

{

tmp.cap-=d;

v[tmp.to][tmp.rev].cap+=d;

return d;

}

}

}

return 0;

}

int max\_flow(int s,int t)

{

int flow=0;

for(;;){

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

int f=dfs(s,t,INF);

if(f==0)

return flow;

flow+=f;

}

}

int main()

{

int n,m;

while(~scanf("%d%d",&n,&m))

{

memset(v,0,sizeof(v));

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

{

int x,y,z;

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

add\_Node(x,y,z);

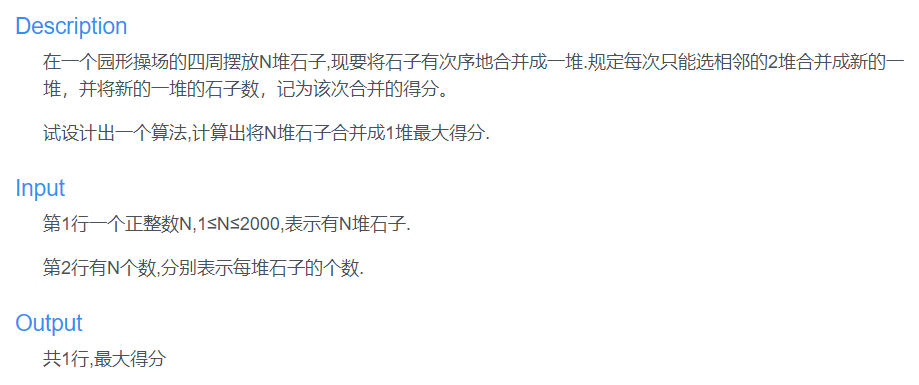
}

printf("%d\n",max\_flow(1,m));

}

}

# 石子合并



#include<stdio.h>

int f[4005][4005];

int s[4005][4005],sum[4005], n, MAX = -1;

int max(int a, int b) {

if (a > b)

return a;

return b;

}

void solve() {

for (int i = 1; i <= 2 \* n - 1; i++)

s[i][i] = i;

for (int L = 2; L <= 2 \* n - 1 ; L++) {

for (int i = 1; i <= 2 \* n - 1 - L + 1; i++) {

int j = i + L - 1;

/\*f[i][j] = -1;

for (int k = s[i][j - 1]; k <= s[i + 1][j] && k+1 <= j && k>=i ; k++) {

if (f[i][j] < f[i][k] + f[k + 1][j] + sum[j] - sum[i - 1]) {

f[i][j] = f[i][k] + f[k + 1][j] + sum[j] - sum[i - 1];

s[i][j] = k;

}

}\*/

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

}

}

}

int main() {

int temp;

scanf("%d", &n);

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

scanf("%d", &temp);

sum[i] = temp;

sum[i + n] = sum[i];

}

for (int i = 1; i <= 2 \* n -1; i++)

sum[i] += sum[i - 1];

solve();

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

MAX = max(MAX, f[i][i + n - 1]);

}

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

}

# 高斯消元法



#include<iostream>

#include<cstdio>

#include<cstring>

#include<cmath>

#include<string>

#include<map>

#include<set>

using namespace std;

#define MV 1000

#define MN 1000

#define is\_digit(c) (c>='0' && c<='9')

#define is\_alpha(c) (c>='a' && c<='z')

typedef double ll;

set<string> vars;

set<string>::iterator iter;

map<string,ll>eqm[MN];

int n,cols;

double A[MN][MV+1],res[MV];

char text[1024];

/\*

Expression: //处理+,-

Term = Number

Term = -Number

Term: //\*或者同级运算

Primary

Term + Primary

Term - Primary

Primary: //处理

Var

Number\*Var

-Primary

+Primary

Var: //处理整数、括号

'a'~'z' + '0'~'99'

Number:

double>=0

\*/

#define MAXCHARS 10240

#define number 1

#define varname 2

//Token用于保存词法分析结果

class Token {

public:

char kind; // token的类型，字符型的类型用字符本身表示，数字的类型为number（1）,变量为var(2)，行结束用0表示

ll value; // kind=number时才有意义，value中保存浮点数

string var; // kind=varname时才有意义，val中保存变量名

Token(char ch) // 解析到字符或者结束字符0

:kind(ch), value(0) { }

Token(char ch, ll val) // 解析到整数

:kind(ch), value(val) { }

Token(char ch, string val) // 解析到变量

:kind(ch), var(val) { }

};

//Token\_stream用于扫描输入流

class Token\_stream {

public:

Token\_stream();

Token get();

void putback(Token t);

char getone();

void putback();

void readline();

void clear();

private:

bool full;

Token buffer;

char line[MAXCHARS];

int index;

set<string> varset;

}ts;

Token\_stream::Token\_stream()

:full(false), buffer(0)

{

}

void Token\_stream::clear()//每次重新开始计算前清除

{

full=false;

buffer=0;

eqm[n].clear();

varset.clear();

index=0;

}

void Token\_stream::readline()//从标准输入读入一行，自动过滤回车、换行

{

index=0;

char ch;

while(1)

{

ch=getchar();

if(ch!='\n'&&ch!=EOF)

line[index++]=ch;

else

break;

}

if(index==0)

throw -2;

line[index]=0;

index=0;

}

void Token\_stream::putback()//解析过头了可以撤回，用于仅当读入下一个字符才知道当前单词是否结束的情况，比如读12345+

{

index--;

}

char Token\_stream::getone()//从缓冲区读入一个字符

{

return line[index++];

}

void Token\_stream::putback(Token t)//用于token不在当前级别语法范围，退回

{

if (!full)

{

buffer = t;

full = true;

}

}

Token Token\_stream::get() //词法分析

{

if (full) { //先看看是否有退回的，如果有直接从退回的缓冲区去取

full=false;

return buffer;

}

while(true){//读完一个token

char ch=getone();

switch (ch) {

case '=':

case '+':

case '-':

case '\*':

case '^':

case 0 ://行结束

return Token(ch);

case '.':

case '0': case '1': case '2': case '3': case '4':

case '5': case '6': case '7': case '8': case '9':

{

ll val=0;

int flag=0;

ll fbase=0.1;

while(ch=='.' || ch>='0' && ch<='9')

{

if(ch=='.')

flag=1;

else

{

if(flag)

{

val+=fbase\*(ch-'0');

fbase=fbase/10.0;

}

else

{

val=val\*10+ch-'0';

}

}

ch=getone();

}

putback();

return Token(number,val);

}

case 'a': case 'b': case 'c': case 'd': case 'e':

case 'f': case 'g': case 'h': case 'i': case 'j':

case 'k': case 'l': case 'm': case 'n': case 'o':

case 'p': case 'q': case 'r': case 's': case 't':

case 'u': case 'v': case 'w': case 'x': case 'y':

case 'z':

{

string str;

str=ch;

ch=getone();

if(is\_digit(ch))

{

str+=ch;

ch=getone();

if(is\_digit(ch))

str+=ch;

else

putback();

