

NWU\_Geass

**薛晓满**

**景若芸**

**张宇**

解法总结

001.DP时用无法表示状态时再增加一维状态

002.注意数据范围long long

003.BFS起点的初始值不同时，但每次增加幅度均为1，可以不必使用优先队列(http://blog.csdn.net/idealism\_xxm/article/details/51565650 D题)：

对初始点按照距离升序排序，初始只将第一个点放入队列 然后每次更新队首的点之前，将距离小于等于队首的点的距离的初始点放入队列，这样就保证了队列中点的距离是升序的

004.状态离散的DP用dfs进行记忆化即可

005.DP需要维护的值是平方时可以同时考虑两个这样的组合，用DP同时维护一下保持同时性，从而使组合值为平方。

006.异或求最值时，可以想到按位贪心

007.DP可用维护前缀和或区间最值等降低时间复杂度

008.区间大小固定时可以使用滑动窗口技术降低时间复杂度

Idealism

**【AC自动机】**

**【RMQ】无修改区间最值（倍增思想也可用于求LCA等）**

**【博弈论】K倍动态减法游戏**

**【后缀数组】（DC3 和 倍增）**

**【划分树】区间第k大**

**【可持久化线段树】区间第k大**

**【树套树】带修改区间第k大**

**【分治】平面最近点对**

**【树状数组】区间最值**

**【网络流】（Dinic）**

**【最小割】（Stoer Wagner） O(n^3)**

**【组合数取模】**

**AC自动机**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXNODE=10000\*50+7;

int T,n;

int q[MAXNODE],head,tail;

char s[1000005];

struct Trie {

int nxt[MAXNODE][26],fail[MAXNODE],ed[MAXNODE];//数组模拟

int l;

const static int root=0;

Trie() {

clear();

}

int newNode() {//取一个结点

for(int i=0;i<26;++i) {

nxt[l][i]=-1;

}

ed[l]=0;//ed[l]表示从根到当前结点的模式串出现的次数

return l++;

}

void insert(char \*p) {//插入一个模式串

int cur=root;

while((\*p)!='\0') {//当模式串未结束时，继续循环

if(nxt[cur][(\*p)-'a']==-1) {//如果当前结点没有指向\*p字符的结点，则分配一个

nxt[cur][(\*p)-'a']=newNode();

}

cur=nxt[cur][(\*p)-'a'];//cur移动到\*p字符的结点

++p;

}

++ed[cur];//cur结点处匹配上的模式串+1

}

void build() {//建立fail指针

int cur,i;

head=tail=0;

fail[root]=root;//根结点的fail指针指向自己

for(int i=0;i<26;++i) {

if(nxt[root][i]==-1) {//如果根结点无i子结点，则让i子结点指向自己

nxt[root][i]=root;

}

else {//如果根结点有i子结点，则i子结点匹配失败后会转向根

fail[nxt[root][i]]=root;

q[tail++]=nxt[root][i];

}

}

while(head!=tail) {

cur=q[head++];

for(int i=0;i<26;++i) {

if(nxt[cur][i]==-1) {//如果当前结点无i子结点

nxt[cur][i]=nxt[fail[cur]][i];//当前结点的i子结点指针 指向 当前结点fail指针指向的结点的i子结点

}

else {

fail[nxt[cur][i]]=nxt[fail[cur]][i];;//当前结点i子结点的fail指针 指向 当前结点fail指针指向的结点的i子结点

q[tail++]=nxt[cur][i];

}

}

}

}

int query(char \*p) {

int cur=root,ans=0,tmp;

while((\*p)!='\0') {

cur=nxt[cur][(\*p)-'a'];

tmp=cur;

while(tmp!=root) {//遍历与当前形成的串有公共后缀的串

ans+=ed[tmp];

ed[tmp]=0;//由于只需要统计出现了关键词，所需需要置0，防止重复计算

tmp=fail[tmp];//转到下一个后缀串

}

++p;

}

return ans;

}

void clear() {

l=root;

newNode();//分配一个结点做根

}

}ac;

int main() {

scanf("%d",&T);

while(T--) {

ac.clear();

scanf("%d",&n);

while(n--) {

scanf("%s",s);

ac.insert(s);

}

ac.build();

scanf("%s",s);

printf("%d\n",ac.query(s));

}

return 0;

}

**RMQ**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN=100005;

const int INF = 0x3f3f3f3f;

int n, m;

/\*

RMQ：无修改求区间最值【本模版是求最小值，求最大值只需将min函数换成max函数即可】

【注意】求最值的数组应该存入dp[i][0]【第一维表示区间左端点，第二维表示区间长度2^0 = 1】

\*/

struct RMQ {

int log\_2[MAXN];

int dp[MAXN][19];

RMQ() {//预处理log

log\_2[0] = -1;

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

log\_2[i] = log\_2[i >> 1] + 1;

}

}

void init() {

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

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

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

}

}

}

int query(int l, int r) {

int k = log\_2[r - l + 1];

return min(dp[l][k],dp[r - (1 << k) + 1][k]);

}

}rmq;

int binSearch(int l, int r, int key) {//二分找到第一个小于等于key的下标

int mid;

while(l <= r) {

mid = (l + r) >> 1;

if(rmq.query(l, mid) <= key) {

r = mid - 1;

}

else {

l = mid + 1;

}

}

return l;

}

int ans, L, R, indx;

int main() {

int T;

scanf("%d", &T);

while(T-- >0) {

scanf("%d", &n);

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

scanf("%d", &rmq.dp[i][0]);

}

rmq.init();

scanf("%d", &m);

while(m-- >0) {

scanf("%d%d", &L, &R);

ans = rmq.dp[L][0];

while(true) {

indx = binSearch(L + 1, R, ans);

if(indx > R || rmq.dp[indx][0] > ans) {

break;

}

ans %= rmq.dp[indx][0];

L = indx;

}

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

}

}

return 0;

}

**【博弈论】K倍动态减法游戏**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

/\*

k倍动态减法游戏

参数：

n 表示初始石子数(n>=2)

k 表示下次取的最大值是本次取的k倍

ans 表示先手必胜的情况下，先手需要拿的石子数

返回值：

true 表示先手必胜

false 表示先手必败

难以理解，已加入模版库

\*/

const int MAXK=748553;//k=1e5时，对应的最大数列长度

long long a[MAXK],b[MAXK];//a为构造的数列；b为辅助数列，b[i]表示a[0..i]能构造的最大的数

bool K\_Dynamic\_Minus\_Game(long long n,long long k,long long& ans) {

int i=0,j=0;

a[0]=b[0]=1;

while(a[i]<n) {

++i;

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

while(k\*a[j+1]<a[i]) {//找到最大的j，使得k\*a[j]<a[i]

++j;

}

if(k\*a[j]<a[i]) {//如果存在最大的j，使得k\*a[j]<a[i]，则b[i]=b[j]+a[i];

b[i]=b[j]+a[i];

}

else {

b[i]=a[i];

}

}

if(n==a[i]) {//如果n在数列a中，则先手必败

return false;

}

while(n>0) {//寻找构成n的a数列中最小的一个

if(n>=a[i]) {

n-=a[i];

ans=a[i];

}

--i;

}

return true;

}

int n,k;

long long ans;

int main() {

int T,kase=0;

scanf("%d",&T);

while(T-->0) {

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

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

if(K\_Dynamic\_Minus\_Game(n,k,ans)) {

printf("%I64d\n",ans);

}

else {

printf("lose\n");

}

}

return 0;

}

**后缀数组**

**DC3算法 O(n)**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN=1000005 \* 3;

/\*

\*suffix array

\***DC3算法 O(n)** //【得开3倍字符串长度的空间】

\*build\_SA( ,sa, len+1, ); //tmp[len]手动赋值为0

\*getHeight( ,len);

\*例如：

\*n = 8;

\*num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ };注意num最后一位为0，其他大于0

\*rank[] = { 4, 6, 8, 1, 2, 3, 5, 7, 0 };rank[0~n-1]为有效值，rank[n]必定为0无效值

\*sa[] = { 8, 3, 4, 5, 0, 6, 1, 7, 2 };sa[1~n]为有效值，sa[0]必定为n是无效值

\*height[]= { 0, 0, 3, 2, 3, 1, 2, 0, 1 };height[2~n]为有效值

\*

\*/

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

int wsf[MAXN],wa[MAXN],wb[MAXN],wv[MAXN],sa[MAXN],rnk[MAXN],height[MAXN],f[MAXN];

int c0(int \*r,int a,int b) {

return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];

}

int c12(int k,int \*r,int a,int b) {

if(k==2) return r[a]<r[b]||(r[a]==r[b]&&c12(1,r,a+1,b+1));

else return r[a]<r[b]||(r[a]==r[b]&&wv[a+1]<wv[b+1]);

}

void sort(int \*r,int \*a,int \*b,int n,int m) {

int i;

for(i=0; i<n; i++) wv[i]=r[a[i]];

for(i=0; i<m; i++) wsf[i]=0;

for(i=0; i<n; i++) wsf[wv[i]]++;

for(i=1; i<m; i++) wsf[i]+=wsf[i-1];

for(i=n-1; i>=0; i--) b[--wsf[wv[i]]]=a[i];

return;

}

void build\_SA\_DC3(int \*r,int \*SA,int n,int m) {

int i,j,\*rn=r+n,\*san=SA+n,ta=0,tb=(n+1)/3,tbc=0,p;

r[n]=r[n+1]=0;

for(i=0; i<n; i++) if(i%3!=0) wa[tbc++]=i;

sort(r+2,wa,wb,tbc,m);

sort(r+1,wb,wa,tbc,m);

sort(r,wa,wb,tbc,m);

for(p=1,rn[F(wb[0])]=0,i=1; i<tbc; i++)

rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;

if(p<tbc) build\_SA\_DC3(rn,san,tbc,p);

else for(i=0; i<tbc; i++) san[rn[i]]=i;

for(i=0; i<tbc; i++) if(san[i]<tb) wb[ta++]=san[i]\*3;

if(n%3==1) wb[ta++]=n-1;

sort(r,wb,wa,ta,m);

for(i=0; i<tbc; i++) wv[wb[i]=G(san[i])]=i;

for(i=0,j=0,p=0; i<ta && j<tbc; p++)

SA[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];

for(; i<ta; p++) SA[p]=wa[i++];

for(; j<tbc; p++) SA[p]=wb[j++];

return;

}

void getHeight(int s[],int len) {

int i,j,k=0;//k表示最长公共前缀的长度

for(i=0;i<=len;++i) {//求rnk数组

rnk[sa[i]]=i;

}

for(i=0;i<len;++i) {//计算排名为i的后缀的height

if(k!=0) {//如果suffix(i-1)与其前一名的后缀串的最长公共前缀长度不为0

--k;

}

j=sa[rnk[i]-1];//j表示suffix(i)前一名的后缀串的起始点

while(s[i+k]==s[j+k]) {//寻找suffix(i)和suffix(j)的最长公共前缀长度

++k;

}

height[rnk[i]]=k;

}

}

int tmp[MAXN],len,T,cnt;

int lcp[MAXN];

char s[MAXN];

int getAns() {

lcp[rnk[0]]=len;

for(int i=rnk[0]-1;i>=0;--i) {

lcp[i]=min(lcp[i+1],height[i+1]);

}

for(int i=rnk[0]+1;i<len;++i) {

lcp[i]=min(lcp[i-1],height[i]);

}

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

if(len%i==0&&lcp[rnk[i]]==len-i) {

return len/i;

}

}

return 1;

}

int main() {

while(scanf("%s",s),s[0]!='.') {

len=strlen(s);

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

tmp[i]=s[i];

}

tmp[len]=0;

build\_SA\_DC3(tmp,sa,len+1,128);

getHeight(tmp,len);

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

}

return 0;

}

**倍增算法 O(n\*logn)**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN=20005;

/\*

\*suffix array

\***倍增算法 O(n\*logn)**

\*build\_SA( , n, );

\*getHeight( ,n);

\*例如：

\*n = 8;

\*num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ };注意num最后一位为0，其他大于0

\*rank[] = { 4, 6, 8, 1, 2, 3, 5, 7, 0 };rank[0~n-1]为有效值，rank[n]必定为0无效值

\*sa[] = { 8, 3, 4, 5, 0, 6, 1, 7, 2 };sa[1~n]为有效值，sa[0]必定为n是无效值

\*height[]= { 0, 0, 3, 2, 3, 1, 2, 0, 1 };height[2~n]为有效值

\*

\*/

int c[MAXN];//求SA数组时计数排序的辅助数组

int sa[MAXN],rnk[MAXN],height[MAXN];//sa[i]表示排名为i的是以第sa[i]个字符起始的后缀串；rnk[i]表示以第i个字符起始的后缀串的排名；height[i]表示suffix(i)和suffix(i-1)的最长公共前缀的长度

void build\_SA(int s[],int len,int mx) {//原字符串为s[0..(len-1)] 【为了方便后面的下标，需要提前将字符串复制到int数组中】，最大值小于mx

int i,j,p=0;//i,j为循环变量；p为下次最大值

int \*x=rnk,\*y=height;//x,y分别指向辅助数组rnk,height【此处由于rnk个height数组都没用，所以暂时用来代替辅助数组】；方便交换数组t1,t2；x数组表示每次排序后的真实rank（可能会相等），y数组表示按第二关键字排序后的rank

s[len++]=0;//为了方便计算，在原字符串末尾加一个0

//第一次用计数排序给每个长度为1的字符串排序

for(i=0;i<mx;++i) {//c数组是计数排序的辅助数组

c[i]=0;

}

for(i=0;i<len;++i) {//统计每个字符出现的次数

++c[x[i]=s[i]];//x数组相当于每个字符的rank值，但rank值只是用来判断rank值是否相等的，所以无需求出真实的rank值，以其字符代替，即相同字符的rank值相等

}

for(i=1;i<mx;++i) {//进行完本次循环后，c[i]就是排序完成后，字符s[i]最后一个排名的下一个排名

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

}

for(i=len-1;i>=0;--i) {//从后统计参与排名的字符，保证相同字符排在前面的排名高

sa[--c[x[i]]]=i;

}

for(j=1;p<len;j<<=1) {//j为当前字符串的长度；当排名达到len时，则所有后缀串的排名均不相同，此时可以停止倍增

//利用上一次求得的SA，直接对第二关键字排序

for(p=0,i=len-j;i<len;++i) {//起始点在[len-j,len)中的后缀字符串长度均小于j，即无第二关键字，所以其第二关键字最小

y[p++]=i;//此时i表示起始点

}

for(i=0;i<len;++i) {//起始点在[0,j)中的长度为j的字符串，前面没有长度大于等于j的字符串，即其不会作为第二关键字，所以不算入排序

if(sa[i]>=j) {//此时i表示排名

y[p++]=sa[i]-j;//起点在[j,len)中的长度为j（后j-1个字符串小于j）的字符串 作为起点在[0,len-j)中的长度为2\*j（后j个字符串小于2\*j）的第二关键字，并对其排序

}

}//循环结束后y[z]表示排名为z长度为2\*j（后j个字符串小于2\*j）的字符串的起始点

//用计数排序对第一关键字进行排序

for(i=0;i<mx;++i) {

c[i]=0;//计数排序辅助数组清零

}

for(i=0;i<len;++i) {//统计每个rank值出现的次数

++c[x[y[i]]];

}

for(i=1;i<mx;++i) {

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

}

for(i=len-1;i>=0;--i) {//从后统计参与排名的字符，保证相同字符串排在前面的排名高

sa[--c[x[y[i]]]]=y[i];

}

swap(x,y);//交换指针以实现整个数组的交换

p=1;

x[sa[0]]=0;

for(i=1;i<len;++i) {//对x数组进行修正（前面没看到，导致怎么看程序都无法理解与图示的倍增过程那个rank值不一样）

x[sa[i]]=(y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+j]==y[sa[i]+j]?p-1:p++);//rank相邻的两个字符串的第一关键字和第二关键字都一样，则其rank一样，否则rank+1

}

mx=p;//更新下次计数排序的最大值

}

}

void getHeight(int s[],int len) {

int i,j,k=0;//k表示最长公共前缀的长度

for(i=0;i<=len;++i) {//求rnk数组

rnk[sa[i]]=i;

}

for(i=0;i<len;++i) {//计算排名为i的后缀的height

if(k!=0) {//如果suffix(i-1)与其前一名的后缀串的最长公共前缀长度不为0

--k;

}

j=sa[rnk[i]-1];//j表示suffix(i)前一名的后缀串的起始点

while(s[i+k]==s[j+k]) {//寻找suffix(i)和suffix(j)的最长公共前缀长度

++k;

}

height[rnk[i]]=k;

}

}

int tmp[MAXN],len;

char s[MAXN];

int main() {

scanf("%s",s);

for(len=0;s[len]!='\0';++len) {

tmp[len]=s[len];

}

build\_SA(tmp,len,128);

getHeight(tmp,len);

return 0;

}

**划分树**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

/\*

划分树 -- 查询区间第k小【查询[l,r]的第k大，即查询[l,r]的第r-l+1-k+1=r-l+2-k小】

初始化：tr[0][1..n]为原序列，sorted[1..n]为排序后的序列【第k小升序，第k大降序】

\*/

const int MAXN=100007;

const int MAXLV=19;//100 000个数用19层够了

int sorted[MAXN];//sorted[1..n]为排序后序列

int cnt[MAXLV][MAXN];//cnt[i][j]统计第i层，[sta(j),j]内分到左子树的个数【sta(j)表示当前层时，j所在区间的起始左端点】

int tr[MAXLV][MAXN];//tr[i]表示第i层的序列，tr[0][1..n]为原序列，

void build(int lv,int l,int r) {

if(l==r) {

return ;

}

int i,mid=(l+r)>>1,lmx=mid-l+1,lindx=l,rindx=mid+1;//lmx表示第lv层[l,r]中会分到左子树的个数，lindx,rindx分表表示左子树和右子树当前该在那个位置存储

for(i=l;i<=r;++i) {

if(sorted[i]<sorted[mid]) {//比中位数小的一定会被分到左子树

--lmx;

}

}

for(int i=l;i<=r;++i) {

cnt[lv][i]=(i==l?0:cnt[lv][i-1]);//初始化本层本区间的cnt，cnt[lv][i]表示[l,i]被分到左子树的个数

if(tr[lv][i]<sorted[mid]) {//如果这个数小于中位数，则划分到左子树中

++cnt[lv][i];

tr[lv+1][lindx++]=tr[lv][i];