}

else

throw -1;

return Token(varname,str);

}

default:

throw -1;//其他字符为非法字符

}

}//while

}

pair<string,ll> primary()

{

Token t = ts.get();

switch (t.kind) {

case '-':

{

pair<string,ll> d=primary();

return make\_pair(d.first,-d.second);

}

case '+':

return primary();

case number:

{

Token t1 = ts.get();

if(t1.kind!='\*')

throw -1;

t1=ts.get();

if(t1.kind!=varname)

throw -1;

return make\_pair(t1.var,t.value);

}

case varname:

{

return make\_pair(t.var,1.0); // 省略1的情况

}

default:

throw -1;

}

}

// 处理 + 、 -

void term()

{

pair<string,ll> left = primary(); // 读左边值

if(eqm[n].find(left.first)==eqm[n].end())

eqm[n].insert(left);

else

throw -1;

Token t = ts.get(); // 获取运算符

while(true) {

switch(t.kind) {

case '+':

left = primary(); // 读右边值

if(eqm[n].find(left.first)==eqm[n].end())

eqm[n].insert(left);

else

throw -1;

t = ts.get();

break;

case '-':

left = primary(); // 读右边值

if(eqm[n].find(left.first)==eqm[n].end())

eqm[n].insert(make\_pair(left.first,-left.second));

else

throw -1;

t = ts.get();

break;

default:

ts.putback(t);

return;

}

}

}

// 处理=

void expression()

{

term(); // 读左边值

int flag=1;

Token t = ts.get(); // 获取运算符

switch(t.kind) {

case '=':

t = ts.get();

if(t.kind=='-')

flag=-1;

else if(t.kind=='+')

flag=1;

else

ts.putback(t);

t = ts.get();

if(t.kind!=number)

throw -1;

else

{

eqm[n].insert(make\_pair("const",flag\*t.value));

for(auto x:eqm[n])

{

if(x.first!="const")

vars.insert(x.first);

}

n++;

}

break;

default:

throw -1;

}

}

int main()

{

int i,j,k,row,er;

double coef,max\_coef;

while(1)

{

try{

ts.clear();

ts.readline();

expression();

}

catch(int j)

{

if(j==-1)

continue;

else

break;

}

}

if(n>0)

{

//make equations

cols=0;

for(iter=vars.begin();iter!=vars.end();iter++,cols++)

{

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

{

if(eqm[i].find(\*iter)!=eqm[i].end())

A[i][cols]=eqm[i].at(\*iter);

else

A[i][cols]=0;

}

}

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

A[i][cols]=eqm[i].at("const");

cols++;

//solve equations

int t=(n<(cols-1))?n:(cols-1);

er=0;

for(k=0;k<t-er;k++)

{

row=k;

max\_coef=fabs(A[k][k]);

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

if(max\_coef<fabs(A[i][k]))

{

max\_coef=fabs(A[i][k]);

row=i;

}

for(j=k;j<cols;j++)

swap(A[row][j],A[k][j]);

if(max\_coef>0.000001)

{

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

{

coef=1.0\*A[i][k]/A[k][k];

for(j=k;j<cols;j++)

A[i][j]-=A[k][j]\*coef;

}

}

else

{

er++;

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

swap(A[i][k],A[i][cols-1-er]);

k--;

}

}

int flag=1;

for(i=k;i<n;i++)

{

if(A[i][cols-1]!=0)

{

flag=0;

break;

}

}

if(flag==0)

printf("error2\n");//no answer

else if(k<cols-1)

printf("error3\n");//infinite answer

else

{

res[k-1]=A[k-1][cols-1]/A[k-1][k-1];

for(i=k-2;i>=0;i--)

{

res[i]=A[i][cols-1];

for(j=i+1;j<cols-1;j++)

res[i]-=A[i][j]\*res[j];

res[i]=res[i]/A[i][i];

}

for(iter=vars.begin(),i=0;i<k;iter++,i++)

printf("%.4lf ",(int)(res[i]\*10000)/10000.0);

}

}

else

printf("error1\n");

return 0;

}

# 维护数列（splay模板）



#include <iostream>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int INF=0x3f3f3f3f;

const int mm=500010;

struct node

{

int fa;

int son[2];

int val;///节点值

int sz;///子树节点数

int mx;///区间最大值，一个节点的区间是指以这个节点为根的子树的所有元素，所以不是固定的

int addtag;///区间增加

bool revtag;///区间翻转

bool sametag;///区间设置相同

int sum;///区间和

int qsum;///序列最大和

int lsum;///左子树序列最大和

int rsum;///右子树序列最大和

}tree[mm];//splay树

int root,tot1;///tot1当前节点总数

int s[mm],tot2;///因删除而空闲的节点

int a[mm];///保存输入的原始数据

int n,q;

#define get(x) (tree[tree[x].fa].son[1]==(x))

#define Key\_value tree[tree[root].son[1]].son[0]

#define t tree[x]

#define lson tree[x].son[0]

#define rson tree[x].son[1]

#define tlson tree[tree[x].son[0]]

#define trson tree[tree[x].son[1]]

void update(int x)//根据左右儿子的值update x节点，发生在左或者右儿子发生改变的时候

{

t.sz=tlson.sz+trson.sz+1;

t.sum=tlson.sum+trson.sum+t.val;

t.lsum=max(tlson.lsum,tlson.sum+t.val+max(0,trson.lsum));

t.rsum=max(trson.rsum,trson.sum+t.val+max(0,tlson.rsum));

t.qsum=max(0,tlson.rsum)+t.val+max(0,trson.lsum);

t.qsum=max(t.qsum,max(tlson.qsum,trson.qsum));

t.mx=max(t.val,tlson.mx);

t.mx=max(t.mx,trson.mx);

}

void change\_same(int x,int v)///标记当前节点x及其子节点设为相同的 V 值

{

if(!x)return;

t.val=t.mx=v;

t.sum=v\*t.sz;

t.lsum=t.rsum=t.qsum=max(v,v\*t.sz);

t.sametag=1;///标记

}

void change\_rev(int x)///标记当前节点x及其子节点逆转

{

if(!x)return;

swap(lson,rson);

swap(t.lsum,t.rsum);

t.revtag^=1;///标记

}

void change\_add(int x,int tag)///标记当前节点x及其子节点增加tag

{

if(!x)return;

int ts=tag\*t.sz;

t.val+=tag;t.mx+=tag;t.addtag+=tag;

t.sum+=ts;t.lsum+=ts;t.rsum+=ts;t.qsum+=ts;

}

void pushdown(int x)///已经访问节点x，将标记下传，下传前要计算当前标记对子节点的影响

{

if(t.sametag)

{

change\_same(lson,t.val);

change\_same(rson,t.val);

t.sametag=0;

}

if(t.revtag)

{

change\_rev(lson);

change\_rev(rson);

t.revtag=0;

}

if(t.addtag)

{

change\_add(lson,t.addtag);

change\_add(rson,t.addtag);

t.addtag=0;