}

else if(tr[lv][i]==sorted[mid]) {//如果这个数等于中位数，根据左子树是否放满决定其划分到的子树

if(lmx==0) {//如果左子树已放满，则其被划分到右子树

tr[lv+1][rindx++]=tr[lv][i];

}

else {

--lmx;

++cnt[lv][i];

tr[lv+1][lindx++]=tr[lv][i];

}

}

else {//如果这个数大于中位数，则划分到右子树

tr[lv+1][rindx++]=tr[lv][i];

}

}

build(lv+1,l,mid);

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

}

int query(int lv,int l,int r,int ql,int qr,int k) {//ql,qr表示查询的区间左右端点

if(l==r) {

return tr[lv][l];

}

int tmp,ttmp,mid=(l+r)>>1;//tmp表示[l,ql)内被划分到左子树的个数，ttmp表示[ql,qr]内被划分到左子树的个数

tmp=(l==ql?0:cnt[lv][ql-1]);//特殊处理查询左端点与当前区间左端点相等时的情况

ttmp=cnt[lv][qr]-tmp;

if(k<=ttmp) {//如果查询区间内被分到左子树的个数大于等于k，则第k小数被分到左子树中

//左子树中[l,l+tmp)【即前tmp个】均不是当前查询的区间内的数，所以更新查询区间左端点为 l+tmp

//左子树中(l+tmp+ttmp-1,mid]均不是当前查询区间内的数，所以更新查询区间右端点为 l+tmp+ttmp-1

return query(lv+1,l,mid,l+tmp,l+tmp+ttmp-1,k);

}

//第k小数被分到右子树中，更新查询区间第k-tmp小数

//右子树中[mid+1,mid+1+ ((ql-l)-tmp) )【即前(ql-l)-tmp个】均不是当前查询区间内的数，所以更新查询区间左端点为 mid+1+((ql-l)-tmp)

//右子树中(mid+1+ ((ql-l)-tmp) + (qr-ql+1-tmp) -1,r]均不是当前查询区间内的数，所以更新查询区间右端点为 mid+1+ ((ql-l)-tmp) + (qr-ql+1-tmp) -1=mid+1+qr-l-tmp-ttmp

return query(lv+1,mid+1,r,mid+1+((ql-l)-tmp),mid+1+qr-l-tmp-ttmp,k-ttmp);

}

int n,m,l,r,k;

int main() {

while(2==scanf("%d%d",&n,&m)) {

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

scanf("%d",sorted+i);

tr[0][i]=sorted[i];

}

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

build(0,1,n);

while(m-->0) {

scanf("%d%d%d",&l,&r,&k);

printf("%d\n",query(0,1,n,l,r,k));

}

}

return 0;

}

**可持久化线段树**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

/\*

可持久化线段树 -- 查询区间第k小【查询区间第k大，即查询区间第 r-l+1-k+1=r-l+2-k 小】

初始化：a[1..n]为原序列，sorted[1..n]为原序列的备份，然后调用build就行

hash后的值为当前数第一次出现的下标，所以存在部分下标无对应的值，但相对简单

\*/

const int MAXN=100007;

const int LOGMAXN=19;//100 000个数用19就够了

//可持久化线段树是权值线段树，即区间就是权值，而与原序列的区间无关

//第i棵数是原序列区间[1,i]建立的权值线段树

struct Node {//为防止MLE，线段树结点不再保存区间端点，而是动态求得

int lson,rson;

int sum;//sum表示各权值出现次数之和

}tr[MAXN\*LOGMAXN];

int n;

int a[MAXN],sorted[MAXN];//a[1..n]表示原序列，sorted[1..n]表示排序后的序列

int root[MAXN],cnt;//cnt表示当前已用的结点数

void inser(int &cur,int ori,int val,int l=1,int r=n) {//cur表示当前线段树的编号【调用是传入root[i]即可】，ori传入root[i-1]即可，val是权值

cur=cnt++;//给当前结点分配一个新的结点

tr[cur]=tr[ori];

++tr[cur].sum;//插入了val后，新结点sum+1

if(l==r) {

return ;

}

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

if(val<=mid) {

inser(tr[cur].lson,tr[ori].lson,val,l,mid);

}

else {

inser(tr[cur].rson,tr[ori].rson,val,mid+1,r);

}

}

int query(int L,int R,int k,int l=1,int r=n) {//L,R分别表示查询区间左右端点代表的线段树的下标【调用是传入root[LL-1],root[RR]即可】

if(l==r) {

return l;

}

int tmp=tr[tr[R].lson].sum - tr[tr[L].lson].sum,mid=(l+r)>>1;//计算查询区间内前一半权值出现的次数

if(k<=tmp) {

return query(tr[L].lson,tr[R].lson,k,l,mid);

}

return query(tr[L].rson,tr[R].rson,k-tmp,mid+1,r);

}

void build() {//初始化并建立可持久化线段树

tr[0].lson=tr[0].rson=tr[0].sum=root[0]=0;//由于以后lson,rson,sum的值都不会更改（子结点还是自己），所以只用一个根结点就能表示一棵空数

cnt=1;

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

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

inser(root[i],root[i-1],lower\_bound(sorted+1,sorted+1+n,a[i])-sorted);//这样hash会有部分下标无对应的数，但相对简单

}

}

int m,l,r,k;

int main() {

int T;

scanf("%d",&T);

while(T-->0) {

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

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

scanf("%d",a+i);

sorted[i]=a[i];

}

build();

while(m-->0) {

scanf("%d%d%d",&l,&r,&k);

printf("%d\n",sorted[query(root[l-1],root[r],k)]);//注意返回的是下标

}

}

return 0;

}

**树套树**

#include <cstdio>

#include <cstring>

#include <algorithm>

#define lson (i<<1)

#define rson ((i<<1)|1)

using namespace std;

/\*

树套树 -- 查询区间第k大【查询区间第k小，只需调用注释的query函数即可】

初始化：root数组初始化为-1，cnt初始化为0

\*/

const int MAXN=100003;//外层线段树结点数，由于插入的c可正可负，所以要开两倍的大小

const int MAXM=MAXN\*17\*17;//内层线段树结点数，极限情况下外层线段树每个点都有一个区间线段树，每个区间线段树有logn个结点

//外层线段树为权值线段树，root[i]表示当前结点对应的区间线段树的根结点下标

int root[MAXN<<2];//为防止MLE，外层线段树不保存区间端点值，而是动态求得；

//内层线段树为区间线段树

struct InnerSegTree {//为防止MLE，内层线段树不保存区间端点值，而是动态求得

int sum,lazy;//永久化标记，不下放，查询时计入lazy即可

int lc,rc;

}tr[MAXM];

int n,L,R;//L,R表示 修改/查询 区间的左右端点

int cnt;//cnt表示内层线段树已分配的结点数

void inser(int &cur,int l=1,int r=n) {//内层线段树，树根为root[i]，给外层线段树第i个结点的[L,R]区间每个位置都+1

if(cur==-1) {//如果当前结点还未分配空间

cur=++cnt;

tr[cur].sum=tr[cur].lazy=0;

tr[cur].lc=tr[cur].rc=-1;//初始化

}

if(L<=l&&r<=R) {//如果当前区间已在需要修改的区间内

tr[cur].sum+=r-l+1;

++tr[cur].lazy;//当前区间的所有位置均+1，但不下放，而是通过永久标记记住，以减少空间消耗

return ;

}

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

if(mid>=L) {//如果修改的区间有部分在左子树

inser(tr[cur].lc,l,mid);

}

if(mid<R) {//如果修改的区间有部分在右子树

inser(tr[cur].rc,mid+1,r);

}

tr[cur].sum+=min(r,R)-max(l,L)+1;//【注意：此处也需要加上

}

void add(int i,int val,int l=-n,int r=n) {//外层线段树，给[L,R]内的每个位置插入一个数val

inser(root[i]);//给第i个结点的区间[L,R]的每个位置均+1

if(l==r) {//如果当前是叶子结点

return ;

}

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

if(val<=mid) {//如果当前权值出现在左子树

add(lson,val,l,mid);

}

else {

add(rson,val,mid+1,r);

}

}

unsigned int Count(int cur,int l=1,int r=n) {//内层线段树

if(L<=l&&r<=R) {//如果当前区间已在需要查询的区间内

return tr[cur].sum;

}

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

unsigned int res=0;

if(mid>=L) {//统计左边一部分的个数

res+=Count(tr[cur].lc,l,mid);

}

if(mid<R) {//统计右边一部分的个数

res+=Count(tr[cur].rc,mid+1,r);

}

return (min(r,R)-max(l,L)+1)\*tr[cur].lazy+res;//最后加上lazy标记与当前区间大小的乘积，即没有下放的个数

}

/\*

int query(int i,int k,int l=-n,int r=n) {//外层线段树，查询区间[L,R]第k小

if(l==r) {

return l;

}

unsigned int tmp=Count(root[lson]);

if(tmp>=k) {//如果左子树数字出现次数大于等于k，则第k小必定在左子树

return query(lson,k,l,(l+r)>>1);

}

return query(rson,k-tmp,((l+r)>>1)+1,r);

}

\*/

int query(int i,int k,int l=-n,int r=n) {//外层线段树，查询区间[L,R]第k大

if(l==r) {

return l;

}

unsigned int tmp=Count(root[rson]);//统计区间右边一部分的数字出现的次数

if(tmp>=k) {//如果右子树数字出现次数大于等于k，则第k大必定在右子树

return query(rson,k,((l+r)>>1)+1,r);

}

return query(lson,k-tmp,l,(l+r)>>1);

}

inline void init() {

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

cnt=0;

}

int m,ope,k;

int main() {

while(scanf("%d%d",&n,&m)==2) {

init();

while(m-->0) {

scanf("%d%d%d%d",&ope,&L,&R,&k);

if(ope==1) {

add(1,k);

}

else {

printf("%d\n",query(1,k));

}

}

}

return 0;

}

**平面最近点对**

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const double EPS=0.000001;

const double INF=100000000000;

struct Point {

double x,y;

bool operator < (const Point& a) const {

return x<a.x||(x==a.x&&y<a.y);

}

}p[100007],tmp[100007];

int n,cnt;

inline bool cmp(const Point& a,const Point& b) {

return a.y<b.y;

}

inline double getDis(const Point& a,const Point& b) {

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

double dfs(int l,int r) {

if(l==r) {//如果只有一个点

return INF;

}

if(l+1==r) {//如果只有两个点

return getDis(p[l],p[r]);

}

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

double delta=min(dfs(l,mid),dfs(mid+1,r)),ans=INF;

cnt=0;

for(int i=l;i<=r;++i) {//将[l,r]区间内与x=p[mid].x的距离小于delta的点加入

if(fabs(p[mid].x-p[i].x)<delta+EPS) {

tmp[cnt++]=p[i];

}

}

sort(tmp,tmp+cnt,cmp);

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

for(int j=i+1;j<cnt&&tmp[j].y-tmp[i].y<delta+EPS;++j) {

ans=min(ans,getDis(tmp[i],tmp[j]));

}

}

return min(delta,ans);

}

int main() {

while(scanf("%d",&n),n!=0) {

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

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

}

sort(p,p+n);

printf("%.2lf\n",dfs(0,n-1)/2);

}

return 0;

}

**树状数组**

#include <cstdio>

#include <cstring>

#include <algorithm>

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

using namespace std;

const int MAXN = 200003;

/\*

树状数组求最大值【求最小值时，将所有的max函数改为min函数即可】

n表示数组的大小，a[1~n]存储数

\*/

int n, cnt;

long long a[MAXN], mx[MAXN];

long long MOD, pre, num;

char cmd[3];

//初始化函数：若a全部给定，可以先调用这个函数，复杂度为O(nlogn)

void init() {

memset(mx, 0, sizeof(mx));

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

mx[i] = a[i];

for(int j = 1; j < lowbit(i); j <<= 1) {

mx[i] = max(mx[i], mx[i - j]);

}

}

}

//修改函数：将a[x]改为val，复杂度为O((logn)^2)

void modify(int x, long long val) {

a[x] = val;

while(x <= n) {

mx[x] = val;

for(int j = 1; j < lowbit(x); j <<= 1) {

mx[x] = max(mx[x], mx[x - j]);

}

x += lowbit(x);

}

}

//查询函数：查询区间[l,r]内的最大值，复杂度为O(logn)

long long query(int l, int r) {

long long ans = 0;

while(true) {

ans = max(ans, a[r]);

if(l == r) {

return ans;

}

for(--r; r - l >= lowbit(r); r -= lowbit(r)) {

ans = max(ans, mx[r]);

}

}

return ans;

}

int main() {

while(2 == scanf("%d%lld", &n, &MOD)) {

memset(mx, 0, sizeof(mx));

pre = cnt = 0;

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

scanf("%s%lld", cmd, &num);

if(cmd[0] == 'Q') {

printf("%lld\n", pre = query(cnt - num + 1, cnt));

}

else {

modify(++cnt, (num + pre) % MOD);

}

}

}

return 0;

}

**网络流**

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

#define opp(x) ((x)^1)

using namespace std;

const int MAXN=407;

const int MAXM=160007;

const int INF=0x3f3f3f3f;

struct Node {

int v,nxt;

int cap;

}edge[MAXM];

int STA,DES,num;

int fir[MAXN],lv[MAXN],cur[MAXN];//cur[MAXN]为当前弧优化需要的数组，为fir[MAXN]的副本

int q[MAXN],head,tail,uu,vv,ccap;

void init() {

num=0;

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

}

void addEdge(int u,int v,int cap) {

edge[num].v=v;

edge[num].cap=cap;

edge[num].nxt=fir[u];

fir[u]=num++;

edge[num].v=u;

edge[num].cap=0;//增加一条反向边，容量为0

edge[num].nxt=fir[v];

fir[v]=num++;

}

bool bfs() {//构建层次网络，一个点的层数就是源点到它最少经过的边数

memset(lv,0,sizeof(lv));

lv[STA]=1;

head=tail=0;

q[tail++]=STA;

while(head!=tail) {

uu=q[head++];

if(uu==DES) {

return true;

}

for(int i=fir[uu];i!=-1;i=edge[i].nxt) {

vv=edge[i].v;

ccap=edge[i].cap;

if(lv[vv]==0&&ccap!=0) {

lv[vv]=lv[uu]+1;

q[tail++]=vv;

}

}

}

return false;

}

int dfs(int u,int mx) {

if(u==DES) {

return mx;

}

int res=0,cap,v;

for(int &i=cur[u];i!=-1;i=edge[i].nxt) {//对弧的下标引用，进行当前弧优化

v=edge[i].v;

cap=edge[i].cap;

if(lv[u]+1==lv[v]&&cap!=0) {

cap=dfs(v,min(mx-res,cap));

edge[i].cap-=cap;

edge[opp(i)].cap+=cap;

res+=cap;

if(res==mx) {

return res;

}

}

}

return res;

}

int Dinic() {

int ans=0,cap;

while(bfs()) {

memcpy(cur,fir,sizeof(cur));//拷贝fir

while(cap=dfs(STA,INF),cap>0) {

ans+=cap;

}

}

return ans;

}

int f,p,s,e,w,sum;

int a[203],b[203];

long long l,r,mid,MAXT=40000000000000LL;

long long g[203][203];

int main() {

while(2==scanf("%d%d",&f,&p)) {

sum=0;

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

scanf("%d%d",a+i,b+i);

sum+=a[i];

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

g[i][j]=g[j][i]=MAXT+1;

}

g[i][i]=0;

}

STA=0;

DES=1;

while(p-->0) {

scanf("%d%d%d",&s,&e,&w);

g[s][e]=g[e][s]=min(g[s][e],(long long)w);

}

for(int k=1;k<=f;++k) {

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

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

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

}

}

}

l=0;

r=MAXT;

while(l<=r) {

mid=(l+r)>>1;

init();

for(int i=1;i<=f;++i) {//将牛棚i拆成两点，源点STA到i<<1有一条容量为a[i]的边，(i<<1)|1到汇点DES有一条容量为b[i]的边

if(a[i]>0) {

addEdge(STA,i<<1,a[i]);

}

if(b[i]>0) {

addEdge((i<<1)|1,DES,b[i]);

}

addEdge(i<<1,(i<<1)|1,INF);

}

for(int i=1;i<=f;++i) {//对所有时间小于mid的最短路建容量为无限的边

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

if(g[i][j]<=mid) {

addEdge(i<<1,(j<<1)|1,INF);

addEdge(j<<1,(i<<1)|1,INF);

}

}

}

if(Dinic()==sum) {//如果是满流，则牛全部能到达满足要求的牛棚

r=mid-1;

}

else {

l=mid+1;

}

}

printf("%lld\n",l==MAXT+1?-1:l);

}

return 0;

}

**最小割**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN=507;

const int INF=0x3f3f3f3f;

int n,m,a,b,c;

int mp[MAXN][MAXN];

int wage[MAXN];//prim算法wage维护的是当前点到树的最小值，本算法wage[i]维护的是直接连向i的边的权值和

bool vis[MAXN],combine[MAXN];//vis[i]表示i是否已经并入集合，combine[i]表示i是否已经合并到其他点

int STA,DES;

int Search() {

int i,j,mx,nxt,minCut;

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

memset(wage,0,sizeof(wage));

STA=DES=-1;

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

mx=-INF;

for(j=0;j<n;++j) {

if(!vis[j]&&!combine[j]&&mx<wage[j]) {//找到wage值最大的点

mx=wage[j];

nxt=j;

}

}

if(DES==nxt) {//如果找不到点，则图不连通

return minCut;

}

minCut=mx;//找到STA-DES割

vis[nxt]=true;//将nxt点并入集合

STA=DES;//更换STA和DES

DES=nxt;

for(j=0;j<n;++j) {

if(!combine[j]&&!vis[j]) {//当前点不再集合中且未被合并

wage[j]+=mp[nxt][j];

}

}

}

return minCut;

}

int Stoer\_Wagner() {

int i,j,ans=INF;

memset(combine,0,sizeof(combine));

for(i=0;i<n-1;++i) {

ans=min(ans,Search());

if(ans==0) {//如果图不连通

return 0;

}

combine[DES]=true;//将DES合并到STA

for(j=0;j<n;++j) {

if(!combine[j]) {//没有合并的点才合并

mp[STA][j]+=mp[DES][j];

mp[j][STA]+=mp[j][DES];

}

}

}

return ans;

}

int main() {

while(2==scanf("%d%d",&n,&m)) {

memset(mp,0,sizeof(mp));

while(m-->0) {

scanf("%d%d%d",&a,&b,&c);

mp[a][b]+=c;

mp[b][a]+=c;