}

}

inline void Rotate(int x)

{

int which=get(x);

int y=tree[x].fa;

int z=tree[y].fa;

pushdown(y);

pushdown(x);

tree[y].son[which]=tree[x].son[which^1];

tree[tree[y].son[which]].fa =y;

tree[x].son[which^1]=y;

tree[y].fa=x;

tree[x].fa=z;

if(z!=0)

{

tree[z].son[tree[z].son[1]==y]=x;

}

update(y);

update(x);

}

inline void Splay(int x,int goal)///将当前节点转到目标goal之下,例如goal=0,则表示转为根

{

int fa=tree[x].fa;

while(fa!=goal)

{

if(tree[fa].fa!=goal)

Rotate(get(x)==get(fa)?fa:x);

Rotate(x);

fa=tree[x].fa;

}

if(goal==0)root=x;

}

inline int Get\_Kth(int x,int k)///得到当前节点下第k个数据在tree中的位置

{

pushdown(x);

int tt=tlson.sz+1;

if(tt==k)return x;

if(tt>k)return Get\_Kth(lson,k);

else return Get\_Kth(rson,k-tt);

}

/\*\*\*初始化\*\*\*/

void NewNode(int &x,int fa,int v)///在节点father下新加一个值为v的节点，x返回该节点在tree数组中的位置

{

if(tot2)x=s[tot2--];

else x=++tot1;

t.fa=fa;

lson=rson=0;

t.val=t.sum=t.mx=t.lsum=t.rsum=t.qsum=v;

t.addtag=t.revtag=t.sametag=0;

t.sz=1;

}

void Build(int &x, int l, int r, int fa) {

if(l > r) return;

int mid = (l + r)/2;

NewNode(x,fa,a[mid]);

Build(lson, l, mid - 1, x);

Build(rson, mid + 1, r, x);

update(x);

}

void Init()

{

root=tot1=tot2=0;

tree[0].mx=tree[0].lsum=tree[0].rsum=tree[0].qsum=-INF; ///////////0/////////////

NewNode(root,0, -1); ///////////|/////////////

NewNode(tree[root].son[1],root, -1); ///////////root(-1)/////////////

for(int i=0;i<n;i++)scanf("%d",&a[i]); ///////////////\//////////

Build(Key\_value,0,n-1,tree[root].son[1]); /////////////////son[1](-1)//////////

update(tree[root].son[1]); ////////////////////////////////////

update(root); ////////////Key\_value/////////////////

}

/\*\*\*\*基本功能\*\*\*\*/

///在第x个数后插入tot个数

void Insert(int x,int tot)

{

for(int i=0;i<tot;i++)scanf("%d",&a[i]);

Splay(Get\_Kth(root,x+1),0);

Splay(Get\_Kth(root,x+2),root);

Build(Key\_value,0,tot-1,tree[root].son[1]);

update(tree[root].son[1]);

update(root);

}

void Erase(int x)

{

if(!x)return;

s[++tot2]=x;

Erase(lson);

Erase(rson);

}

///从第x个数开始连续删除tot个数

void Delete(int x,int tot)

{

Splay(Get\_Kth(root,x),0);

Splay(Get\_Kth(root,x+tot+1),root);

Erase(Key\_value);

tree[Key\_value].fa=0;

Key\_value=0;

update(tree[root].son[1]);

update(root);

}

///从第x个数连续开始的tot个数修改为c

void Make\_Same(int x,int tot,int c)

{

Splay(Get\_Kth(root,x),0);

Splay(Get\_Kth(root,x+tot+1),root);

change\_same(Key\_value,c);

update(tree[root].son[1]);

update(root);

}

///反转

void Reverse(int x,int tot)

{

Splay(Get\_Kth(root,x),0);

Splay(Get\_Kth(root,x+tot+1),root);

change\_rev(Key\_value);

update(tree[root].son[1]);

update(root);

}

///求和

int Get\_Sum(int x,int tot)

{

Splay(Get\_Kth(root,x),0);

Splay(Get\_Kth(root,x+tot+1),root);

return tree[Key\_value].sum;

}

///得到子序列最大和

int Get\_MaxSum(int x,int tot)

{

Splay(Get\_Kth(root,x),0);

Splay(Get\_Kth(root,x+tot+1),root);

return tree[Key\_value].qsum;

}

/\*

返回x代表的区间的最大值

int Get\_Max(int x)

{

return t.mx;

}

\*/

//中序打印调试

void Inorder(int x)

{

if(!x)return;

pushdown(x);

Inorder(lson);

printf("%d\*%d ",t.val,t.qsum);

Inorder(rson);

}

int main()

{

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

{

Init();

char op[20];

int x,y,z;

while(q--)

{

///printf("\n");

///Inorder(root);

scanf("%s",op);

if(op[0]=='I')

{

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

Insert(x,y);

}

else if(op[0]=='D')

{

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

Delete(x,y);

}

else if(op[0]=='M'&&op[2]=='K')

{

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

Make\_Same(x,y,z);

}

else if(op[0]=='R')

{

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

Reverse(x,y);

}

else if(op[0]=='G')

{

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

printf("%d\n",Get\_Sum(x,y));

}

else

{

printf("%d\n",Get\_MaxSum(1,tree[root].sz-2));

}

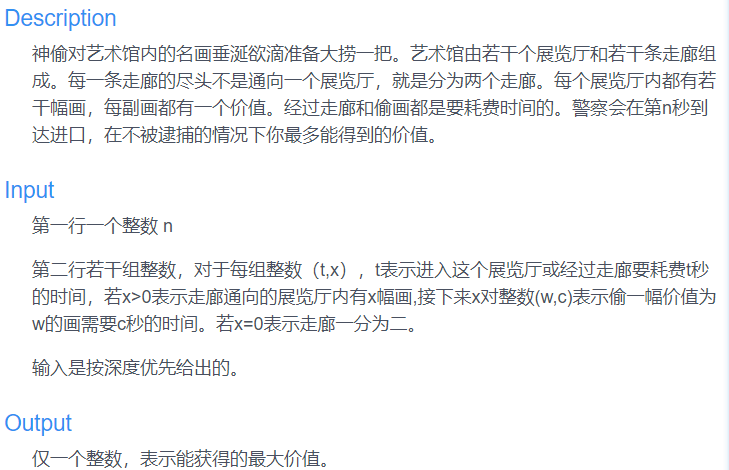
}

}

return 0;

}

# 偷天换日



#include<stdio.h>

const int N = 2010;

int a[N], b[N];

int f[5 \* N][5 \* N];

int s;

int max(int a, int b) {

if (a > b)

return a;

return b;

}

void read(int x) {

int t, k;

scanf("%d%d", &t, &k);

t = t \* 2;

if (k > 0) {

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

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

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

for (int j = s; j >= t + b[i]; j--)

f[x][j] = max(f[x][j - b[i]] + a[i], f[x][j]);

}

if (k == 0) {

read(2 \* x);

read(2 \* x + 1);

for (int i = s; i >= t; i--)