}

printf("%d\n",Stoer\_Wagner());

}

return 0;

}

**组合数取模**

**Lucas定理模版**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long LL;

const LL MOD=1e9+7;

LL n,m;

LL quick\_pow(LL a, LL b) {

LL ans = 1;

a %= MOD;//防止指数为0时，返回未取模的结果

while(b!=0) {

if((b & 1)==1) {

ans = ans \* a % MOD;

}

b >>= 1;

a = a \* a % MOD;

}

return ans;

}

LL C(LL n, LL m) {

if(n < m) {

return 0;

}

if(n == m) {

return 1;

}

if(m > n - m) {//利用组合数性质，减少运算次数

m=n - m;

}

LL ans, a = 1, b = 1;

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

a = (a \* (n - i)) % MOD;

b = (b \* (m - i)) % MOD;

}

ans = (a \* quick\_pow(b, MOD-2)) % MOD;

return ans;

}

LL Lucas(LL n, LL m) {

LL ans=1;

while(m>0) {

ans=(ans\*C(n % MOD, m % MOD)) % MOD;

n/=MOD;

m/=MOD;

}

return ans;

}

**快速计算**

#define LL long long

const int MAXN=200005;

const int MOD=1e9+7;

LL quick\_pow(LL a, LL b) {

LL ans = 1;

a %= MOD;//防止指数为0时，返回未取模的结果

while(b!=0) {

if((b & 1)==1) {

ans = ans \* a % MOD;

}

b >>= 1;

a = a \* a % MOD;

}

return ans;

}

LL fac[MAXN];

LL C(LL n,LL m) {

if(n<m||m<0||n==0) {

return 0;

}

LL s1=fac[n],s2=fac[n-m]\*fac[m]%MOD;

return s1\*quick\_pow(s2,MOD-2)%MOD;

}

void init() {

fac[0]=1;

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

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

}

}

**01.数位DP**

**02.概率DP**

**03.区间DP**

**04.树形DP**

**05.TSP问题**

**06.A\***

**07.欧拉回路**

**08.最小树形图**

**09.最小生成树**

1. **二分图匹配**
2. **Tarjan（割点、桥、联通分量）**
3. **线段树**
4. **Tire**
5. **KMP**
6. **AC自动机**
7. **树状数组**
8. **单调栈**
9. **最小环**
10. **最大流**
11. **负权回路**

**数位DP**

通常的数位dp可以写成如下形式：

[IMG_256](http://www.cnblogs.com/jffifa/archive/2012/08/17/javascript:void(0);)

int dfs(int i, int s, bool e) {

if (i==-1) return s==target\_s;

if (!e && ~f[i][s]) return f[i][s];

int res = 0;

int u = e?num[i]:9;

for (int d = first?1:0; d <= u; ++d)

res += dfs(i-1, new\_s(s, d), e&&d==u);

return e?res:f[i][s]=res;

}

其中：

f为记忆化数组；

i为当前处理串的第i位（权重表示法，也即后面剩下i+1位待填数）；

s为之前数字的状态（如果要求后面的数满足什么状态，也可以再记一个目标状态t之类，for的时候枚举下t）；

e表示之前的数是否是上界的前缀（即后面的数能否任意填）。

for循环枚举数字时，要注意是否能枚举0，以及0对于状态的影响，有的题目前导0和中间的0是等价的，但有的不是，对于后者可以在dfs时再加一个状态变量z，表示前面是否全部是前导0，也可以看是否是首位，然后外面统计时候枚举一下位数。It depends.

于是关键就在怎么设计状态。当然做多了之后状态一眼就可以瞄出来。

**题目大意：求区间[1,n]中含13且是13的倍数的数字个数？**

#include <cstdio>

#include <cstring>

using namespace std;

int bit[35],len;

int dp[35][13][3];//dp[i][j][0]表示长度为i，模13后为j且不含13的数字个数；dp[i][j][1]表示长度为1，高位的数模13后为j且不含13，第i位为3的数字个数；dp[i][j][2]表示长度为i，模13后为j且含有13的数字个数

int dfs(int pos,int pre,int rem,bool limit,bool flag) {//pos表示当前考虑的位置，pre前一位的数，rem高位的数的模13的值，limit当前位取的数是否有限制，flag前面是否出现过13

if(pos<=0) {

return flag&&rem==0?1:0;//如果前面出现过13，且余数为0，则符合题意，为1；否则不符合题意，为0

}

if(!limit) {//如果没有限制，就可以直接统计后面的所有位的情况

if(flag) {//如果前面已经出现过13，则后面所有的位任取不含13的

if(dp[pos][rem][0]!=-1) {//如果计算过就直接返回

return dp[pos][rem][0];

}

}

else {//如果前面没出现过13

if(pre==1&&dp[pos][rem][1]!=-1) {//如果上一位是1，则后面的最高位取3，若计算过就直接返回

return dp[pos][rem][1];

}

if(pre!=1&&dp[pos][rem][2]!=-1) {//如果上一位不是1，则后面取含13的数，若计算过就直接返回

return dp[pos][rem][2];

}

}

}

int res=0,mx=limit?bit[pos]:9;//mx表示当前位最大能取的数

for(int i=0;i<=mx;++i) {//枚举当前位的数字

res+=dfs(pos-1,i,(rem\*10+i)%13,limit&&i==mx,flag||(pre==1&&i==3));

}

if(!limit) {//此时保存刚才算得的结果

if(flag) {

dp[pos][rem][0]=res;

}

else {

if(pre==1) {

dp[pos][rem][1]=res;

}

else {

dp[pos][rem][2]=res;

}

}

}

return res;

}

int getCnt(int n) {

len=0;

while(n>0) {

bit[++len]=n%10;

n/=10;

}

return dfs(len,0,0,true,false);

}

int main() {

int n=0;

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

while(1==scanf("%d",&n)) {

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

}

return 0;

}

**概率DP**

题目大意：共有n张卡，每次一包方便面，可能有p[i]概率获得第i张卡，也有可能没有卡获得，求拿到全部n张卡需要买的方便面的期望？

# 解法一：概率DP

感觉做了这么多概率dp，还是离熟悉比较远

合集里看到的，结果一眼就看到是用状态压缩做，然后状态都出来了，转移就没什么难度了...

设dp[i]表示当前取到了i的二进制中位的为1的卡时，离达到目标状态还需要购买方便面的期望，初始状态：dp[(1<<n)-1]=0;

则dp[i]可以转化为：

①：下一袋方便面没有卡，或j卡已有，即：(∑p[j]+pp)\*(dp[i]+1);

②：下一袋方面面存在j卡，且当前没有，即：(∑p[j]\*(dp[i|(1<<j)]+1);

则状态转移方程为：dp[i]=(∑p[j]+pp)\*(dp[i]+1)+∑p[j]\*(dp[i|(1<<j)]+1);

化简后得：dp[i]=(∑p[j]\*dp[i|(1<<j)]+1)/(1-∑p[j]-pp);

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN=(1<<20)+5;

int n,mx;

double p[25],pp,tmp;//pp表示没有卡的概率

double dp[MAXN];//dp[i]表示当前取到了i的二进制中位的为1的卡时，离达到目标状态还需要购买方便面的期望

int main() {

while(1==scanf("%d",&n)) {

mx=1<<n;

pp=1;

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

scanf("%lf",p+i);

pp-=p[i];

}

dp[mx-1]=0;

for(int i=mx-2;i>=0;--i) {

dp[i]=1;

tmp=pp;

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

if((i&(1<<j))==0) {//由仅比i状态多1张卡的状态转移

dp[i]+=p[j]\*dp[i|(1<<j)];

}

else {//tmp表示再买一包时，里面没卡以及卡是已在i状态有时的概率和

tmp+=p[j];

}

}

dp[i]/=1-tmp;

}

printf("%.4lf\n",dp[0]);

}

return 0;

}

**区间DP**

题目大意：给定两堆均含有n张牌的牌堆，牌上的数字分别是a[i],b[i]，每次都已从任意一堆的顶部或底部拿一张，每个人都用最优策略，求先手能拿到的最大数字和？

设dp[i][j][k][l]表示先手从a数组区间[i,j]中以及b数组区间[k,l]中能选得的最大数字和

则dp[i][j][k][l]可由四个状态转移而来：

①先手拿a[i]，由dp[i+1][j][k][l]转移而来，即 a[i]+sum(i+1,j,k,l)-dp[i+1][j][k][l];

②先手拿a[j]，由dp[i][j-1][k][l]转移而来，即 a[j]+sum(i,j-1,k,l)-dp[i][j-1][k][l];

③先手拿b[k]，由dp[i][j][k+1][l]转移而来，即 b[k]+sum(i,j,k+1,l)-dp[i][j][k+1][l];

④先手拿b[l]，由dp[i][j][k][l-1]转移而来，即 b[l]+sum(i,j,k,l-1)-dp[i][j][k][l-1];

则dp[i][j][k][l]为上述最大值

感觉和只有一堆牌的情况差不多，但是连状态如何转移都知道了，却不知道如何实现，因为存在一堆没有选，而另一堆已选了的状态（这些状态没法表示，不好控制边界），看了题解才知道要用dfs控制边界从而进行状态的转移

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN=10005;

int par[MAXN],ps,pe,n,m,a[25],b[25];

int dp[25][25][25][25];//dp[i][j][k][l]表示先手从a数组区间[i,j]中以及b数组区间[k,l]中能选得的最大数字和

int dfs(int i,int j,int k,int l) {

if(i>j&&k>l) {

return 0;

}

if(dp[i][j][k][l]!=0) {

return dp[i][j][k][l];

}

int mx=0,sum=0;

if(i<=j) {

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

}

if(k<=l) {

sum+=b[l]-b[k-1];

}

if(i<=j) {

mx=max(mx,sum-dfs(i+1,j,k,l));

mx=max(mx,sum-dfs(i,j-1,k,l));

}

if(k<=l) {

mx=max(mx,sum-dfs(i,j,k+1,l));

mx=max(mx,sum-dfs(i,j,k,l-1));

}

return dp[i][j][k][l]=mx;

}

int main(){

int T;

scanf("%d",&T);

while(T-->0) {

a[0]=b[0]=0;

memset(dp,0,sizeof(dp));

scanf("%d",&n);

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

scanf("%d",a+i);

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

}

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

scanf("%d",b+i);

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

}

printf("%d\n",dfs(1,n,1,n));

}

return 0;

}

**树形DP**

#### 描述

上回说到，小Ho有着一棵灰常好玩的树玩具！这棵树玩具是由N个小球和N-1根木棍拼凑而成，这N个小球都被小Ho标上了不同的数字，并且这些数字都是处于1..N的范围之内，每根木棍都连接着两个不同的小球，并且保证任意两个小球间都不存在两条不同的路径可以互相到达。没错，这次说的还是这棵树玩具的故事！

小Ho的树玩具的质量似乎不是很好，短短玩了几个星期，便掉漆了！

“简直是一场噩梦！”小Ho拿着树玩具眼含热泪道。

“这有什么好忧伤的，自己买点油漆刷一刷不就行了？”小Hi表示不能理解。

“还可以这样？”小Ho顿时兴高采烈了起来，立马跑出去买回来了油漆，但是小Ho身上的钱却不够——于是他只买回了有限的油漆，这些油漆最多能给M个结点涂上颜色，这就意味着小Ho不能够将他心爱的树玩具中的每一个结点都涂上油漆！

小Ho低头思索了半天——他既不想只选一部分结点补漆，也不想找小Hi借钱，但是很快，他想出了一个非常棒的主意：将包含1号结点的一部分连通的结点进行涂漆（这里的连通指的是这一些涂漆的结点可以互相到达并且不会经过没有涂漆的结点），然后将剩下的结点拆掉！

那么究竟选择哪些结点进行涂漆呢？小Ho想了想给每个结点都评上了分——他希望最后留下来，也就是涂漆了的那些结点的评分之和可以尽可能的高！

那么，小Ho该如何做呢？

#### 输入

每个测试点（输入文件）有且仅有一组测试数据。

每组测试数据的第一行为两个整数N、M，意义如前文所述。

每组测试数据的第二行为N个整数，其中第i个整数Vi表示标号为i的结点的评分

每组测试数据的第3~N+1行，每行分别描述一根木棍，其中第i+1行为两个整数Ai，Bi，表示第i根木棍连接的两个小球的编号。

对于100%的数据，满足N<=10^2，1<=Ai<=N, 1<=Bi<=N, 1<=Vi<=10^3, 1<=M<=N

小Hi的Tip：那些用数组存储树边的记得要开两倍大小哦！

#### 输出

对于每组测试数据，输出一个整数Ans，表示使得涂漆结点的评分之和最高可能是多少。

**样例输入**

10 4

370 328 750 930 604 732 159 167 945 210

1 2

2 3

1 4

1 5

4 6

4 7

4 8

6 9

5 10

**样例输出**

2977

#### 官方提示：树上的动态规划？其实老早就接触过了吧！

“是啊，我该怎么做呢？”小Ho想道，但是如果能很快就自己想出来那也就不是小Ho了，于是小Ho还是老老实实去请教了小Hi。

小Hi听了小Ho的问题，道：“这个问题不是很简单么？来，我们再重复一下之前的步骤——先抽象你的问题。”

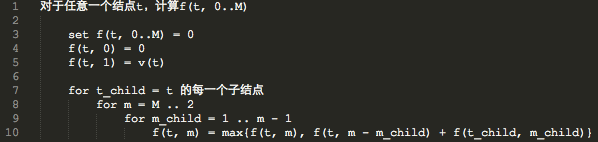
“好的！应该是这样的——**f(t, m)表示，在以t为根的一棵树中，选出包含根节点t的m个连通的结点，能够获得的最高的评分，然后我们的答案就是f(1, M)！**”身经百战的小Ho也是只需要一下点拨，立马就答了出来。

“那么你应该如何分解这个问题为子问题呢？”小Hi继续问道。

“一般的思路会是这样子的，首先我要包含根节点，然后与根节点连通的结点最开始便是根节点的子结点，而所有选择的结点都要互相连通的话，那么如果选择某一棵子树中的结点的话就势必也需要选择这棵子树的根节点——所以就变成了一个规模小一些的子问题。比如在求解f(t, m)的时候，我先枚举t的第一个子结点t1中选出的结点数m1,然后枚举t的第二个子结点t2中选出的结点数m2……一直到t的最后一个子结点tk中选出的结点数mk，这样就有f(t, m) = max{f(t1, m1) + f(t2, m2) + …… + f(tk, mk)} + v(t)，并且需要保证m1+m2+...+mk+1=m。”小Ho答道。

小Hi摇了摇头：“但是你不觉得这样这个[**算法**](http://lib.csdn.net/base/31)就是指数级了么？m1...mk可能有的方案数可是非常多的呢！”

“唔……我知道了，这里不是和无限背包问题很像么？我可以不用单独的求解每一个f(t, m)而是针对于每一个t，同时求解它的f(t, 0..M)，这样的话，我就可以把m视作背包容量，把每个子结点t\_child都视作一件单位重量为1的物品，但是和背包问题不同的是，这件物品的总价值并不是单位价值乘以总重量，而是重量为m\_child的该物品的价值为f(t\_child, m\_child)，这样我就可以像无限背包问题一样，用这样的方法来进行求解！



“没错呢！但是你这样的话不会导致f(t\_child, m\_child)计算很多次么？”小Hi也是故意要考一考小Ho。

“这你就小瞧我了，我学了这么久动态规划难道还不知道我可以以后序遍历的方式访问这棵树，这样当计算f(t, 0..M)的时候，我就已经计算出了所有的f(t\_child, m\_child)的值，如果我将这些值储存在数组中的话，我就不需要再递归计算了！”小Ho信心满满的答道。

“真聪明！那你还不快去写程序？你的油漆都要干了哦！”

第一次做树形DP，以前听过也找到过各种讲解，可是本身DP不太好，所以不敢接触...

实际做一次发现其实很好理解，只不过是将DP过程放在后序遍历中

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

struct Node {

int e,nxt;

}node[205];//数组模拟链表

int n,m,num;

int h[105];

int dp[105][105];//dp[i][j]表示以i为根结点，选出包含根节点的j个节点，这样的选法最高的评分

void add(int s,int e) {//加边

node[++num].e=e;

node[num].nxt=h[s];

h[s]=num;

}

void dfs(int u,int pre) {

int p=h[u],i,j;

while(p) {

if(node[p].e!=pre) {

dfs(node[p].e,u);//先对树u的子树进行dp

for(i=m;i>=2;--i)//这里i递减是为了用到上一个子树产生的dp值，如果反过来就会用到本次子树产生的dp值，导致一个结点被多次涂油漆，差不多与01背包和完全背包的区别一样

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

dp[u][i]=max(dp[u][i],dp[u][i-j]+dp[node[p].e][j]);//对树u进行dp

}

p=node[p].nxt;

}

}

int main() {

int s,e,i;

while(2==scanf("%d%d",&n,&m)) {

memset(h,0,sizeof(h));

memset(dp,0,sizeof(dp));

num=0;

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

scanf("%d",&dp[i][1]);

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

scanf("%d%d",&s,&e);

add(s,e);

add(e,s);

}

dfs(1,0);

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

}

return 0;

}

TSP问题

题目大意：给定一个n\*n的格子，'.'代表可以走，'#'代表不可以走，每分钟可以往相邻的格子走，又存起点和终点已知的m条隧道，可以从任意一点开始，求经过所有隧道所用的最短时间？

大致思路：先bfs预处理出任意两点的最短路，然后就转换为TSP问题，由于必经的隧道很小，所以可以用状压DP

设dp[i][j]表示当前i中二进制中为1的位已经走过，且最后一个走过的隧道为第j条隧道

【注意】必须把表示走过的某些隧道这个状态放在最外层，把隧道放在最内层，WA在这2.5h，还是不太熟悉状压DP

只有这样才能保证当前使用的状态是最优状态，否则就限定了部分遍历隧道的顺序

①若未遍历的隧道在最外层，则限定了最后一个必定经过第m-1条隧道

②若已遍历的隧道在最外层，则限定了倒数第二个必定经过第m-1条隧道

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

using namespace std;

struct Node {

int r,c,t;

Node(int rr=0,int cc=0,int tt=0):r(rr),c(cc),t(tt) {}

}u;

struct Tunnel {

int sr,sc,er,ec;

}tun[17];

char city[17][17];

int n,m,rr,cc,mx;

int time[17][17][17][17];

bool vis[17][17];

int dp[(1<<15)+5][17];

const int dr[4]={-1,0,1,0};

const int dc[4]={0,1,0,-1};

inline bool isInside(int r,int c) {

return 1<=r&&r<=n&&1<=c&&c<=n;

}

void bfs(int r,int c) {

queue<Node> q;

q.push(Node(r,c,0));

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

vis[r][c]=true;

while(!q.empty()) {

u=q.front();

q.pop();

time[r][c][u.r][u.c]=u.t;

for(int i=0;i<4;++i) {

rr=u.r+dr[i];

cc=u.c+dc[i];

if(isInside(rr,cc)&&city[rr][cc]!='#'&&!vis[rr][cc]) {

vis[rr][cc]=true;

q.push(Node(rr,cc,u.t+1));

}

}

}

}

int main() {

while(2==scanf("%d%d",&n,&m)) {

mx=1<<m;

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

scanf("%s",&city[i][1]);

}

memset(time,0x3f,sizeof(time));

memset(dp,0x3f,sizeof(dp));

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

scanf("%d%d%d%d",&tun[i].sr,&tun[i].sc,&tun[i].er,&tun[i].ec);

bfs(tun[i].er,tun[i].ec);

dp[1<<i][i]=0;

}

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

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

if((j&(1<<i))==0) {//如果当前状态的第i条隧道未走过

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

if(dp[j][k]!=0x3f3f3f3f) {//如果当前状态的第k条隧道已走过

dp[j|(1<<i)][i]=min(dp[j|(1<<i)][i],dp[j][k]+time[tun[k].er][tun[k].ec][tun[i].sr][tun[i].sc]);

}

}

}

}

}

int ans=0x3f3f3f3f;

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

ans=min(ans,dp[mx-1][i]);

}

printf("%d\n",ans==0x3f3f3f3f?-1:ans);

}

return 0;

}

**A\***

**Input**

You will receive, several descriptions of configuration of the 8 puzzle. One description is just a list of the tiles in their initial positions, with the rows listed from top to bottom, and the tiles listed from left to right within a row, where the tiles are represented by numbers 1 to 8, plus 'x'. For example, this puzzle   
  
1 2 3   
x 4 6   
7 5 8   
  
is described by this list:   
  
1 2 3 x 4 6 7 5 8

**Output**

You will print to standard output either the word ``unsolvable'', if the puzzle has no solution, or a string consisting entirely of the letters 'r', 'l', 'u' and 'd' that describes a series of moves that produce a solution. The string should include no spaces and start at the beginning of the line. Do not print a blank line between cases.

**Sample Input**

2 3 4 1 5 x 7 6 8

**Sample Output**

ullddrurdllurdruldr

今年在北大暑期学校听讲时，郭老师讲过，但他的代码不能完全理解，特别是排列和序号相互转换。

此次再遇八数码问题，看到别的大神用康托展开计算序号，感觉十分方便，应该牢记。

但看到大神代码，优先队列以h为第一关键字，g为第二关键字能AC且450ms；

我自己本地跑，连样例都无法过，不过以f为第一关键字，h为第二关键字的方法比只以f为关键字的方法快大概200ms

#include <cstdio>

#include <cstring>

#include <string>

#include <queue>

using namespace std;

struct Node {

int a[9],pos,has,h,g;

bool operator < (const Node& a) const {

if(h+g!=a.h+a.g)

return h+g>a.h+a.g;

return h!=a.h?h>a.h:g>a.g;

}

}sta,u,v;

inline bool up(const Node& t) {

return t.pos>2;

}

inline bool right(const Node& t) {

return t.pos!=2&&t.pos!=5&&t.pos!=8;

}

inline bool down(const Node& t) {

return t.pos<6;

}

inline bool left(const Node& t) {

return t.pos!=0&&t.pos!=3&&t.pos!=6;

}

bool (\*tab[4])(const Node& t)={up,right,down,left};

int HASH[9]={40320,5040,720,120,24,6,2,1,1};//n的阶乘，变进制

int turn[400001];

int pre[400001];

const int des=46233;

const int d[]={-3,1,3,-1};

const char m[]={"urdl"};

const int hs[9][9]={0,0,1,2,1,2,3,2,3,//位置hs[i][j]表示位置i数字为j时与正确位置的曼哈顿距离

0,1,0,1,2,1,2,3,2,

0,2,1,0,3,2,1,4,3,

0,1,2,3,0,1,2,1,2,

0,2,1,2,1,0,1,2,1,

0,3,2,1,2,1,0,3,2,

0,2,3,4,1,2,3,0,1,

0,3,2,3,2,1,2,1,0,

0,4,3,2,3,2,1,2,1};

bool judge() {//奇偶剪枝

int num=0,i,j;

for(i=0;i<9;++i)

for(j=i+1;j<9;++j)

if(sta.a[i]&&sta.a[j]&&sta.a[i]>sta.a[j])

++num;

return (num&1)==1;

}

int get\_hash(const Node& t) {

int num=0,i,j,k;

for(i=0;i<9;++i) {

k=0;

for(j=i+1;j<9;++j)

if(t.a[i]>t.a[j])

++k;

num+=HASH[i]\*k;

}

return num;

}

int get\_h(const Node& t) {//计算估价函数h，曼哈顿距离和

int num=0;

for(int i=0;i<9;++i)

num+=hs[i][t.a[i]];

return num;

}

void astar() {

int i;

priority\_queue<Node> q;

q.push(sta);

while(!q.empty()) {

u=q.top();

q.pop();

for(i=0;i<4;++i) {

if(tab[i](u)) {

v=u;

v.pos+=d[i];

swap(v.a[u.pos],v.a[v.pos]);

if(turn[v.has=get\_hash(v)]==-1) {

turn[v.has]=i;

++v.g;

v.h=get\_h(v);

pre[v.has]=u.has;

q.push(v);

}

if(v.has==des)

return ;

}

}

}

}

void print() {

string ans;

int nxt=des;

while(pre[nxt]!=-2) {

ans.push\_back(m[turn[nxt]]);

nxt=pre[nxt];

}

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

printf("%c",ans[i]);

printf("\n");

}

int main() {

//freopen("out.txt","w",stdout);

int i;

char s[3];

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

if(s[0]=='x') {

sta.a[0]=0;

sta.pos=0;

}

else

sta.a[0]=s[0]-'0';

for(i=1;i<9;++i) {

scanf("%s",s);

if(s[0]=='x') {

sta.a[i]=0;

sta.pos=i;

}

else

sta.a[i]=s[0]-'0';

}

if(judge()) {

printf("unsolvable\n");

continue;

}

if((sta.has=get\_hash(sta))==des) {

printf("\n");

continue;

}

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

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

pre[sta.has]=-2;

turn[sta.has]=0;

sta.g=0,sta.h=get\_h(sta);

astar();

print();

}

return 0;

}

**欧拉回路**

## 描述

Farmer John每年有很多栅栏要修理。他总是骑着马穿过每一个栅栏并修复它破损的地方。

John是一个与其他农民一样懒的人。他讨厌骑马，因此从来不两次经过一个栅栏。你必须编一个程序，读入栅栏网络的描述，并计算出一条修栅栏的路径，使每个栅栏都恰好被经过一次。John能从任何一个顶点(即两个栅栏的交点)开始骑马，在任意一个顶点结束。

每一个栅栏连接两个顶点，顶点用1到500标号(虽然有的农场并没有500个顶点)。一个顶点上可连接任意多(>=1)个栅栏。两顶点间可能有多个栅栏。所有栅栏都是连通的(也就是你可以从任意一个栅栏到达另外的所有栅栏)。

你的程序必须输出骑马的路径(用路上依次经过的顶点号码表示)。我们如果把输出的路径看成是一个500进制的数，那么当存在多组解的情况下，输出500进制表示法中最小的一个 (也就是输出第一位较小的，如果还有多组解，输出第二位较小的，等等)。

输入数据保证至少有一个解。

## 格式

**PROGRAM NAME**: fence

**INPUT FORMAT**：

(file fence.in)

第1行: 一个整数F(1 <= F <= 1024)，表示栅栏的数目

第2到F+1行: 每行两个整数i, j(1 <= i,j <= 500)表示这条栅栏连接i与j号顶点。

**OUTPUT FORMAT**：

(file fence.out)

输出应当有F+1行，每行一个整数，依次表示路径经过的顶点号。注意数据可能有多组解，但是只有上面题目要求的那一组解是认为正确的。

## SAMPLE INPUT

9

1 2

2 3

3 4

4 2

4 5

2 5

5 6

5 7

4 6

## SAMPLE OUTPUT

1

2

3

4

2

5

4

6

5

7

这个算法的大致思路弄懂了：递归到一个点时，对该点所有的临接点递归调用（WA了后才发现根本没弄懂，直接跳过了图片模拟过程）

但在写代码时想直接输出结果，就直接在dfs的第一行输出当前点，发现WA在第7组数据（其实早有心理准备，因为如果下一个点是终点，但是还存在其他边，这样就会出错）

看了别人的解释：遍历完当前点的所有临接点后在记录当前点，最后逆序输出，这样能纠正类似上面提到的错误，但还不清楚原理...

/\*

ID: your\_id\_here

PROG: fence

LANG: C++

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

int f,s,e,mn=505,mx=0;

int g[505][505];

int ans[1405],len,sta;

void dfs(int cur) {

for(int i=mn;i<=mx;++i) {

if(g[cur][i]>0) {

--g[cur][i];

--g[i][cur];

dfs(i);

}

}

ans[len++]=cur;

}

int main() {

freopen("fence.in","r",stdin);

freopen("fence.out","w",stdout);

memset(g,0,sizeof(g));

scanf("%d",&f);

while(f--) {

scanf("%d%d",&s,&e);

mx=max(mx,s>e?s:e);

mn=min(mn,s<e?s:e);

++g[s][e];

++g[e][s];

++g[s][0];

++g[e][0];

}

sta=mn;//令起点为值最小的点

int i=mn;

for(;i<=mx;++i)

if((g[i][0]&1)==1) {//若存在度为奇数的点，则起点为该点

sta=i;

break;

}

len=0;

dfs(sta);

while(len>0)

printf("%d\n",ans[--len]);

return 0;

}

**最小树形图**

题目大意：有n个课程，每个课程又等级a[i]，初始都在等级0，满足c[j]课程等级大于等于L1[j]时，可花费money[j]使得d[j]课程等级达到L2[j]，求最少需要多少钱？

把每个课程点每一个等级看成一个点，则题目给点条件就可以看成有向边，再添上每一个课程高等级到低等级点权值为0的边，就可以使c[j]课程等级大于L1[j]时，仍可花费money[j]达到d[j]课程等级L2[j]，再添加一个虚拟点根结点，使其能无花费到达每一个课程的0等级结点。

这样就可以直接用 最小树形图 点模板了。

#include <cstdio>

#include <cstring>

using namespace std;

const int MAXN=555;

const int MAXM=2555;

const int INF=0x3f3f3f3f;

int n,m;

int pre[MAXN],id[MAXN],vis[MAXN];//pre[i]表示以i点为终点的最小权值入边的起点，id[i]表示i点在新图上对应的点

int inEdge[MAXN];//inEdge[i]表示以i点为终点的入边的最小权值

struct Edge {

int s,e;

int w;

}edge[MAXM];

int Directed\_MST(int root) {

int s,e;

int ans=0;

while(true) {

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

inEdge[i]=INF;

for(int i=0;i<m;++i) {//找权值最小的入边

s=edge[i].s;

e=edge[i].e;

if(edge[i].w<inEdge[e]&&s!=e) {//如果当前边非自环且权值更小

pre[e]=s;

inEdge[e]=edge[i].w;

}

}

for(int i=0;i<n;++i)//如果存在一点没有入边则无法形成最小树形图

if(i!=root&&inEdge[i]==INF)

return -1;

int cnt=0;//cnt表示新图的点数

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

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

inEdge[root]=0;

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

ans+=inEdge[i];

int cur=i;

while(vis[cur]!=i&&cur!=root&&id[cur]==-1) {

vis[cur]=i;

cur=pre[cur];

}

if(cur!=root&&id[cur]==-1) {//找到一个环

e=pre[cur];

while(e!=cur) {//在一个环上的点都映射到新图上的同一个点

id[e]=cnt;

e=pre[e];

}

id[cur]=cnt++;

}

}

if(cnt==0)//无环则返回答案

return ans;

for(int i=0;i<n;++i)//每个点映射到新图的一个点

if(id[i]==-1)

id[i]=cnt++;

for(int i=0;i<m;++i) {//建立新图

s=edge[i].s;

e=edge[i].e;

edge[i].s=id[s];

edge[i].e=id[e];

if(id[s]!=id[e])

edge[i].w-=inEdge[e];

}

n=cnt;//更新新图的点数

root=id[root];//更新新图的根

}

return ans;

}

int main() {

int a,c,L1,d,L2,money,num;

int index[55][505];//index为每个课程点每个等级建立一个点

while(scanf("%d%d",&n,&m),n!=0||m!=0) {

num=1;

for(int i=1;i<=n;++i) {//每个课程每个等级映射到一个点

scanf("%d",&a);

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

index[i][j]=num++;

index[i][502]=a;

}

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

scanf("%d%d%d%d%d",&c,&L1,&d,&L2,&money);

edge[i].s=index[c][L1];

edge[i].e=index[d][L2];

edge[i].w=money;

}

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

for(int j=1;j<=index[i][502];++j) {//每一课程相邻的两个等级，高等级到低等级有一条权值为0点有向边

edge[m].s=index[i][j];

edge[m].e=index[i][j-1];

edge[m++].w=0;

}

}

for(int i=1;i<=n;++i) {//建立一个虚拟树根，到每个课程的0等级都有一条权值为0的有向边

edge[m].s=0;

edge[m].e=index[i][0];

edge[m++].w=0;

}

n=num;

printf("%d\n",Directed\_MST(0));

}

return 0;

}

**最小生成树**

题目大意：经过调查评估，得到的统计表中列出了有可能建设公路的若干条道路的成本。现请你编写程序，计算出全省畅通需要的最低成本。

#include <cstdio>

#include <cstring>

#include <queue>

#include <vector>

#include <algorithm>

using namespace std;

const int INF=0x3f3f3f3f;

struct Edge{

int v,w;//v表示边端点，另一个端点已知；w表示边权值，也表示v到最小生成树的距离

Edge(int vv=0,int ww=INF):v(vv),w(ww) {}

bool operator < (const Edge& a) const {

return w>a.w;

}

}u;

int m,n,dis[105],ans,cnt;//dis表示各顶点到最小生成树的距离

bool vis[105];//vis表示各顶点是否已被加入最小生成树

vector<vector<Edge> > g(105);//邻接表

bool prim() {//prim算法求最小生成树

int i,j,v,w;

priority\_queue<Edge> q;

q.push(Edge(1,0));

ans=cnt=0;

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

memset(dis,0x3f,sizeof(dis));

while(cnt<n&&!q.empty()) {

do {

u=q.top();

q.pop();

}while(vis[u.v]&&!q.empty());

if(!vis[u.v]) {

++cnt;

ans+=u.w;

vis[u.v]=true;

for(i=0,j=g[u.v].size();i<j;++i)

if(!vis[v=g[u.v][i].v]&&dis[v]>(w=g[u.v][i].w)) {

dis[v]=w;

q.push(Edge(v,w));

}

}

}

return cnt==n;

}

int main() {

int i,w,s,e;

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

g.clear();

g.resize(n+1);

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

scanf("%d%d%d",&s,&e,&w);

g[s].push\_back(Edge(e,w));

g[e].push\_back(Edge(s,w));

}

if(prim())

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

else

printf("?\n");

}

return 0;

}

**二分图匹配**

匈牙利[**算法**](http://lib.csdn.net/base/31)专门用于部图的匹配，所以速度极快，全部test都是0.00s

大致思路：枚举n头奶牛为起点，若找到一条增广路径（终点为未匹配，且奇数边均不在原匹配中，偶数边均在原匹配中），则答案+1

DFS实现找增广路径：dfs参数是i表示奶牛，枚举奶牛喜欢的牛棚，若牛棚不在当前增广路径中，则将其“纳入”增广路径（不一定出现在增广路径中，失败时不必再除去，因为当前状况下就失败，后面的状况不会更好，算作剪枝），若枚举的牛棚未被占据 或 以占据该牛棚的牛为起点时存在一条增广路径，则将牛棚与当前牛匹配，并返回true，表示成功找到一条增广路径

/\*

ID: your\_id\_here

PROG: stall4

LANG: C++

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

int n,m;

int adj[205][205],match[205];//adj[i][k]表示奶牛i喜欢牛棚adj[i][k]，adj[i][0]表示奶牛i喜欢的牛棚的个数

bool onPath[205];//onPath[j]表示牛棚是否在增广路径上；match[j]表示牛棚j被奶牛match[j]占据

bool dfs(int i) {

int j,k;

for(k=adj[i][0];k>=1;--k) {

j=adj[i][k];

if(!onPath[j]) {//若牛棚j不在本次增广路径上

onPath[j]=true;

if(0==match[j]||dfs(match[j])) {//若牛棚j未被占据 或者 有一条以 奶牛match[j]为起点的增广路径

match[j]=i;//牛棚j被奶牛i占据

return true;

}//即使匹配失败，也不需要将onPath[j]重置为false，因为当前状况下就失败，后面的状况不会更好，算是一种剪枝吧

}

}

return false;

}

int hungary() {

int ans=0;

memset(match,0,sizeof(match));

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

memset(onPath,false,sizeof(onPath));

if(dfs(i))//若有一条以奶牛i为起点的增广路径

++ans;

}

return ans;

}

int main() {

freopen("stall4.in","r",stdin);

freopen("stall4.out","w",stdout);

int num;

memset(adj,0,sizeof(adj));

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

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

scanf("%d",&num);

adj[i][0]=num;

while(num>0) {

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

}

}

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

return 0;

}

**Tarjan（连通分量、割点、桥）**

**题目大意：给定一个无向图，若删除某个点，剩余的图不连通，则输出该点和剩余图的连通分量数。（割点）**

找不到练习寻找割点的裸的简单题了，本题是求删除割点后能形成的连通分量数，感觉求连通分量数的写法与讲义还是有些差别，但是明白之后发现这样写更简洁明了

割点：①若u为树根，且u有多余一个子树

            ②若u不为树根，且存在(u,v)为树枝边（或称父子边，即u为v在搜索树中的父亲），使得dfn[u]<=low[v]