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

f[x][i] = max(f[x][i], f[2 \* x][j] + f[2 \* x + 1][i - j - t]);

}

}

}

int main() {

scanf("%d", &s);

s--;//在警察到达之前就要离开

read(1);

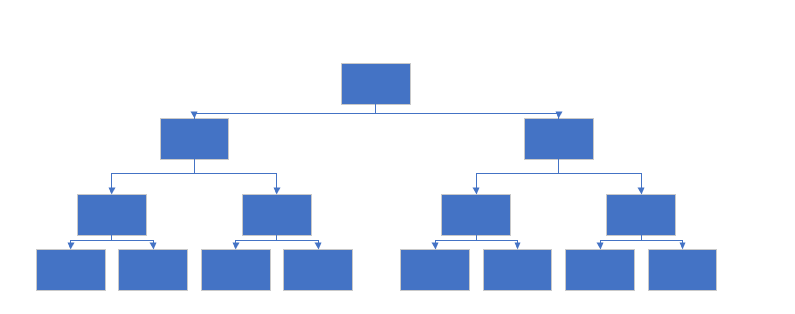
printf("%d\n", f[1][s]);

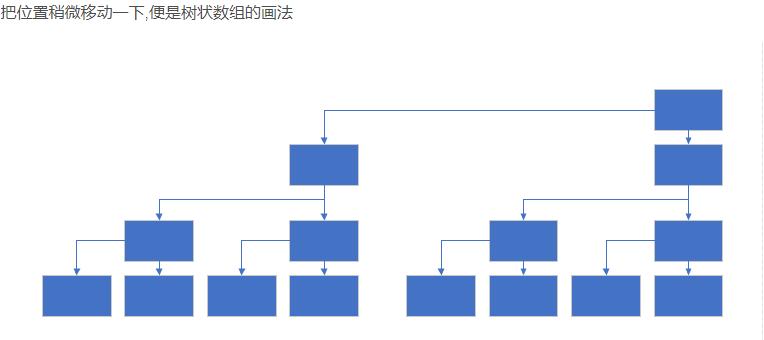
return 0;