桥：当(u,v)为树枝边，且满足dfn[u]<low[v]（前提是其没有重边）

#include <cstdio>

#include <cstring>

#include <vector>

using namespace std;

const int MAXN=1005;

int n,num;

int mp[MAXN],low[MAXN],dfn[MAXN],blocks[MAXN];//biocks[i]表示删掉i点后，能形成的连通块数

vector<int> g[MAXN];

bool haveSPF;//haveSPF表示有无SPF

void Tarjan(int u,int p) {

dfn[u]=low[u]=++num;

int v;

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

v=g[u][i];

if(u!=p) {//若v不为u的父节点

if(dfn[v]==0) {//若v未遍历

Tarjan(v,u);

low[u]=min(low[u],low[v]);

if(dfn[u]<=low[v])//若最终blocks[u]>1时，u才割点（若u为根节点，则其至少2次为true；若u不为根节点，则其至少1次为true）

++blocks[u];//删除点u后，其子结点能形成的连通块数+1

//若该不等式不取等号，则(u,v)为桥

}

else if(dfn[v]<low[u])//若v遍历到时间更小

low[u]=dfn[v];

}

}

}

int main() {

int s,e,kase=0;

while(scanf("%d",&s),s!=0) {

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

g[i].clear();

memset(mp,0,sizeof(mp));

num=n=0;

do {

scanf("%d",&e);

if(mp[s]==0)

mp[s]=++n;

if(mp[e]==0)

mp[e]=++n;

g[mp[s]].push\_back(mp[e]);

g[mp[e]].push\_back(mp[s]);

} while(scanf("%d",&s),s!=0);

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

dfn[i]=0;

blocks[i]=1;//默认删除i点后，i点的父亲结点有一个连通块

}

for(int i=1;i<=n;++i) {//防止图不连通

if(dfn[i]==0) {

blocks[i]=0;//根节点无父亲结点，即不存在父亲结点的连通块

Tarjan(i,0);

}

}

haveSPF=false;

printf("Network #%d\n",++kase);

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

if(blocks[mp[i]]>1) {

haveSPF=true;

printf(" SPF node %d leaves %d subnets\n",i,blocks[mp[i]]);

}

}

if(!haveSPF)

printf(" No SPF nodes\n");

printf("\n");

}

return 0;

}

**题目大意：求无向图中桥的个数？**

#include <cstdio>

#include <cstring>

#include <vector>

using namespace std;

int n,m,cnt;

int dfn[10005],low[10005],num;

vector<int> g[10005];

void Tarjan(int u,int p) {

dfn[u]=low[u]=++num;

int v;

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

v=g[u][i];

if(v!=p) {

if(dfn[v]==0) {

Tarjan(v,u);

low[u]=min(low[u],low[v]);

if(dfn[u]<low[v]) {

++cnt;

}

}

else {

low[u]=min(low[u],dfn[v]);

}

}

}

}

int main() {

int T,s,e;

scanf("%d",&T);

while(T-->0) {

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

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

g[i].clear();

dfn[i]=0;

}

while(m-->0) {

scanf("%d%d",&s,&e);

g[s].push\_back(e);

g[e].push\_back(s);

}

num=cnt=0;

Tarjan(0,-1);

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

}

return 0;

}

**边双联通分量**

题目大意：给定一个无向图，求加入最少的边，使该图变成一个双连通分量？

点双连通分量：不存在割点的连通分量

边双连通分量：不存在桥的连通分量（即任意两点互相可达的路径有不同的两条）

大致方法：可以求出所有的桥，把桥删掉。然后把所有的边双连通分量求出来，显然这些边双连通分量就是原图中的双连通分量。把它们缩成点，然后添上刚才删去的桥，就构成了一棵树。在树上添边使得树变成一个双连通分量即可。可以统计该树度为1的叶子节点（设共有x个），则答案为：(x+1)/2

注意：题目数据会给出重边，但是重边只算一次（即题目意思是两点之间有路，这个信息可能会重复出现，但实际上只有一条路），但是我看有一个[题解](http://www.cnblogs.com/frog112111/p/3367039.html)没有判断重边也过了，很是费解（这个题解用数组模拟的邻接表，判断父子边时，由于父子边的下标只差1，所以通过异或运算判断，但是不能判断重边，然后就不明白了）

#include <cstdio>

#include <cstring>

#include <vector>

using namespace std;

const int MAXN=5005;

int n,m,num,ans,cnt,top;

int stak[MAXN],low[MAXN],dfn[MAXN],color[MAXN],deg[MAXN];//biocks[i]表示删掉i点后，能形成的连通块数

vector<int> g[MAXN];

bool isIn[MAXN],vis[MAXN][MAXN];

void Tarjan(int u,int p) {//求无向图的边双连通分量的Tarjan与求有向图的强连通分量的Tarjan好像...

stak[++top]=u;

isIn[u]=true;

dfn[u]=low[u]=++num;

int v;

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

v=g[u][i];

if(v!=p) {

if(dfn[v]==0) {

Tarjan(v,u);

low[u]=min(low[u],low[v]);

}

else if(isIn[v]&&dfn[v]<low[u])

low[u]=dfn[v];

}

}

if(dfn[u]==low[u]) {

++cnt;

do {

v=stak[top--];

isIn[v]=false;

color[v]=cnt;//一个边双连通分量缩成一个点

} while(v!=u);

}

}

int solve() {

cnt=num=top=0;//cnt表示边双连通分量的个数

memset(dfn,0,sizeof(dfn));

memset(isIn,false,sizeof(isIn));

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

if(dfn[i]==0)

Tarjan(i,0);

}

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

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

for(int j=0;j<g[i].size();++j)

if(color[i]!=color[g[i][j]])//若这两个点不再同一个双连通分量中

++deg[color[i]];//由于是无向图，所以只对起点的度+1即可

}

int ans=0;

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

if(deg[i]==1)

++ans;

return (ans+1)>>1;

}

int main() {

int s,e;

while(scanf("%d%d",&n,&m)==2) {

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

g[i].clear();

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

vis[i][j]=vis[j][i]=false;

}

while(m-->0) {

scanf("%d%d",&s,&e);

if(!vis[s][e]) {//去掉重边

vis[s][e]=vis[e][s]=true;

g[s].push\_back(e);

g[e].push\_back(s);

}

}

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

}

return 0;

}

**线段树**

# Handling the Past

[http://acm.hdu.edu.cn/showproblem.](http://acm.hdu.edu.cn/showproblem.php?pid=4967)[**PHP**](http://lib.csdn.net/base/36)**?pid=4967**

Time Limit: 4000/2000 MS (Java/Others)    Memory Limit: 131072/131072 K (Java/Others)

**Problem Description**

Nowadays, cloud computing is a popular topic among business and technology circles. In cloud computing, everything can be stored in a cloud server, and clients deal with the data by sending operation-requests to the server. However, such client-server mode may cause some problems. For example, if you send two operations op1 and op2 to the server, expecting that op1 should be executed first and followed by op2. Due to the network delay, op1 may arrive at the server later than op2. In order to inform the server the correct operation order, each operation is associated with a timestamp. Now if the server gets two operations (op1, t1) and (op2, t2), where t1 < t2, the server will know that op1 should be executed earlier than op2. So, if (op2, t2) arrives first, the server will execute op2 immediately. And when (op1, t1) arrives, the server will find that op1 should be executed before op2 (because t1 < t2), thus it has to undo op2, execute op1, and re-execute op2 finally.   
  
In this problem, you are asked to simulate the above process. To simplify the problem, we assume that there is only a stack, a last-in-first-out data structure as you know, stored in the server. Three types of operations are considered, whose formats and semantics are given as follows.  
  
push x t -- push x into the stack, and t is a timestamp  
pop t -- pop the top element from the stack, and t is a timestamp  
peak t -- return the top element in the stack, and t is a timestamp  
  
When an operation op with a timestamp t arrives, the server process it in the following three steps:  
  
Step 1: undo all the "push" and "pop" operations having timestamp larger than t.  
Step 2: execute op.  
Step 3: redo all the "push" and "pop" operations which were undone in step 1.  
  
The server do not need to undo or redo any "peak" operations. In another word, every "peak" operation is executed only once after it arrives at the server.  
  
Given the operations arriving at the server in order, you are asked to simulate the above process. The stack is empty initially. To simplify the problem further, another two assumptions are made:  
  
1. All the "pop" operations are valid. In another word, if you simulate the process correctly, no "pop" operations will be performed on an empty stack.  
2. All timestamps are different.

**Input**

The input contains multiple test cases.  
  
Each case begins with an integer N (1<=N<=50000), indicating the number of operations. The following N lines each contain an operation in one of the following three formats:  
  
push x t  
pop t  
peak t  
where 0<=x, t<=10^9.  
  
The operations are given in the order in which they arrive at the server.  
The input is terminated by N = 0.

**Output**

For each case, output "Case #X:" in a line where X is the case number, staring from 1. Then for each "peak" operation, output the answer in a line. If the stack is empty, output -1 instead.

**Sample Input**

7

push 100 3

push 200 7

peak 4

push 50 2

pop 5

peak 6

peak 8

4

push 25 1

pop 5

peak 6

peak 3

4

push 10 1

peak 7

pop 3

peak 4

0

**Sample Output**

Case #1:

100

50

200

Case #2:

-1

25

Case #3:

10

-1

初看本题完全想不到是线段树，看了题解后发现解法好巧妙

对时间按照升序排序后离散化处理，建立线区间[1,n]的线段树，线段树维护区间和与最大后缀和，push操作对相应时间+1，pop操作对相应时间-1，peak操作查询[1,t)区间内最右的tt，使[tt,t)的和大于0

注意：时间1就是操作peak时，直接输出-1

#include <cstdio>

#include <algorithm>

#define lson (i<<1)

#define rson ((i<<1)|1)

using namespace std;

int index[50005],sum,x[50005];

struct Operation {//0代表push，1代表pop，2代表peak

int ope,x,t;

}sta[50005];

struct Node {

int l,r,sum,rmx;

}tr[200005];

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

tr[i].l=l;

tr[i].r=r;

tr[i].sum=tr[i].rmx=0;

if(l==r)

return ;

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

build(lson,l,mid);

build(rson,mid+1,r);

}

void update(int i,int x,int t) {

if(tr[i].l==tr[i].r) {

tr[i].sum=tr[i].rmx=x;

return ;

}

if(tr[lson].r>=t)

update(lson,x,t);

else

update(rson,x,t);

tr[i].sum=tr[lson].sum+tr[rson].sum;

tr[i].rmx=max(tr[rson].rmx,tr[rson].sum+tr[lson].rmx);//当前结点的最大后缀和为 右儿子的最大后缀和 与 右儿子的区间和+左儿子的最大后缀和 的最大值

}

int query(int i,int t) {

if(tr[i].r<=t) {

int ss=sum+tr[i].rmx;

if(ss<=0) {//如果最大后缀和小于等于0

sum+=tr[i].sum;//后缀和加上本区间的和

return -1;

}

}

if(tr[i].l==tr[i].r)

return x[tr[i].l];

if(tr[lson].r<t) {//如果时间t不完全在左子区间

int ans=query(rson,t);

if(ans!=-1)//右区间找到答案

return ans;

}

return query(lson,t);

}

int main() {

int n,kase=0;

char s[5];

while(scanf("%d",&n),n!=0) {

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

scanf("%s",s);

if(s[1]=='u') {

scanf("%d%d",&sta[i].x,&sta[i].t);

sta[i].ope=0;

}

else if(s[1]=='o') {

scanf("%d",&sta[i].t);

sta[i].ope=1;

}

else {

scanf("%d",&sta[i].t);

sta[i].ope=2;

}

index[i]=sta[i].t;

}

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

sort(index,index+n);

for(int i=0;i<n;++i) {//离散化

sta[i].t=lower\_bound(index,index+n,sta[i].t)-index+1;//时间区间为[1,n]

x[sta[i].t]=sta[i].x;

}

build(1,1,n);

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

if(sta[i].ope==0)

update(1,1,sta[i].t);//push的时间+1

else if(sta[i].ope==1)

update(1,-1,sta[i].t);//pop的时间-1

else {

sum=0;

printf("%d\n",sta[i].t==1?-1:query(1,sta[i].t-1));//若时间1就是peak则直接输出-1

}

}

}

return 0;

}

**Tire**

#include <cstdio>

using namespace std;

const int MAXN=3000005;

struct Trie {

int next[MAXN][26];

int num[MAXN];

int l;

const static int root=0;

Trie() {

clear();

}

int newNode() {

num[l]=0;

for(int i=0;i<26;++i) {

next[l][i]=-1;

}

return l++;

}

void insert(char \*p) {

int cur=root;

while((\*p)!='\0') {

if(next[cur][(\*p)-'a']==-1) {

next[cur][(\*p)-'a']=newNode();

}

cur=next[cur][(\*p)-'a'];

++num[cur];

++p;

}

}

void delet(char \*p,int cnt) {

int cur=root;

while((\*p)!='\0') {

cur=next[cur][(\*p)-'a'];

num[cur]-=cnt;

++p;

}

for(int i=0;i<26;++i) {

next[cur][i]=-1;

}

}

int query(char \*p) {

int cur=root;

while((\*p)!='\0') {

if(next[cur][(\*p)-'a']==-1) {

return 0;

}

cur=next[cur][(\*p)-'a'];

++p;

}

return num[cur];

}

void clear() {

l=root;

newNode();

}

}trie;

int main() {

int n;

char cmd[35],s[35];

scanf("%d",&n);

trie.clear();

while(n-->0) {

scanf("%s%s",cmd,s);

if(cmd[0]=='i') {

trie.insert(s);

}

else if(cmd[0]=='d') {

int cnt=trie.query(s);

if(cnt>0) {

trie.delet(s,cnt);

}

}

else {

printf("%s\n",trie.query(s)>0?"Yes":"No");

}

}

return 0;

}

**KMP**

#include <cstdio>

#include <cstring>

using namespace std;

int n,plen,slen,nxt[10005],i,j,k,ans;

char p[10005],s[1000005];

void getNxt() {

nxt[j=0]=k=-1;

while(j<plen) {

if(k==-1||p[j]==p[k]) {//如果k==-1或当前字符匹配成功

if(p[++j]!=p[++k])//如果下一个字符不同

nxt[j]=k;

else

nxt[j]=nxt[k];

}

else

k=nxt[k];

}

}

int kmp() {

getNxt();

i=j=ans=0;

while(i<=slen) {//s串要运行到'\0'，防止s串和p串匹配成功时同时结束而少记一个答案

if(j==plen) {//如果成功匹配，则继续下一次匹配，字符串可重叠

++ans;

j=nxt[j];

}

else if(j==-1||s[i]==p[j]) {//如果j==-1或当前字符匹配成功

++i;

++j;

}

else

j=nxt[j];

}

return ans;

}

int main() {

scanf("%d",&n);

while(n--) {

scanf("%s%s",p,s);

plen=strlen(p);

slen=strlen(s);

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

}

return 0;

}

**AC自动机**

#include <cstdio>

#include <queue>

using namespace std;

const int MAXNODE=10000\*50+5;

int T,n;

char s[1000005];

struct Trie {

int nxt[MAXNODE][26],fail[MAXNODE],ed[MAXNODE];//数组模拟

int l;

const static int root=0;

Trie() {

clear();

}

int newNode() {//取一个结点

for(int i=0;i<26;++i)

nxt[l][i]=-1;

ed[l]=0;//ed[l]表示从根到当前结点的模式串出现的次数

return l++;

}

void insert(char \*p) {//插入一个模式串

int cur=root;

while(\*p) {

if(nxt[cur][\*p-'a']==-1)

nxt[cur][\*p-'a']=newNode();

cur=nxt[cur][\*p-'a'];

++p;

}

++ed[cur];

}

void build() {//建立fail指针

int cur,i;

queue<int> q;

fail[root]=root;

for(i=0;i<26;++i)

if(nxt[root][i]==-1)//如果根结点无i子结点

nxt[root][i]=root;

else {

fail[nxt[root][i]]=root;

q.push(nxt[root][i]);

}

while(!q.empty()) {

cur=q.front();

q.pop();

for(i=0;i<26;++i)

if(nxt[cur][i]==-1)//如果当前结点无i子结点

nxt[cur][i]=nxt[fail[cur]][i];//当前结点的i子结点指针 指向 当前结点fail指针指向的结点的i子结点

else {

fail[nxt[cur][i]]=nxt[fail[cur]][i];//当前结点i子结点的fail指针 指向 当前结点fail指针指向的结点的i子结点

q.push(nxt[cur][i]);

}

}

}

int query(char \*p) {

int cur=root,ans=0,tmp;

while(\*p) {

cur=nxt[cur][\*p-'a'];

tmp=cur;

while(tmp!=root) {//遍历与当前形成的串有公共后缀的串

ans+=ed[tmp];

ed[tmp]=0;//由于只需要统计出现了关键词，所需需要置0，防止重复计算

tmp=fail[tmp];

}

++p;

}

return ans;

}

void clear() {

l=root;

newNode();

}

}ac;

int main() {

scanf("%d",&T);

while(T--) {

ac.clear();

scanf("%d",&n);

while(n--) {

scanf("%s",s);

ac.insert(s);

}

ac.build();

scanf("%s",s);

printf("%d\n",ac.query(s));

}

return 0;

}

**树状数组**

#include <cstdio>

#include <cstring>

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

using namespace std;

int n,m,k,l,r,d,x,ans;

long long sum[100005],damage[100005],h;

void update(int i,int v) {

while(i<=n) {

damage[i]+=v;

i+=lowbit(i);

}

}

long long getSum(int i) {

long long all=0;

while(i>0) {

all+=damage[i];

i-=lowbit(i);

}

return all;

}

**单调栈**

题目大意：给定n个相邻的矩形，宽和高分别为w,h，求这些矩形能构成的矩形的最大面积？

今天知道有单调栈这种[**数据结构**](http://lib.csdn.net/base/31)

用单调递增栈很容易就能求出以h[i]为最小值的区间

大致过程如下：

若h[i]<s[top].h，则不停弹出栈顶元素，直至h[i]>=s[top].h，且此h[i]的起始位置为弹出的最后一个元素的起始位置；

否则直接进栈，其起始位置为i

#include <cstdio>

#include <algorithm>

using namespace std;

int n,w,h[50005],top,oriTop;

long long sumW[50005],ans;//sumW[i]表示w在区间[1,i]上的和

struct Node {

int h,sta;//sta表示高度h的起始下标

}s[50005];

int main() {

while(scanf("%d",&n),n!=-1) {

sumW[0]=0;

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

scanf("%d%d",&w,h+i);

sumW[i]=sumW[i-1]+w;

}

h[++n]=-1;//令最后一个元素的下一个高度为-1，避免循环完毕后还要弹出栈中所有元素

s[0].h=-1;

s[0].sta=top=0;

ans=0;

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

oriTop=top;

while(h[i]<s[top].h) {

ans=max(ans,(sumW[i-1]-sumW[s[top].sta-1])\*s[top].h);

--top;//弹出栈顶元素

}

s[++top].h=h[i];

if(oriTop<top)//如果没有元素弹出，则其起始下标就是自己的下标，否则其起始下标是弹出的最后一个元素的起始下标

s[top].sta=i;

}

printf("%I64d\n",ans);

}

return 0;

}

**最小环**

大致思路：无向图的环最少有三个点，所以需要增加一部分求最小环；枚举中间点k，在用其更新最短路前，先找最小环，令1<=i<j<k，即k点必定不在i,j的最短路上，则这个环中至少有三个点，这个环的权值为：dis[i][j]+g[i][k]+g[k][j]

没仔细看样例，会存在重边，结果WA了

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

int dis[105][105],g[105][105],n,m,ans;

void floyd() {

ans=0x1f1f1f1f;

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

for(int i=1;i<k;++i)//更新最小环

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

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

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

for(int j=1;j<=n;++j)//更新最短路

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

}

}

int main() {

int s,e,w;

while(scanf("%d%d",&n,&m)==2) {

memset(g,0x1f,sizeof(g));

memset(dis,0x1f,sizeof(dis));

while(m-->0) {

scanf("%d%d%d",&s,&e,&w);

if(w<g[s][e])

dis[s][e]=dis[e][s]=g[s][e]=g[e][s]=w;

}

floyd();

if(ans==0x1f1f1f1f)

printf("It's impossible.\n");

else

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

}

return 0;

}

**最大流**

题目大意：给定np个生产者，nc个消费者，m条能量转移的边，求最多能消耗的能量？

添加一个虚拟源点和一个虚拟汇点，添加源点到生产者的边（权值为生产者能生产的能量），添加消费者到汇点的边（权值为消费者能消费的能量），则转化为最大流模版题。

刚开始用STL库的queue和priority\_queue，结果都超时了，第一次被STL的慢坑住了。。。换成数组模拟的队列800ms AC

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

using namespace std;

const int MAXN=105;

const int INF=0x3f3f3f3f;

struct Node {

int pre,u,mn;//u表示当前节点，mn表示从源点到该点的流量上限中最小的

}cur,q[MAXN\*MAXN];

int n,np,nc,m,s,e,l,STA,DES,head,tail;

int g[MAXN][MAXN],pre[MAXN];

bool vis[MAXN];

int bfs(int sta,int des) {//用bfs找到流量最大的一条增广路径，并返回这条增广路径的流量

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

head=tail=0;

q[tail].pre=-1;

q[tail].u=sta;

q[tail++].mn=INF;

while(head!=tail) {

do {

cur=q[head++];

}while(head!=tail&&vis[cur.u]);

if(vis[cur.u]) {//如果无法到达汇点，则返回0

return 0;

}

vis[cur.u]=true;

pre[cur.u]=cur.pre;

if(cur.u==des) {

return cur.mn;

}

for(int i=0;i<=n;++i) {//枚举可流向的下一个点

if(!vis[i]&&g[cur.u][i]!=0) {

q[tail].pre=cur.u;

q[tail].u=i;

q[tail++].mn=min(cur.mn,g[cur.u][i]);

}

}

}

return 0;

}

int Ford\_Fulkerson(int sta,int des) {//当可增加的流量不为0时，继续算法

int mn,e,ans=0;

while(mn=bfs(sta,des),mn!=0) {

ans+=mn;

e=des;

while(e!=sta) {

g[pre[e]][e]-=mn;//正向的边减去相应的流量

g[e][pre[e]]+=mn;//反向的边加上相应的流量

e=pre[e];

}

}

return ans;

}

int main() {

while(4==scanf("%d%d%d%d",&n,&np,&nc,&m)) {

STA=n;

DES=++n;

memset(g,0,sizeof(g));

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

scanf(" (%d,%d)%d",&s,&e,&l);

g[s][e]+=l;

}

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

scanf(" (%d)%d",&e,&l);

g[STA][e]+=l;

}

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

scanf(" (%d)%d",&s,&l);

g[s][DES]+=l;

}

printf("%d\n",Ford\_Fulkerson(STA,DES));

}

return 0;

}

**负权回路**

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int MAXN=505;

const int INF=0x3f3f3f3f;

struct Edge {

int s,e,v;

Edge(int ss=0,int ee=0,int vv=0):s(ss),e(ee),v(vv) {}

}u;

int n,m,w,s,e,v;

vector<Edge> edge;

int dis[MAXN];

bool Bellman\_Ford(int sta) {//可判断负权回路

bool relaxed;

memset(dis,0x3f,sizeof(dis));

dis[sta]=0;

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

relaxed=false;

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

if(dis[edge[j].s]+edge[j].v<dis[edge[j].e]) {//松弛

dis[edge[j].e]=dis[edge[j].s]+edge[j].v;

relaxed=true;

}

}

if(!relaxed) {//如果未更新，则不会再更新，且无负权回路

return false;

}

}

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

if(dis[edge[j].s]+edge[j].v<dis[edge[j].e]) {//如果可以继续松弛，则存在负权回路

return true;

}

}

return false;

}

int main() {

int F;

scanf("%d",&F);

while(F-->0) {

edge.clear();

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

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

scanf("%d%d%d",&s,&e,&v);

edge.push\_back(Edge(s,e,v));

edge.push\_back(Edge(e,s,v));

}

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

scanf("%d%d%d",&s,&e,&v);

edge.push\_back(Edge(s,e,-v));

}

printf("%s\n",Bellman\_Ford(1)?"YES":"NO");

}

return 0;

}

1. **快速幂**
2. **矩阵快速幂**
3. **圆相交面积**
4. **Lucas定理（组合数取模）**
5. **素数测试**
6. **三维计算几何**

**快速幂**

下面是 m^n % k 的快速幂：

// m^n % k

int quickpow(int m,int n,int k) { **//注意数据类型**

int b = 1;

while (n > 0) {

if (n & 1)

b = (b\*m)%k;

n = n >> 1 ;

m = (m\*m)%k;

}

return b;

}

**矩阵快速幂**

/\*===================================\*/

|| 快速幂（quickpow）模板

|| P 为等比，I 为单位矩阵

|| MAX 要初始化！！！！

||

/\*===================================\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int MAX=4;

const long long MOD=1000000007;

struct Matrix{

long long m[MAX][MAX];

};

Matrix P = {3,1,0,0,

-2,0,1,0,

-1,1,0,1,

1,0,0,0};

Matrix I = {1,0,0,0,

0,1,0,0,

0,0,1,0,

0,0,0,1};

Matrix matrixmul(Matrix a,Matrix b) {//矩阵乘法

int i,j,k;

Matrix c;

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

for (j = 0; j < MAX;j++) {

c.m[i][j] = 0;

for (k = 0; k < MAX; k++)

c.m[i][j] += (a.m[i][k] \* b.m[k][j])%MOD;

c.m[i][j] %= MOD;

}

return c;

}

Matrix quickpow(long long n) {

Matrix m = P, b = I;

while (n >= 1) {

if (n & 1)

b = matrixmul(b,m);

n = n >> 1;

m = matrixmul(m,m);

}

return b;

}

**求圆相交面积**

struct point

{

double x;

double y;

}p[N];

double r[N];

int n;

double dis(point a,point b)

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

double interarea(point a,double ra,point b,double rb) //求圆相交面积的模板 重点

{

double ans=0;

double d=dis(a,b);

double temp;

if(ra<rb)

{

temp=ra;

ra=rb;

rb=temp;

}

if(d>=ra+rb)return 0; //相离

if(d<=ra-rb)return PI\*rb\*rb; //内含

double angle1=acos((ra\*ra+d\*d-rb\*rb)/2.0/ra/d);

double angle2=acos((rb\*rb+d\*d-ra\*ra)/2.0/rb/d);

ans-=d\*ra\*sin(angle1);

ans+=angle1\*ra\*ra+angle2\*rb\*rb;

return ans;

}

**Lucas定理（组合数取模）**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long LL;

const LL MOD=1e9+7;

LL n,m;

LL quick\_pow(LL a, LL b) {

LL ans = 1;

a %= MOD;//防止指数为0时，返回未取模的结果

while(b!=0) {

if((b & 1)==1) {

ans = ans \* a % MOD;

}

b >>= 1;

a = a \* a % MOD;

}

return ans;

}

LL C(LL n, LL m) {

if(n < m) {

return 0;

}

if(n == m) {

return 1;

}

if(m > n - m) {//利用组合数性质，减少运算次数

m=n - m;

}

LL ans, a = 1, b = 1;

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

a = (a \* (n - i)) % MOD;

b = (b \* (m - i)) % MOD;

}

ans = (a \* quick\_pow(b, MOD-2)) % MOD;

return ans;

}

LL Lucas(LL n, LL m) {

LL ans=1;

while(m>0) {

ans=(ans\*C(n % MOD, m % MOD)) % MOD;

n/=MOD;

m/=MOD;

}

return ans;

}

int main() {

return 0;

}

**Miller-Rabin素数测试**

**第一种：速度快，代码简洁，枚举10^8用时40s左右【判断10^8以内的素数够用】**

bool witness(long long a,long long n) {

long long d=1,x;

int i=ceil(log(n-1.0)/log(2.0))-1; //ceil()是向上取整

for(;i>=0;i--) {

x=d;

d=(d\*d)%n;

if(d==1&&x!=1&&x!=n-1)

return true;

if(((n-1)&(1<<i))>0)

d=(d\*a)%n;

}

return d!=1;

}

bool miller\_rabin(long long n) {

int s[]={2,3,5}; //只用2，3，5和下面的特例判断就能正确判断10^8以内的所有素数，枚举一遍用时40s左右

if(n==2||n==3||n==5||n==7)

return true;

if(n==1||(n&1)==0||0==n%3||0==n%5||n==4097||n==1048577||n==16777217||n==25326001)

return false;

for(int i=0;i<3;i++)

if(witness(s[i], n))

return false;

return true;

}

**第二种：正确率高【2,7,61对unsigned int内的所有数够用了，最小不能判断的数为4 759 123 141；用2,3,7,61在 10^16 内唯一不能判断的数是 46 856 248 225 981】**

typedef unsigned long long LL;

LL modular\_multi(LL x,LL y,LL mo) {

LL t;

x%=mo;

for(t=0;y;x=(x<<1)%mo,y>>=1)

if (y&1)

t=(t+x)%mo;

return t;

}

LL modular\_exp(LL num,LL t,LL mo) {

LL ret=1,temp=num%mo;

for(;t;t>>=1,temp=modular\_multi(temp,temp,mo))

if (t&1)

ret=modular\_multi(ret,temp,mo);

return ret;

}

bool miller\_rabin(LL n) {

if (n==2||n==7||n==61)

return true;

if (n==1||(n&1)==0)

return false;

int t=0,num[3]={2,7,61}; //2,7,61对unsigned int内的所有数够用了，最小不能判断的数为4 759 123 141；用2,3,7,61在 10^16 内唯一不能判断的数是 46 856 248 225 981

LL a,x,y,u=n-1;

while((u&1)==0)

t++,u>>=1;

for(int i=0;i<3;i++) {

a=num[i];

x=modular\_exp(a,u,n);

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

y=modular\_multi(x,x,n);

if (y==1&&x!=1&&x!=n-1)

return false;

//其中用到定理，如果对模n存在1的非平凡平方根，则n是合数。

//如果一个数x满足方程x^2≡1 (mod n),但x不等于对模n来说1的两个‘平凡’平方根：1或-1，则x是对模n来说1的非平凡平方根

x=y;

}

if (x!=1) //根据费马小定理,若n是素数，有a^(n-1)≡1(mod n).因此n不可能是素数

return false;

}

return true;

}

**三维计算几何**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*基础\*\*\*\*\*\*\*\*\*\*\*\*\*/

const double EPS=0.000001;

typedef struct Point\_3D {

double x, y, z;

Point\_3D(double xx = 0, double yy = 0, double zz = 0): x(xx), y(yy), z(zz) {}

bool operator == (const Point\_3D& A) const {

return x==A.x && y==A.y && z==A.z;

}

}Vector\_3D;

Point\_3D read\_Point\_3D() {

double x,y,z;

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

return Point\_3D(x,y,z);

}

Vector\_3D operator + (const Vector\_3D & A, const Vector\_3D & B) {

return Vector\_3D(A.x + B.x, A.y + B.y, A.z + B.z);

}

Vector\_3D operator - (const Point\_3D & A, const Point\_3D & B) {

return Vector\_3D(A.x - B.x, A.y - B.y, A.z - B.z);

}

Vector\_3D operator \* (const Vector\_3D & A, double p) {

return Vector\_3D(A.x \* p, A.y \* p, A.z \* p);

}

Vector\_3D operator / (const Vector\_3D & A, double p) {

return Vector\_3D(A.x / p, A.y / p, A.z / p);

}

double Dot(const Vector\_3D & A, const Vector\_3D & B) {

return A.x \* B.x + A.y \* B.y + A.z \* B.z;

}

double Length(const Vector\_3D & A) {

return sqrt(Dot(A, A));

}

double Angle(const Vector\_3D & A, const Vector\_3D & B) {

return acos(Dot(A, B) / Length(A) / Length(B));

}

Vector\_3D Cross(const Vector\_3D & A, const Vector\_3D & B) {

return Vector\_3D(A.y \* B.z - A.z \* B.y, A.z \* B.x - A.x \* B.z, A.x \* B.y - A.y \* B.x);

}

double Area2(const Point\_3D & A, const Point\_3D & B, const Point\_3D & C) {

return Length(Cross(B - A, C - A));

}

double Volume6(const Point\_3D & A, const Point\_3D & B, const Point\_3D & C, const Point\_3D & D) {

return Dot(D - A, Cross(B - A, C - A));

}

**// 四面体的重心**

Point\_3D Centroid(const Point\_3D & A, const Point\_3D & B, const Point\_3D & C, const Point\_3D & D) {

return (A + B + C + D) / 4.0;

}

**/\*\*\*\*\*\*\*\*\*\*\*\*点线面\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// 点p到平面p0-n的距离。n必须为单位向量**

double DistanceToPlane(const Point\_3D & p, const Point\_3D & p0, const Vector\_3D & n)

{

return fabs(Dot(p - p0, n)); // 如果不取绝对值，得到的是有向距离

}

**// 点p在平面p0-n上的投影。n必须为单位向量**

Point\_3D GetPlaneProjection(const Point\_3D & p, const Point\_3D & p0, const Vector\_3D & n)

{

return p - n \* Dot(p - p0, n);

}

**//直线p1-p2 与平面p0-n的交点**

Point\_3D LinePlaneIntersection(Point\_3D p1, Point\_3D p2, Point\_3D p0, Vector\_3D n)

{

Vector\_3D v= p2 - p1;

double t = (Dot(n, p0 - p1) / Dot(n, p2 - p1)); //分母为0，直线与平面平行或在平面上

return p1 + v \* t; //如果是线段 判断t是否在0~1之间

}

**// 点P到直线AB的距离**

double DistanceToLine(const Point\_3D & P, const Point\_3D & A, const Point\_3D & B)

{

Vector\_3D v1 = B - A, v2 = P - A;

return Length(Cross(v1, v2)) / Length(v1);

}

**//点到线段的距离**

double DistanceToSeg(Point\_3D p, Point\_3D a, Point\_3D b)

{

if(a == b)

{

return Length(p - a);

}

Vector\_3D v1 = b - a, v2 = p - a, v3 = p - b;

if(Dot(v1, v2) + EPS < 0)

{

return Length(v2);

}

else

{

if(Dot(v1, v3) - EPS > 0)

{

return Length(v3);

}

else

{

return Length(Cross(v1, v2)) / Length(v1);

}

}

}

**//求异面直线 p1+s\*u与p2+t\*v的公垂线对应的s 如果平行|重合，返回false**

bool LineDistance3D(Point\_3D p1, Vector\_3D u, Point\_3D p2, Vector\_3D v, double & s)

{

double b = Dot(u, u) \* Dot(v, v) - Dot(u, v) \* Dot(u, v);

if(abs(b) <= EPS)

{

return false;

}

double a = Dot(u, v) \* Dot(v, p1 - p2) - Dot(v, v) \* Dot(u, p1 - p2);

s = a / b;

return true;

}

**// p1和p2是否在线段a-b的同侧**

bool SameSide(const Point\_3D & p1, const Point\_3D & p2, const Point\_3D & a, const Point\_3D & b)

{

return Dot(Cross(b - a, p1 - a), Cross(b - a, p2 - a)) - EPS >= 0;

}

**// 点P在三角形P0, P1, P2中**

bool PointInTri(const Point\_3D & P, const Point\_3D & P0, const Point\_3D & P1, const Point\_3D & P2)

{

return SameSide(P, P0, P1, P2) && SameSide(P, P1, P0, P2) && SameSide(P, P2, P0, P1);

}

**// 三角形P0P1P2是否和线段AB相交**

bool TriSegIntersection(const Point\_3D & P0, const Point\_3D & P1, const Point\_3D & P2, const Point\_3D & A, const Point\_3D & B, Point\_3D & P)

{

Vector\_3D n = Cross(P1 - P0, P2 - P0);

if(abs(Dot(n, B - A)) <= EPS)

{

return false; // 线段A-B和平面P0P1P2平行或共面

}

else // 平面A和直线P1-P2有惟一交点

{

double t = Dot(n, P0 - A) / Dot(n, B - A);

if(t + EPS < 0 || t - 1 - EPS > 0)

{

return false; // 不在线段AB上

}

P = A + (B - A) \* t; // 交点

return PointInTri(P, P0, P1, P2);

}

}

**//空间两三角形是否相交**

bool TriTriIntersection(Point\_3D \* T1, Point\_3D \* T2)

{

Point\_3D P;

for(int i = 0; i < 3; i++)

{

if(TriSegIntersection(T1[0], T1[1], T1[2], T2[i], T2[(i + 1) % 3], P))

{

return true;

}

if(TriSegIntersection(T2[0], T2[1], T2[2], T1[i], T1[(i + 1) % 3], P))

{

return true;

}

}

return false;