}

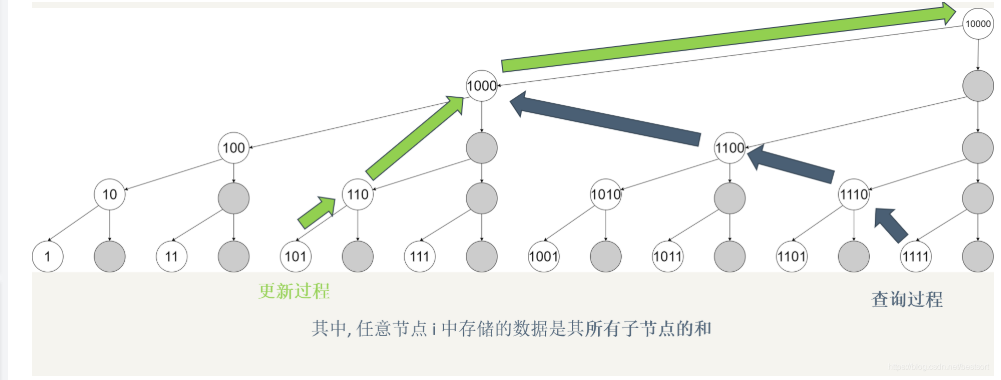
# 树状数组



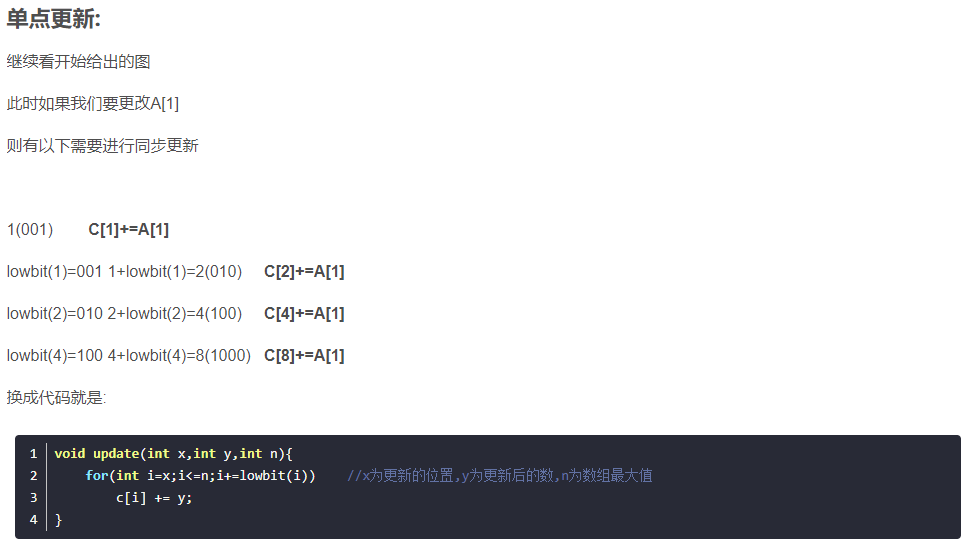


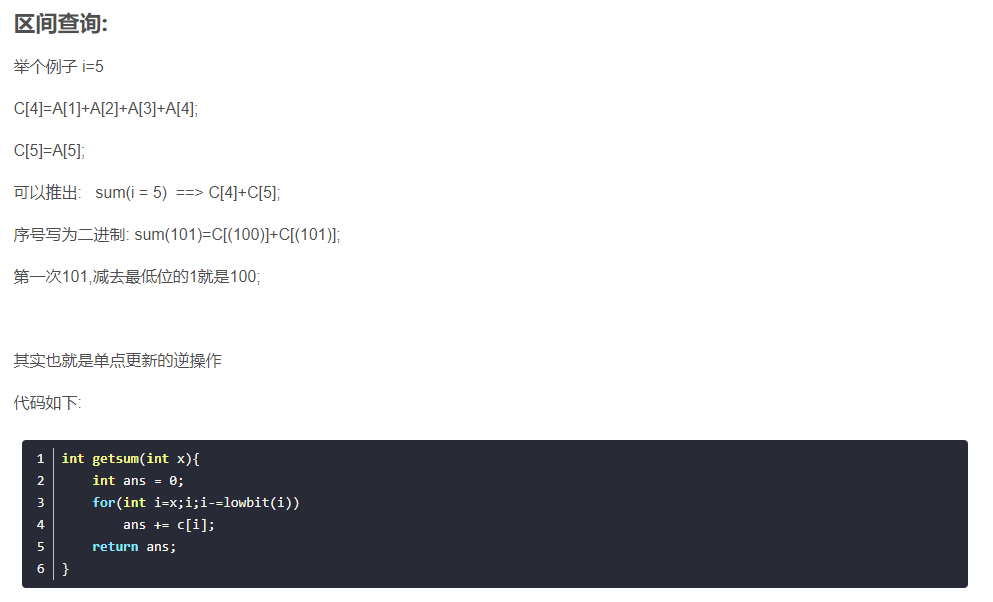




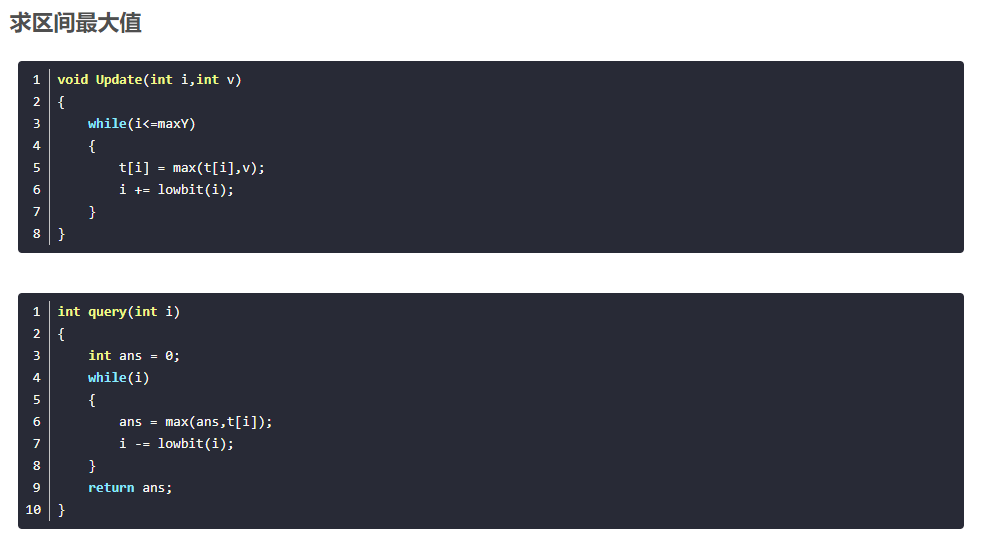


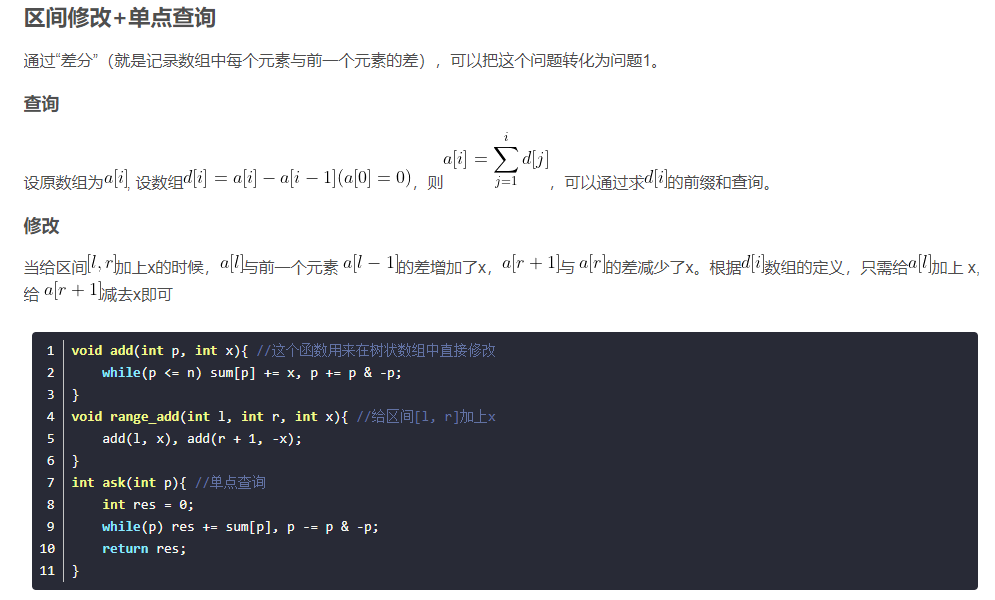


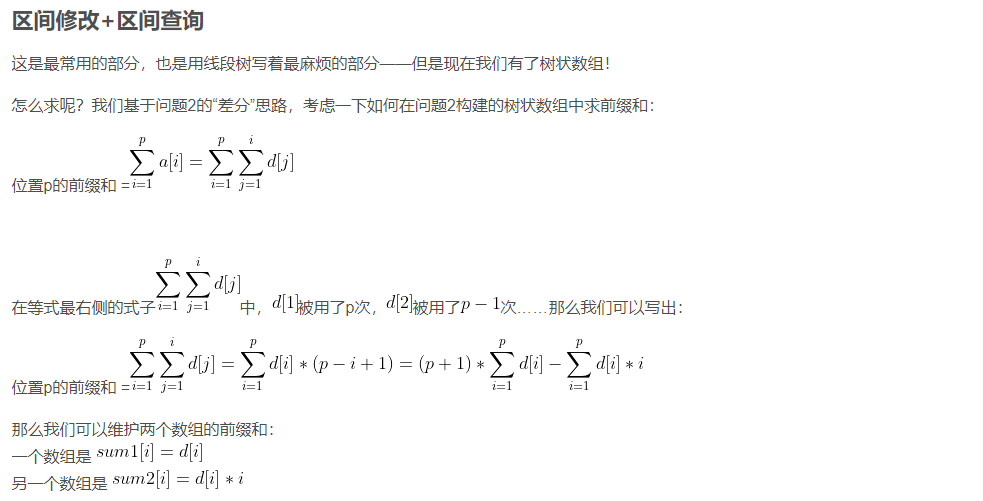


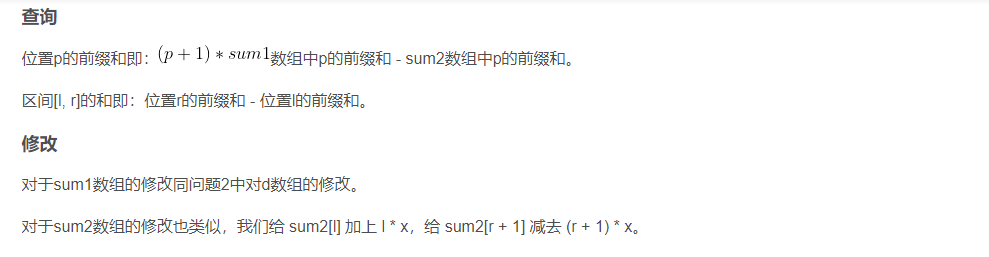


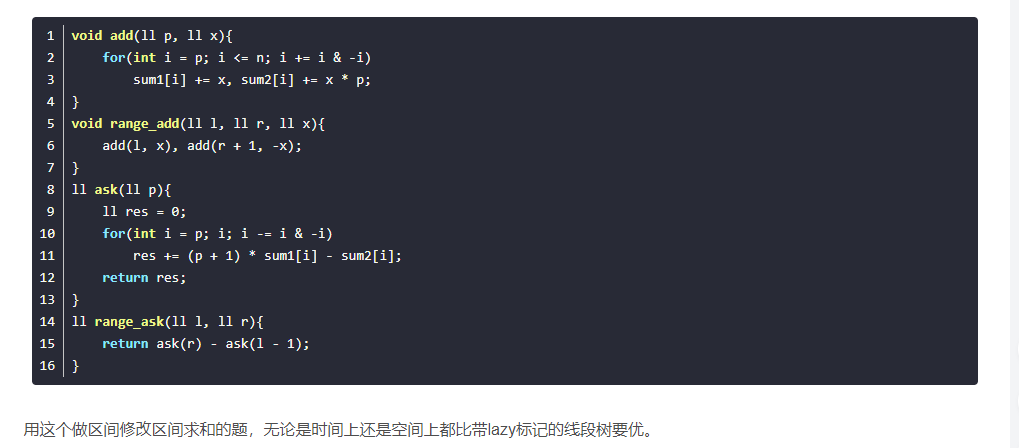