}

**//空间两直线上最近点对 返回最近距离 点对保存在ans1 ans2中**

double SegSegDistance(Point\_3D a1, Point\_3D b1, Point\_3D a2, Point\_3D b2, Point\_3D& ans1, Point\_3D& ans2)

{

Vector\_3D v1 = (a1 - b1), v2 = (a2 - b2);

Vector\_3D N = Cross(v1, v2);

Vector\_3D ab = (a1 - a2);

double ans = Dot(N, ab) / Length(N);

Point\_3D p1 = a1, p2 = a2;

Vector\_3D d1 = b1 - a1, d2 = b2 - a2;

double t1, t2;

t1 = Dot((Cross(p2 - p1, d2)), Cross(d1, d2));

t2 = Dot((Cross(p2 - p1, d1)), Cross(d1, d2));

double dd = Length((Cross(d1, d2)));

t1 /= dd \* dd;

t2 /= dd \* dd;

ans1 = (a1 + (b1 - a1) \* t1);

ans2 = (a2 + (b2 - a2) \* t2);

return fabs(ans);

}

**// 判断P是否在三角形A, B, C中，并且到三条边的距离都至少为mindist。保证P, A, B, C共面**

bool InsideWithMinDistance(const Point\_3D & P, const Point\_3D & A, const Point\_3D & B, const Point\_3D & C, double mindist)

{

if(!PointInTri(P, A, B, C))

{

return false;

}

if(DistanceToLine(P, A, B) < mindist)

{

return false;

}

if(DistanceToLine(P, B, C) < mindist)

{

return false;

}

if(DistanceToLine(P, C, A) < mindist)

{

return false;

}

return true;

}

**// 判断P是否在凸四边形ABCD（顺时针或逆时针）中，并且到四条边的距离都至少为mindist。保证P, A, B, C, D共面**

bool InsideWithMinDistance(const Point\_3D & P, const Point\_3D & A, const Point\_3D & B, const Point\_3D & C, const Point\_3D & D, double mindist)

{

if(!PointInTri(P, A, B, C))

{

return false;

}

if(!PointInTri(P, C, D, A))

{

return false;

}

if(DistanceToLine(P, A, B) < mindist)

{

return false;

}

if(DistanceToLine(P, B, C) < mindist)

{

return false;

}

if(DistanceToLine(P, C, D) < mindist)

{

return false;

}

if(DistanceToLine(P, D, A) < mindist)

{

return false;

}

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*凸包相关问题\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//加干扰

double rand01()

{

return rand() / (double)RAND\_MAX;

}

double randeps()

{

return (rand01() - 0.5) \* EPS;

}

Point\_3D add\_noise(const Point\_3D & p)

{

return Point\_3D(p.x + randeps(), p.y + randeps(), p.z + randeps());

}

struct Face

{

int v[3];

Face(int a, int b, int c)

{

v[0] = a;

v[1] = b;

v[2] = c;

}

Vector\_3D Normal(const vector<Point\_3D> & P) const

{

return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);

}

// f是否能看见P[i]

int CanSee(const vector<Point\_3D> & P, int i) const

{

return Dot(P[i] - P[v[0]], Normal(P)) > 0;

}

};

// 增量法求三维凸包

// 注意：没有考虑各种特殊情况（如四点共面）。实践中，请在调用前对输入点进行微小扰动

vector<Face> CH3D(const vector<Point\_3D> & P)

{

int n = P.size();

vector<vector<int> > vis(n);

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

{

vis[i].resize(n);

}

vector<Face> cur;

cur.push\_back(Face(0, 1, 2)); // 由于已经进行扰动，前三个点不共线

cur.push\_back(Face(2, 1, 0));

for(int i = 3; i < n; i++)

{

vector<Face> next;

// 计算每条边的“左面”的可见性

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

{

Face & f = cur[j];

int res = f.CanSee(P, i);

if(!res)

{

next.push\_back(f);

}

for(int k = 0; k < 3; k++)

{

vis[f.v[k]][f.v[(k + 1) % 3]] = res;

}

}

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

for(int k = 0; k < 3; k++)

{

int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];

if(vis[a][b] != vis[b][a] && vis[a][b]) // (a,b)是分界线，左边对P[i]可见

{

next.push\_back(Face(a, b, i));

}

}

cur = next;

}

return cur;

}

struct ConvexPolyhedron

{

int n;

vector<Point\_3D> P, P2;

vector<Face> faces;

bool read()

{

if(scanf("%d", &n) != 1)

{

return false;

}

P.resize(n);

P2.resize(n);

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

{

P[i] = read\_Point\_3D();

P2[i] = add\_noise(P[i]);

}

faces = CH3D(P2);

return true;

}

//三维凸包重心

Point\_3D centroid()

{

Point\_3D C = P[0];

double totv = 0;

Point\_3D tot(0, 0, 0);

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

{

Point\_3D p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];

double v = -Volume6(p1, p2, p3, C);

totv += v;

tot = tot + Centroid(p1, p2, p3, C) \* v;

}

return tot / totv;

}

//凸包重心到表面最近距离

double mindist(Point\_3D C)

{

double ans = 1e30;

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

{

Point\_3D p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];

ans = min(ans, fabs(-Volume6(p1, p2, p3, C) / Area2(p1, p2, p3)));

}

return ans;

}

};

Rylynnn

一．数论

1. gcd&exgcd

**题意**

有n个虫洞，每个虫洞有三个参数X、Y、Z，如果存在一个ID，可以使ID%X在[Y,Z]区间内，则这个虫洞将可以吸引飞船，但是，如果同时两个或者两个以上的虫洞可以吸引飞船，那么飞船将被撕碎，不能通过。

**分析**

由于我们的目的是：确定每两个虫洞的参数之间不存在一个ID，ID满足：  
A*x1+C=ID  
B*X2+D=ID  
X1，X2是两个虫洞的X参数，C在Y1,Z1之间取值，D在Y2，Z2之间取值。  
然后，我当时就。。卡住了，把这个式子转化成了两个同余方程，但是其实不用计算ID，因为我们的目的是存在ID，然后C,D都是存在的，所以只要A,B存在就可以了，所以联立式子变成一个二元一次不定方程，然后根据数论的不定方程的相关知识：  
A*X1-B*x2=D-C  
X1≡(D-C)(mod X2)  
这个同余方程有解的条件是(D-C)%gcd(x1,x2)==0  
然后对每两组虫洞的参数，看在D-C取值范围内的是否存在gcd(x1,x2)的整数倍，队友是用容斥判断的，我写的是分类讨论，虽然被基友说过很多次了。。不要老用分类讨论，要证明要证明，分类讨论通常会漏掉很多情况（比如上次的西电校赛网络赛的贪心。。），嘛这次以后！！！我会改的！！！（不过很多情况下第一反应还是分类讨论啊。。

#include <bits/stdc++.h>  
#define N 1007  
using namespace std;  
long long x[N],y[N],z[N],n;  
long long gcd(long long a,long long b){  
 return (b==0)?a:gcd(b,a%b);  
}  
bool solve(int i,int j){  
 int mn,mx,mmn,mmx,g;  
 g=gcd(x[i],x[j]);  
 mn=y[i]-z[j];//可以取的gcd的范围  
 mx=z[i]-y[j];  
 //cout<<g<<endl;  
 if(g>=min(fabs(mx),fabs(mn))&&g<=max(fabs(mx),fabs(mn))){  
 return true;  
 }  
 else if(g<min(fabs(mx),fabs(mn))){  
 mmn=mn/g;  
 mmx=mx/g;//cout<<mmx<<' '<<mmn;  
 return mmn!=mmx;  
 }  
 else if(g>max(fabs(mx),fabs(mn))){  
 if(0>=min(mx,mn)&&0<=max(mx,mn)){  
 return true;  
 }  
 return false;  
 }  
}  
int main()  
{  
 while(scanf("%d",&n)!=EOF){  
 int p=0;  
 for(int i=0;i<n;i++){  
 scanf("%I64d%I64d%I64d",&x[i],&y[i],&z[i]);  
 for(int j=0;j<i;j++){//cout<<solve(i,j);  
 if(solve(i,j)){  
 p=1;  
 }  
 }  
 }  
 if(p==1){  
 printf("Cannot Take off\n");  
 }  
 else{  
 printf("Can Take off\n");  
 }  
  
 }  
 return 0;  
}

## 题意

青蛙1从x开始跳，每次跳m，青蛙2从y开始跳，每次跳n，环行线总长L

## 分析

裸的二元一次不定方程，如果两个青蛙的步数相同，则永远不可能，若(x-y)%gcd(n-m,l)!=0则永无正整数解

(x+pm)-(y+pn)=kl

p(n-m)+kl=x-y

wa1:gcd(a,b)应当存起来。。因为后面会改变  
wa2：因为a,b可能是负数，所以原本的gcd就是写错了。。。

## 题意

满足

ax+by+c = 0

的x和y在(x1,x2)和(y1,y2)范围内的有多少组解

## 分析

照旧c放到方程右边，然后首先要分类讨论

1.a==0 && b==0  
2.a==0 && b!=0  
3.a!-0 && b==0  
4.a!=0 && b!=0

对第四种情况，用扩展欧几里得，然后将x1,x2,y1,y2带入求出t的范围，然后取重合区间

## 题意

求满足Xa + Yb = 1的最小的非负正整数x，同时输出x和对应的y。

## 分析

裸题,判定一下gcd(a,b)是不是等于1，然后直接处理x的非负就行了。

#include <bits/stdc++.h>  
#define ll long long  
using namespace std;  
ll gcd(ll a,ll b){  
 return b==0?a:gcd(b,a%b);  
}  
void exgcd(ll a,ll b,ll& x,ll& y){  
 if(b==0){  
 x=1;y=0;  
 return;  
 }  
 exgcd(b,a%b,x,y);  
 ll temp=x;  
 x=y;  
 y=temp-a/b\*y;  
 return;  
}  
int main()  
{  
 ll a,b,x,y;  
 while(scanf("%I64d%I64d",&a,&b)!=EOF){  
 if(gcd(a,b)!=1){  
 printf("sorry\n");  
 }  
 else{  
 exgcd(a,b,x,y);  
 if(x<0){  
 x+=b;  
 y-=a;  
 }  
 printf("%I64d %I64d\n",x,y);  
 }  
 }  
 return 0;  
}

## 题意

要求(A/B)%9973，但由于A很大，我们只给出n(n=A%9973)(A必能被B整除，且gcd(B,9973) = 1)。

## 分析

（k\*9973+n）/ B = q\*9973+r

9973^q\*B+r\*B = 9973\*k+n

9973\*(q\*B-k)+r\*B = n

## 题意

计算以下for语句mod2^k的循环次数

for (variable = A; variable != B; variable += C)

## 分析

(A+C\*X)%2^k=B

A+C\*x=2^k\*y+B

C\*x-2^k\*y=B-A

}  
 return 0;  
}

1. 中国剩余定理
2. 欧拉函数和逆元

---

title: 逆元和欧拉函数

date: 2016-09-06

categories:

- 数学

- 数论

- 逆元和欧拉函数

tags:

- 学习笔记

- Templates

toc: true

---

## 定义

对于正整数a和m，如果有ax≡1(mod m)，那么把这个同余方程中的最小正整数解叫做a模m的逆元。

## 前提

求amod m意义下的逆元,要求a与m互质,否则不存在乘法逆元

## 欧拉定理

### 欧拉函数

#### O(√n)求单个数的欧拉函数

\*\*欧拉公式的延伸：一个数的所有质因子之和是euler(n) \* n/2。\*\*

```

#include<cstdio>

#include<cstring>

#include<algorithm>

#define MAXN 200010

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

int n,i,temp;

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

temp=n;

for(i=2;i\*i<=n;i++){

if(n%i==0){

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

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

}

if(n<i+1)

break;

}

if(n>1)

temp=temp/n\*(n-1);

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

}

return 0;

}

```

#### O(n)求1~n的素数和欧拉函数

```

int prime[N],phi[N],cnt;// prime:记录质数，phi记录欧拉函数

int Min\_factor[N];// i的最小素因子

bool vis[N];

void Init()

{

cnt=0;

phi[1]=1;

int x;

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

if(!vis[i]) {

prime[++cnt]=i;

phi[i]=i-1;

Min\_factor[i]=i;

}

for(int k=1;k<=cnt&&prime[k]\*i<N;k++) {

x=prime[k]\*i;

vis[x]=true;

Min\_factor[x]=prime[k];

if(i%prime[k]==0){

phi[x]=phi[i]\*prime[k];

break;

}

else phi[x]=phi[i]\*(prime[k]-1);

}

}

}

```

### 欧拉函数求逆元

## Extended\_GCD

## O(N)求1~n < p的所有逆元

```

#include<cstdio>

#include<cstring>

#include<algorithm>

#define MAXN 200010

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

int n,mod;

int x[MAXN];

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

x[1]=1;

for(int i=2;i<=n;i++){//在最后求值的时候，对1~n的每个数先取模，再求其x[i%m]的值

x[i]=((mod-mod/i)\*x[mod%i])%mod;

}

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

printf("%d ",x[i%mod]);

}

}

return 0;

}

```

## O(n)求n!的逆元

预处理出来n的阶乘存在fac[n]里面，然后通过计算n!的逆元，倒着求出每个i!(i < n)的逆元。

```

#include<cstdio>

#include<cstring>

#include<algorithm>

#define MAXN 200010

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

int n,mod;

int x[MAXN];

fac[0]=fac[1]=1;

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

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

}

inv[MAXN]=pow\_mod(fac[MAXN],phi(mod)-1);//mod为质数时为phi(mod)=mod-1

for(int i=MAXN-1;i>=0;i--){

inv[i]=inv[i+1]\*(i+1)%mod;

}

return 0;

}

```

## 题意

给你A，B求A^B的所有因子的和模9901的值

## 分析

将A进行唯一分解，因为A在5^1e7，所以A的质因子在1e4以内，我们预处理出1e4的素数表，然后对每个A计算其质因子的系数，又由生成函数可知，A^B的所有质因数的和为：

$$sum=(1+p\_1+p\_1^1+p\_1^2+p\_1^3\dots+p\_1^{a\_1B})(1+p\_2+p\_2^1+p\_2^2+p\_2^3\dots+p\_2^{a\_2B})\dots(1+p\_n+p\_n^1+p\_n^2+p\_n^3\dots+p\_n^{a\_nB})$$

计算

$$\frac{p\_1^{a\_1B+1}-1}{p\_1-1}$$

需要用到乘法逆元，并同时需要处理当$p\_1-1$与9901不同余的时候的状况，而且其中因为快速幂的值可能过大，需要快速乘加速。

```

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<iostream>

#include<cmath>

#define MAXN 10007

#define MOD 9901

#define ll long long

using namespace std;

ll check[MAXN],prime[MAXN];

ll getprime(){

ll tot=0;

memset(check,false,sizeof(check));

check[0]=check[1]=true;

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

if(!check[i]){

prime[tot++]=i;

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

check[j]=true;

}

}

}

return tot;

}

ll getprime\_Euler(){

ll tot=0;

memset(check,false,sizeof(check));

check[0]=check[1]=true;

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

if(!check[i]){

prime[tot++]=i;

}

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

if(i\*prime[j]>MAXN){

break;

}

check[i\*prime[j]]=true;

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

break;

}

}

}

return tot;

}

ll mul\_mod(ll a,ll b,ll m){//wa:mul\_mod写错了。。。

ll ans=0;

a%=m;

while(b>0){

if(b&1){

ans+=a;

ans%=m;

}

b=b>>1;

a+=a;

a%=m;

}

return ans;

}

ll pow\_mod(ll a,ll b,ll m){

ll ans=1;

a%=m;

while(b>0){

if(b&1){

ans=mul\_mod(ans,a,m)%m;

}

b=b>>1;

a=mul\_mod(a,a,m)%m;

}

return ans;

}

int main()

{

ll a,b,m,num,ans;

ans=1;

ll tot=getprime\_Euler();

scanf("%lld%lld",&a,&b);

for(int i=0;prime[i]\*prime[i]<=a;i++){

if(a%prime[i]==0){

num=0;

while(a%prime[i]==0){

num++;

a/=prime[i];

}

if((prime[i]-1)%MOD==0){

/\* 如果此时prime[i]-1是素数9901的整数倍，假设a,则 (a-1) mod 9901 = 0

a^0 + a^1 + a^2 + ... + a^n = 1 + ... + (a-1)^n + (a-1)^(n-1) + ... + (a-1)^0 = n + 1

所以可以得出以下式子

\*/

ans=(ans\*((num\*b+1)%MOD))%MOD;

}

else{

m=MOD\*(prime[i]-1);

//cout<<m<<endl;

ans=ans\*((pow\_mod(prime[i],num\*b+1,m)+m-1)/(prime[i]-1))%MOD;

}

}

}

if(a!=1){

/\*如果a是质数则需要用这样的判断方法，这样筛素数的时候，就只用筛√n以内的素数，就可以了\*/

if((a-1)%MOD==0){

ans=(ans\*((b+1)%MOD)%MOD);

}

else{

m=MOD\*(a-1);

//cout<<m<<endl;

ans=ans\*((pow\_mod(a,b+1,m)+m-1)/(a-1))%MOD;

}

}

printf("%lld\n",(ans+MOD)%MOD);

}

```

## 题意

给定一个n,然后让你从1-n中选出某些数乘起来，使得乘积最大，并且乘积必须是完全平方数

## 分析

首先我们预处理出来1e7以内的所有素数，并且预处理出，n在1e7以内的所有n!的结果，存在fac[n]中。

根据n!素因子分解中素数p的幂为 [n/p]+[n/(p^2)]+[n/(p^3)]+......求出所有n!素因子的幂以后，对其中每个素因子的幂的个数进行判断，如果个数为奇数，则最后的fac[n]要对除以这个素因子，将所有需要被除去的素因子乘起来存在ans中，最后因为(fac[n]/ans)%MOD，不能直接除，所以求出ans的逆元：ans^(MOD-1)，用这个结果乘以fac[n]就是最终的结果。

```

#include <bits/stdc++.h>

#define MAXN 10000007

#define MOD 1000000007

#define ll long long

using namespace std;

ll prime[MAXN],fac[MAXN], tot,n;

bool notPrime[MAXN];

void getPrime() {

tot=0;

memset(notPrime, false, sizeof(notPrime));

notPrime[0] = notPrime[1] = true;

for(ll i = 2 ; i < MAXN ; ++i) {

if(!notPrime[i]) {

prime[tot++] = i;

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

notPrime[j] = true;

}

}

}

fac[1]=1;

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

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

}

}

ll getnum(ll x){

ll now=n,sum=0;

while(now){

sum+=(now/=x);

}

return sum;

}

ll gcd(ll a,ll b){

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

}

ll phi(ll k)

{

ll i;

ll result = k;

ll t = (ll)sqrt(k\*1.0);

for(i = 2; i <= t; i++)

{

if(k%i==0)

{

result = result/i\*(i-1ll);

while(k%i==0)

k=k/i;

}

}

if(k>1ll)

result = result/k\*(k-1ll);

return result;

}

ll pow\_mod(ll m,ll c,ll k) {

ll b=1;

while(c>0){

if(c&1)

b=(b\*m)%k;

c=c>>1 ;

m=(m\*m)%k;

}

return b;

}

int main()

{

ll a;

getPrime();

//cout<<tot;

while(scanf("%I64d",&n)!=EOF){

if(n==0){

break;

}

ll ans=1;

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

if(prime[i]>n){

break;

}

a=getnum(prime[i]);

if((a&1)==1){

ans=(prime[i]\*ans)%MOD;

}

//cout<<ans<<' '<<prime[i]<<endl;

}

ans=(fac[n]\*pow\_mod(ans,MOD-2,MOD))%MOD;

printf("%I64d\n",ans);

}

return 0;

}

```

---

title: HDU4869 Turn the pokers

date: 2016-07-19

categories:

- 数学

- 数论

- 欧拉定理

tags:

- HDU

- 2014Muti 1

toc: true

---

[原题](http://acm.hdu.edu.cn/showproblem.php?pid=4869)

## 题意

给你一个长度为m的01序列，做n次翻转，每次翻转任意Xi个子序列，问最终结果有多少种情况。

## 分析

在n张牌选k张，很容易想到组合数，但是关键是怎么进行组合数计算呢？我们可以发现，在牌数固定的情况下，总共进行了sum次操作的话，其实有很多牌是经过了多次翻转，而每次翻转只有0和1两种状态，那么，奇偶性就出来了，也就是说，无论怎么进行翻牌，最终态无论有几个1，这些1的总数的奇偶性是固定的。

那么我们现在只需要找到最大的1的个数和最小的1的个数，然后再这个区间内进行组合数的求解即可

但是又有一个问题出来了，数据很大，进行除法是一个不明智的选择，但是组合数公式必定有除法

C（n,m） = n!/(m!\*(n-m)!)