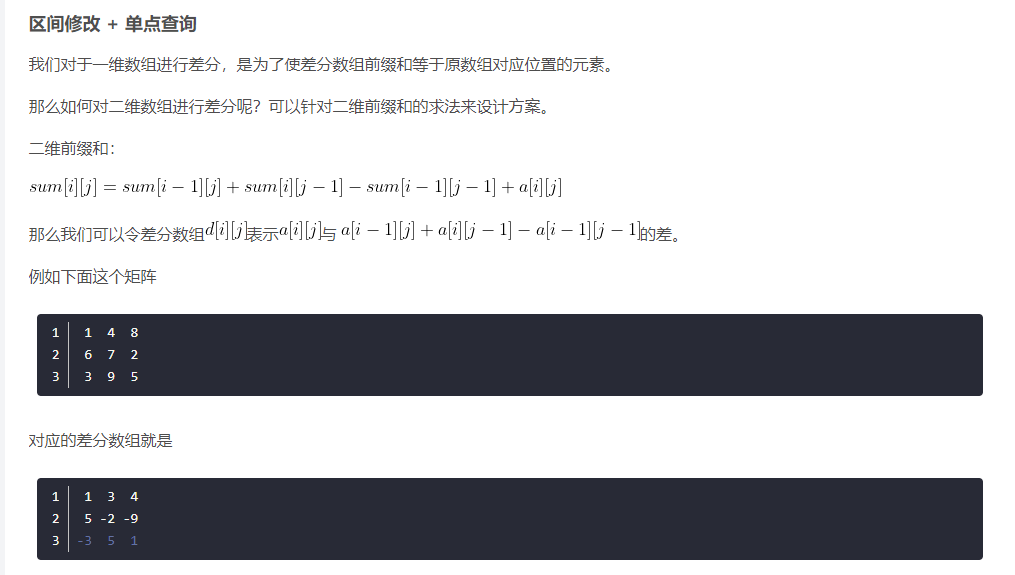


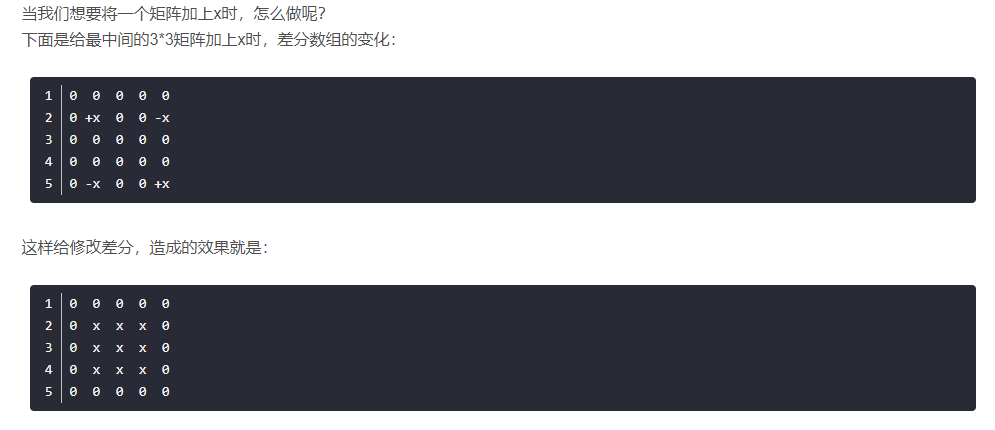


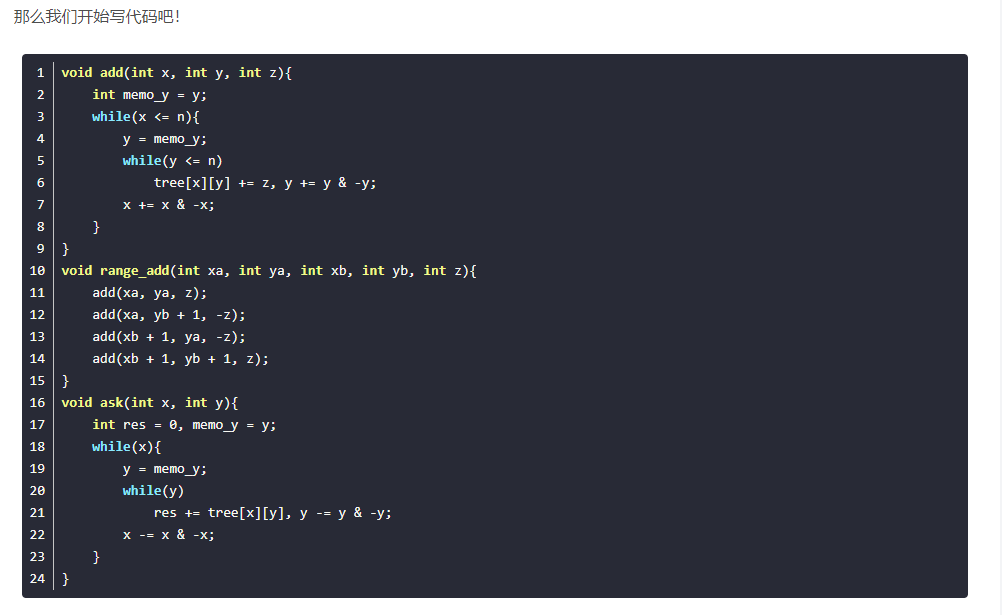


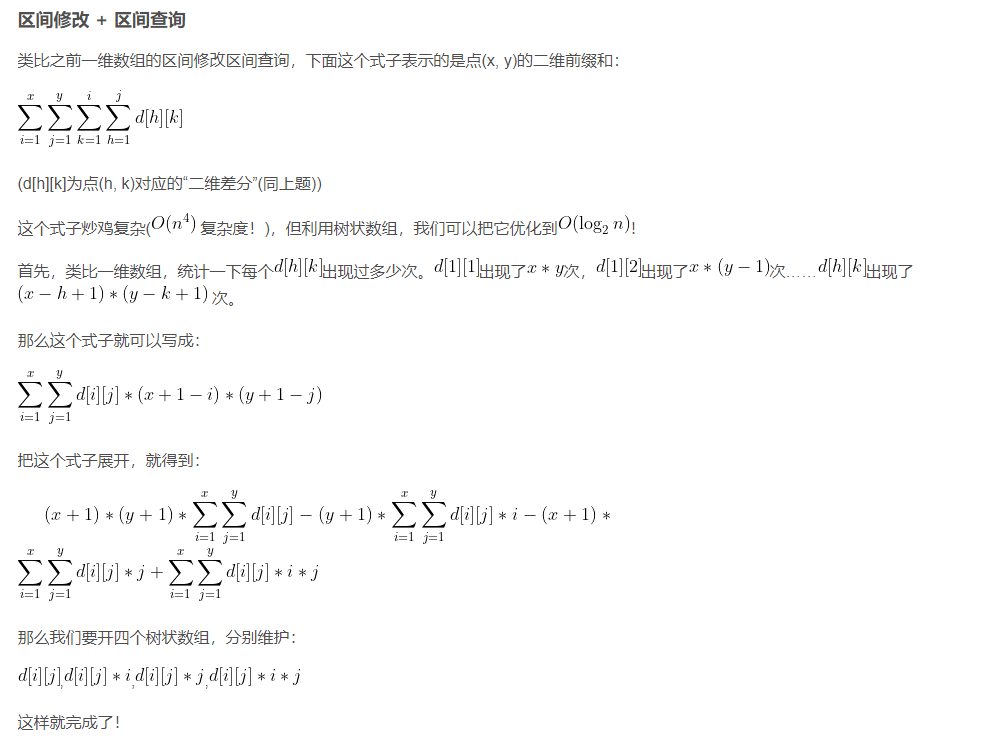


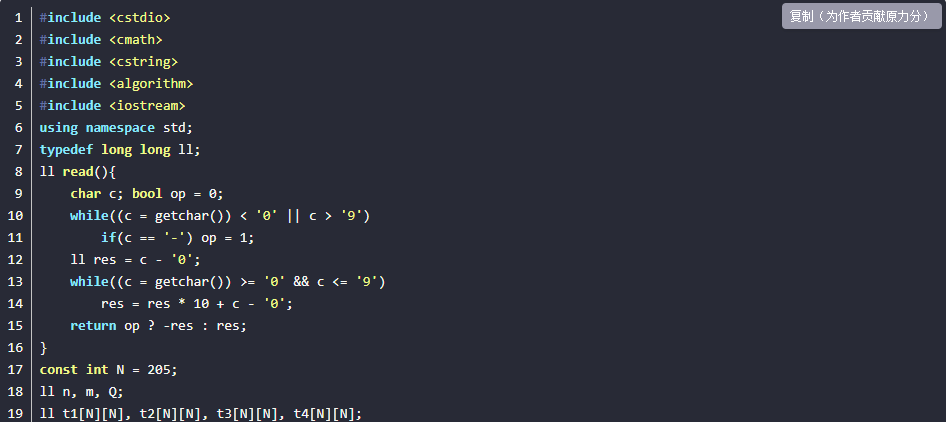


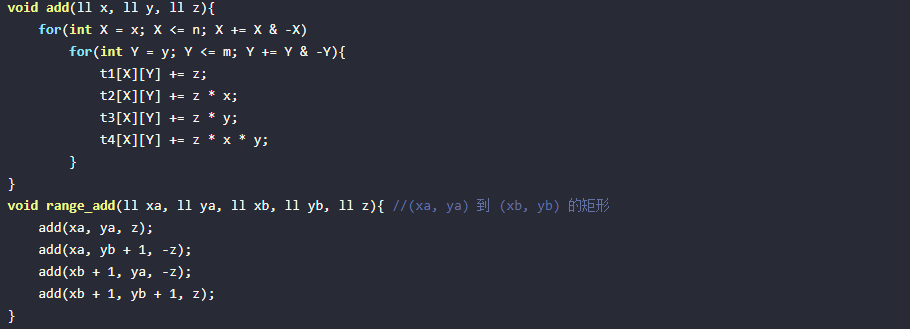


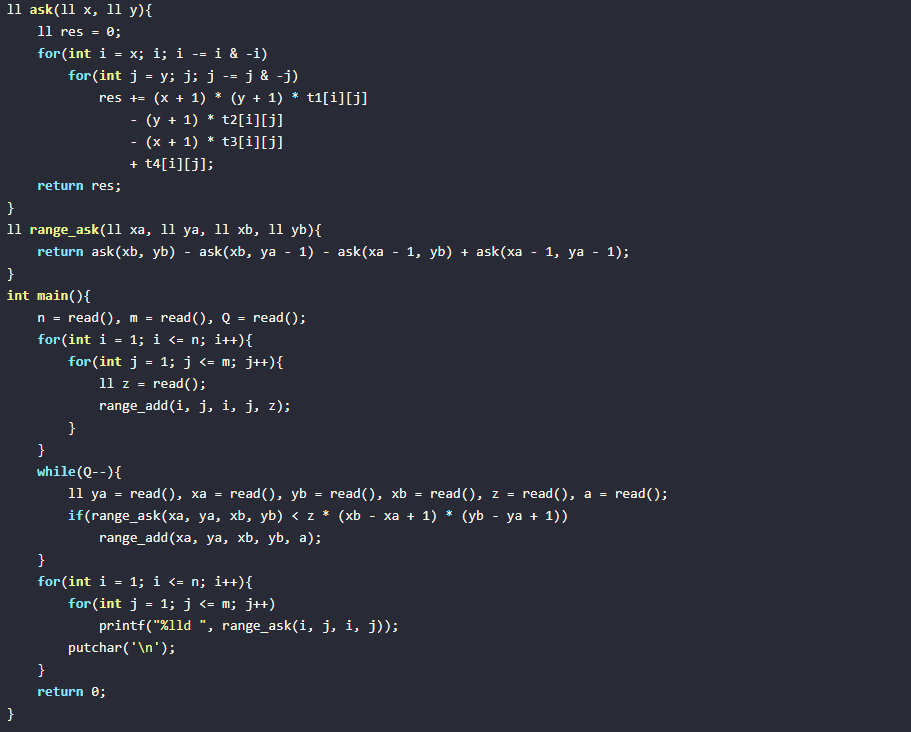












# 数学知识

### //组合数

//C(n,m) 在n个数中选m个的方案数

ll C[N][N];

void get\_C(int n)

{

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

{

C[i][i]=C[i][0]=1;

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

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

}

}

### //欧几里得算法

//(a,b)

ll gcd(ll a,ll b)

{

return b==0? a:gcd(b,a%b);

}

### //拓展欧几里得算法

//解同余方程 a\*x+b\*y = (a,b)

ll exgcd(ll a,ll b,ll& d,ll& x,ll& y)

{

if(!b) { d=a; x=1; y=0; }

else { exgcd(b,a%b,d,y,x); y-=x\*(a/b); }

}

### //逆元

//a\*inv(a,n) = 1 mod n

ll inv(ll a,ll n)

{

ll d,x,y;

exgcd(a,n,d,x,y);

return d==1? (x+n)%n:-1;

}

### //lucas定理

//计算较大的组合数

ll fac[N];

void get\_pre(int n)

{

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

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

}

ll C(ll n,ll m,ll mod)

{

if(n<m) return 0;

if(n<mod&&m<mod)

return fac[n]\*inv(fac[m],mod)%mod\*inv(fac[n-m],mod)%mod;

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

}

### //快速幂

//a^p % mod

ll pow(ll a,ll p,ll mod)

{

ll ans=1;

while(p)

{

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

a=(a\*a)%mod; p>>=1;

}

return ans;

}

### //中国剩余定理

//解线性同余方程组

//sigma{ ai\*(1-ai\*mi) } % M , ai\*mi+wi\*y=1

ll a[N],m[N];

ll china(int n)

{

ll M=1,d,x=0,y;

for(int i=1;i<=n;i++) M\*=m[i];

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

{

ll w=M/m[i];

exgcd(m[i],w,d,d,y);

x=(x+y\*w\*a[i])%M;

}

return (x+M)%M;

}

### //大步小步算法

//计算a^x=b mod n中的最小x

map<int,int> mp;

int BSGS(int a,int b,int n)

{

int m=sqrt(n)+1,e=1,i;

int v=inv(pow(a,m,n),n);

mp[e]=0;

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

{

e=(e\*m)%n;

if(!mp.count(e)) mp[e]=i;

}

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

{

if(mp.count(b)) return i\*m+mp[b];

b=(b\*v)%mod;

}

return -1;

}

### //快速筛法求素数表

int su[N],vis[N];

void get\_su(int n)

{

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

{

if(!vis[i]) su[++su[0]]=i;

for(int j=1;j<=su[0]&&i\*su[j]<=n;j++)

{

vis[i\*su[j]]=1;

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

}

}

}

### //欧拉函数

//phi(n)小于n的数中与n互素的数的个数

ll get\_phi(int n)

{

int m=sqrt(n)+1;

ll ans=n;

for(int i=2;i<=m;i++) if(n%i==0)

{

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

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

}

if(n>1) ans=ans/n\*(n-1);

return ans;

}

ll phi[N];

void get\_phi\_table(int n)

{

phi[1]=1;

for(int i=2;i<=n;i++) if(!phi[i])

{

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

{

if(!phi[j]) phi[j]=j;

phi[j]=phi[j]/i\*(i-1);

}

}

}

### //莫比乌斯函数

int mu[N],su[N],vis[N];

void get\_mu(int n)

{

mu[1]=1;

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

{

if(!vis[i]) mu[i]=-1,su[++su[0]]=i;

for(int j=1;j<=su[0]&&i\*su[j]<=n;j++)

{

vis[i\*su[j]]=1;

if(i%su[j]==0) mu[i\*su[j]]=0;

else mu[i\*su[j]]=-mu[i];

}

}

}

### //高斯消元

//解线性方程组

double a[N][N];

void gause(int n)

{

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

{

int r=i;

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

if(fabs(a[j][i])>fabs(a[r][i])) r=i;

for(int j=1;j<=n+1;j++) swap(a[i][j],a[r][j]);

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

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

a[k][j]-=a[k][i]/a[i][i]\*a[i][j];

}

for(int i=n;i;i--)

{

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

a[i][n+1]-=a[j][n+1]\*a[i][j];

a[i][n+1]/=a[i][i];

}

}

### //高精度

//高精度

const int maxn = 1e3+10;

const int base = 1e8;

const int wlen = 8;

int trans(char\* s,int st,int ed)

{

int x=0;

for(int i=st;i<ed;i++) x=x\*10+s[i]-‘0‘;

return x;

}

struct Bign

{

int len; ll N[maxn];

Bign() { len=1; memset(N,0,sizeof(N)); }

Bign(ll num) { \*this=num; }

Bign(const char\* s) { \*this=s; }

void print()

{

printf("%d",N[len-1]);

for(int i=len-2;i>=0;i--)

printf("%08d",N[i]);

puts("");

}

Bign operator = (const ll x)

{

ll num=x;

while(num>base)

{

N[len++]=num%base;

num/=base;

}

N[len++]=num;

return \*this;

}

Bign operator = (char\* s)

{

int L=strlen(s);

len=(L-1)/wlen+1;

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

{

int ed=L-i\*wlen;

int st=max(0,ed-wlen);

N[i]=trans(s,st,ed);

}

return \*this;

}

bool operator < (const Bign& B) const

{

if(len!=B.len) return len<B.len;

for(int i=len-1;i>=0;i--)

if(N[i]!=B.N[i]) return N[i]<B.N[i];

return 0;

}

bool operator <= (const Bign& B) const

{

return !(B<(\*this));

}

void clear()

{

while(len>1&&N[len-1]==0) len--;

}

Bign operator + (const Bign& B) const

{

Bign C;

C.len=max(len,B.len)+10;

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

{

C.N[i]+=N[i]+B.N[i];

C.N[i+1]+=C.N[i]/base;

C.N[i]%=base;

}

C.clear();

return C;

}

Bign operator - (Bign B)

{

Bign C;

C.len=max(len,B.len);

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

{

if(N[i]<B.N[i]) {

N[i+1]--;

N[i]+=base;

}

C.N[i]=N[i]-B.N[i];

}

C.clear();

return C;

}

Bign operator \* (const Bign& B) const

{

Bign C;

C.len=len+B.len;

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

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

C.N[i+j]+=N[i]\*B.N[j];

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

{

C.N[i+1]+=C.N[i]/base;

C.N[i]%=base;

}

C.clear();

return C;

}

Bign operator / (const Bign& B)

{

Bign C,F;

C.len=len;

for(int i=len-1;i>=0;i--)

{

F=F\*base;

F.N[0]=N[i];

while(B<=F)

{

F=F-B;

C.N[i]++;

}

}

C.clear();

return C;

}

Bign operator % (const Bign& B)

{

Bign r=\*this/B;

return \*this-r\*B;

}

}A,B;

### //矩阵乘法

//矩阵乘法

struct Mat

{

int r,c; ll N[maxn][maxn];

Mat(int r=0,int c=0)

{

this->r=r,this->c=c;

memset(N,0,sizeof(N));

}

Mat operator \* (const Mat& B) const

{

Mat C(r,B.c);

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

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

for(int k=0;k<c;k++)

C.N[i][j]=(C.N[i][j]+N[i][k]\*B.N[k][j])%mod;

return C;

}

Mat operator ^ (int p)

{

Mat ans(r,r),tmp=\*this;

for(int i=0;i<r;i++) ans.N[i][i]=1;

while(p)

{

if(p&1) ans=ans\*tmp;

tmp=tmp\*tmp; p>>=1;

}

return ans;

}

};