费马小定理：a^(p-1)=1%p

那么a^(p-1)/a = 1/a%p 得到 a^(p-2) = 1/a%p

sum+=((f[m]%mod) \* (quickmod((f[i] \* f[m-i])%mod,mod-2)%mod))%mod

```

#include <cstdio>

#include <cstring>

#include <algorithm>

#define LL long long

using namespace std;

const int MAXN=200005;

const int MOD=1e9+9;

LL quick\_pow(LL a, LL b) {

LL ans = 1;

a %= MOD;//防止指数为0时，返回未取模的结果

while(b!=0) {

if((b & 1)==1) {

ans = ans \* a % MOD;

}

b >>= 1;

a = a \* a % MOD;

}

return ans;

}

LL fac[MAXN];

LL C(LL n,LL m) {

if(n<m||m<0||n==0) {

return 0;

}

LL s1=fac[n],s2=fac[n-m]\*fac[m]%MOD;

return s1\*quick\_pow(s2,MOD-2)%MOD;

}

void init() {

fac[0]=1;

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

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

}

}

int n,m,x,l,r,ll,rr;

long long ans;

int main() {

init();

while(2==scanf("%d%d",&n,&m)) {

l=r=0;

while(n-->0) {

scanf("%d",&x);

if(l>=x) {

ll=l-x;

}

else if(r>=x){

ll=(x-l)%2==0?0:1;

}

else {

ll=x-r;

}

if(r+x<=m) {

rr=r+x;

}

else if(l+x<=m) {

rr=(m-l-x)%2==0?m:m-1;

}

else {

rr=m-l+m-x;

}

l=ll;

r=rr;

}

ans=0;

while(l<=r) {

ans=(ans+C(m,l))%MOD;

l+=2;

}

printf("%I64d\n",ans);

}

return 0;

}

```

1. Miller-robin素数测试+RHO整数分解

大数+素数测试

#include <cstdio>

#include <cstring>

#include <cstdlib>

#define MAXL 4

#define M10 1000000000

#define Z10 9

const int zero[MAXL-1] = {0};

struct bnum {

int data[MAXL];

void read() {

memset( data, 0, sizeof( data ) );

char buf[32];

scanf( "%s", buf );

int len = strlen( buf );

int i = 0, k;

while( len >= Z10 ) {

for( k = len - Z10; k < len; ++k ) data[i] = data[i] \* 10 + buf[k] - '0';

++i; len -= Z10;

}

if( len > 0 ) {

for( k = 0; k < len; ++k ) data[i] = data[i] \* 10 + buf[k] - '0';

}

}

bool operator == ( const int x ) {

if( data[0] != x ) return false;

return memcmp( data + 1, zero, sizeof( zero ) ) == 0;

}

bool operator == ( const bnum & x ) {

return memcmp( data, x.data, sizeof( data ) ) == 0;

}

bnum & operator = ( const bnum & x ) {

memcpy( data, x.data, sizeof( data ) );

return \*this;

}

bnum & operator = ( const int x ) {

memset( data, 0, sizeof( data ) );

data[0] = x;

return \*this;

}

bnum operator + ( const bnum & x ) {

int i, carry = 0;

bnum ans;

for( i = 0; i < MAXL; ++i ) {

ans.data[i] = data[i] + x.data[i] + carry;

carry = ans.data[i] / M10;

ans.data[i] %= M10;

}

return ans;

}

bnum operator - ( const bnum & x ) {

int i, carry = 0;

bnum ans;

for( i = 0; i < MAXL; ++i ) {

ans.data[i] = data[i] - x.data[i] - carry;

if( ans.data[i] < 0 ) {

ans.data[i] += M10;

carry = 1;

}

else carry = 0;

}

return ans;

}

// assume \*this < x \* 2

bnum operator % ( const bnum & x ) {

int i;

for( i = MAXL - 1; i >= 0; --i ) {

if( data[i] < x.data[i] ) return \*this;

else if( data[i] > x.data[i] ) break;

}

return ( (\*this) - x );

}

bnum & div2() {

int i, carry = 0, tmp;

for( i = MAXL - 1; i >= 0; --i ) {

tmp = data[i] & 1;

data[i] = ( data[i] + carry ) >> 1;

carry = tmp \* M10;

}

return \*this;

}

bool is\_odd() { return ( data[0] & 1 ) == 1; }

bool is\_zero() { for( int i = 0; i < MAXL; ++i ) if( data[i] ) return false; return true; }

void print() {

int i;

for( i = MAXL - 1; i > 0; --i ) if( data[i] ) break;

for( printf( "%d", data[i--] ); i >= 0; --i ) printf( "%09d", data[i] );

putchar( '\n' );

}

};

inline

void mulmod( bnum & a0, bnum & b0, bnum & p, bnum & ans ) {

bnum tmp = a0, b = b0;

ans = 0;

while( ! b.is\_zero() ) {

if( b.is\_odd() ) ans = ( ans + tmp ) % p;

tmp = ( tmp + tmp ) % p;

b.div2();

}

}

inline

void powmod( bnum & a0, bnum & b0, bnum & p, bnum & ans ) {

bnum tmp = a0, b = b0;

ans = 1;

while( ! b.is\_zero() ) {

if( b.is\_odd() ) mulmod( ans, tmp, p, ans );

mulmod( tmp, tmp, p, tmp );

b.div2();

}

}

bool MillerRabinTest( bnum & p, int iter ) {

int i, small = 0, j, d = 0;

for( i = 1; i < MAXL; ++i ) if( p.data[i] ) break;

if( i == MAXL ) {

if( p.data[0] < 2 ) return false;

if( p.data[0] == 2 ) return true;

small = 1;

}

if( ! p.is\_odd() ) return false;

bnum a, s, m, one, pd1;

one = 1; s = pd1 = p - one;

while( ! s.is\_odd() ) { s.div2(); ++d; }

for( i = 0; i < iter; ++i ) {

a = rand();

if( small ) a.data[0] = a.data[0] % ( p.data[0] - 1 ) + 1;

else { a.data[1] = a.data[0] / M10; a.data[0] %= M10; }

if( a == one ) continue;

powmod( a, s, p, m );

for( j = 0; j < d && !( m == one ) && !( m == pd1 ); ++j ) {

mulmod( m, m, p, m );

}

if( !( m == pd1 ) && j > 0 ) return false;

}

return true;

}

int main() {

bnum x;

x.read();

puts( MillerRabinTest( x, 5 ) ? "Yes" : "No" );

return 0;

}

POJ1181要求判断给定数是否为质数，如果是合数则输出最小的质因子。

#include <cstdio>

#include <algorithm>

using namespace std;

typedef long long llt;

int const Repeat = 10;

//利用二进制计算a\*b%mod

llt multiMod(llt a,llt b,llt mod){

llt ret = 0LL;

a %= mod;

while( b ){

if ( b & 1LL ) ret = ( ret + a ) % mod, --b;

b >>= 1LL;

a = ( a + a ) % mod;

}

return ret;

}

//计算a^b%mod

llt powerMod(llt a,llt b,llt mod){

llt ret = 1LL;

a %= mod;

while( b ){

if ( b & 1LL ) ret = multiMod(ret,a,mod),--b;

b >>= 1LL;

a = multiMod(a,a,mod);

}

return ret;

}

//Miller-Rabin测试,测试n是否为素数

bool Miller\_Rabin(llt n,int repeat){

if ( 2LL == n || 3LL == n ) return true;

if ( !( n & 1LL ) ) return false;

//将n分解为2^s\*d

llt d = n - 1LL;

int s = 0;

while( !( d & 1LL ) ) ++s, d>>=1LL;

//srand((unsigned)time(0));

for(int i=0;i<repeat;++i){//重复repeat次

llt a = rand() % ( n - 3 ) + 2;//取一个随机数,[2,n-1)

llt x = powerMod(a,d,n);

llt y = 0LL;

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

y = multiMod(x,x,n);

if ( 1LL == y && 1LL != x && n-1LL != x ) return false;

x = y;

}

if ( 1LL != y ) return false;

}

return true;

}

llt Fac[100];//质因数分解结果（刚返回时是无序的）

int FCnt;//质因数的个数。数组小标从0开始

llt gcd(llt a,llt b){

if (0L == a || 0L == b) return 1;

if ( a < 0 ) a = -a;

if ( b < 0 ) b = -b;

while(b){

llt t = a % b;

a = b;

b = t;

}

return a;

}

llt Pollard\_Rho(llt n,llt c){

llt i = 1, k = 2;

llt x = rand() % n;

llt y = x;

while(1){

++i;

x = ( multiMod(x,x,n) + c ) % n;

llt d = gcd(y-x,n);

if (d!=1LL && d!=n) return d;

if (y == x) return n;

if ( i == k ) y = x, k <<= 1;

}

}

void find(llt n){

if ( 4LL == n ){

Fac[0] = Fac[1] = 2LL;

FCnt = 2;

return;

}

if ( Miller\_Rabin(n,Repeat) ){

Fac[FCnt++] = n;

return;

}

llt p;

while( ( p = Pollard\_Rho(n,rand()%(n-3)+3) ) == n );

find(p);

find(n/p);

}

int main(){

int kase;

scanf("%d",&kase);

while(kase--){

llt n;

scanf("%I64d",&n);

FCnt=0;

find(n);

if(Fac[0] == n) printf("Prime\n");

else printf("%I64d\n",\*min\_element(Fac,Fac+FCnt));

}

return 0;

}

1. BSGS离散对数&扩展BSGS（底数与模数不互质）
2. 原根

一个素数的原根

#include <bits/stdc++.h>

#define N 100007

#define ll long long

using namespace std;

ll prime[N],vis[N],tot;

void getprime(){

tot=0;

memset(vis,0,sizeof(vis));

vis[0]=vis[1]=1;

for(ll i=2;i<N;i++){

if(!vis[i]){

prime[tot++]=i;

}

for(ll j=0;j<tot;j++){

if(i\*prime[j]>N){//i\*prime[j]¿ÉÄÜ³¬ll

break;

}

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

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

break;

}

}

}

}

ll pow\_mod(ll a,ll b,ll mod){

ll ans=1;

while(b){

if(b&1){

ans=(ans\*a)%mod;

}

b>>=1;

a=(a\*a)%mod;

}

return ans;

}

int main()

{

int p,phi,tmp;

getprime();

scanf("%d",&p);

if(p==2){

printf("1\n");

return 0;

}

if(p==3){

printf("2\n");

return 0;

}

for(int i=2;i<=p-1;i++){

tmp=0;

for(int j=0;j<=tot&&prime[j]\*prime[j]<=p-1;j++){

if((p-1)%prime[j]==0){

int now=(p-1)/prime[j];

if(pow\_mod((long long)i,(long long)now,(long long)p)==1LL){

tmp=1;

break;

}

}

}

if(tmp==0){

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

break;

}

}

return 0;

}

```

1. 组合数学
2. 母函数

HDU 4651：将n拆分成多个正整数之和，有多少种拆法；

广义五边形数+分割函数：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<cmath>

#include<string>

#define inf 10000000000LL

#define MAX 100007

#define MOD 1000000007

#define ll long long

using namespace std;

ll dp[100007];

ll p(){//数拆分O(n√n)广义五边形数

memset(dp,0,sizeof(dp));

dp[1]=dp[0]=1;

dp[2]=2,dp[3]=3;

int now;

for(int i=4;i<MAX;i++){

now=1;

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

ll a=(j\*j\*3+j)/2;

ll b=(j\*j\*3-j)/2;

if(a>i&&b>i){

break;

}

if(a<=i){

dp[i]=(dp[i]+dp[i-a]\*now+MOD)%MOD;

}

if(b<=i){

dp[i]=(dp[i]+dp[i-b]\*now+MOD)%MOD;

}

now\*=-1;

}

}

return 0;

}

int main()

{

int t;

ll n,ans;

p();

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

while(t--){

scanf("%I64d",&n);

printf("%I64d\n",dp[n]%MOD);

}

}

}

HDU 4958：将n划分为最大值不超过n的若干数之和，每个数最多出现m-1次

ans=p[n]-p[n-m]-p[n-2m]+p[n-5m]+p[n-7m]-p[n-12m]…..

#include<iostream>

#include<cstdio>

#include<cstring>

#include<cmath>

#include<string>

#define inf 10000000000LL

#define MAX 100007

#define MOD 1000000007

#define ll long long

using namespace std;

ll dp[100007],ans,k,n;

ll p(){//数拆分O(n√n)广义五边形数

memset(dp,0,sizeof(dp));

dp[1]=dp[0]=1;

dp[2]=2,dp[3]=3;

int now;

for(int i=4;i<MAX;i++){

now=1;

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

ll a=(j\*j\*3-j)/2;

ll b=(j\*j\*3+j)/2;

if(a>i&&b>i){

break;

}

if(a<=i){

dp[i]=(dp[i]+dp[i-a]\*now+MOD)%MOD;//之后必须多加一个MOD将负数变为正数

}

if(b<=i){

dp[i]=(dp[i]+dp[i-b]\*now+MOD)%MOD;

}

now\*=-1;

}

}

return 0;

}

ll pp(){//数拆分O(n√n)广义五边形数

ll now=-1;

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

ll a=k\*((j\*j\*3-j)/2);

ll b=k\*((j\*j\*3+j)/2);

if(a>n&&b>n){

break;

}

if(a<=n){

ans=(ans+dp[n-a]\*now+MOD)%MOD;//之后必须多加一个MOD将负数变为正数

}

if(b<=n){

ans=(ans+dp[n-b]\*now+MOD)%MOD;

}

now\*=-1;

}

return 0;

}

int main()

{

int t;

p();

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

while(t--){

scanf("%I64d%I64d",&n,&k);

ans=dp[n];

pp();

printf("%I64d\n",ans%MOD);

}

}

}

HDU1171:普通的母函数

#include<iostream>

#include<cstdio>

#include<cstring>

#include<cmath>

#include<string>

#define inf 10000000000LL

#define MAX 100007

#define MOD 1000000007

#define ll long long

using namespace std;

int t,sum,p,now,ans;

int a[107],b[107],vis[250007];

void func(){

memset(vis,0,sizeof(vis));

p=0;

vis[0]=1;

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

for(int j=0;j<=b[i]\*a[i];j+=a[i]){

for(int k=0;k<=p;k++){

if(k+j>sum/2){

break;//是break不是return不然会少添加数。。

}

if(vis[k]==1){

vis[k+j]=1;

}

}

}

p+=a[i]\*b[i];

}

}

int main()

{

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

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

memset(b,0,sizeof(b));

if(t==-1){

break;

}

sum=0;

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

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

sum+=a[i]\*b[i];

}

func();

for(int i=sum/2;i>=0;i--){

if(vis[i]==1){

ans=i;

break;

}

}

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

}

}

3.群论

置换群

---

title: CF Round334div2 D

date: 2016-07-28

categories:

- 数学

- 组合数学

- 群论

- 置换群

tags:

- Code Forces

toc: true

---

[原题](http://codeforces.com/contest/603/problem/B)

## 题意

给定p和k，p是大于2的素数。找满足f(kx mod p)=k\*f(x) mod p的方案数，其中f(1,2,3,4......p-1)->{1,2,3,4......p-1}.

例如p=3,k=2的方案数有三种：

f(0) = 0, f(1) = 1, f(2) = 2.

f(0) = 0, f(1) = 2, f(2) = 1.

f(0) = f(1) = f(2) = 0.

## 分析

当p=3,k=2的时候：

2\*f(0)%3=f(0)=f(2\*0%3)

2\*f(1)%3=f(2)=f(2\*1%3)

2\*f(2)%3=f(1)=f(2\*2%3)

所以：

0->0

1->2->1

2->1->2

这题主要应用了群的性质，除去0的模p运算，是一个群

先不考虑0，让x和k都非0

我们先证明k \* x % p的值各不相同，且会包含全部的1~p-1

假设k \* x1 % p = k \* x2 % p ,且x1 != x2

在群中，所以每个元素有逆元，那么k^-1 \* k \* x1 %p = k^-1 \* k \* x2 %p

所以x1 = x2，由此得证。

利用不停迭代，可以得出下面的关系：

任取x1，有f(x1) = k\*f(x2)%p = k^f(k\*f(x3)) % p =k^2\*f(x2)%p= … = k^l \* f(x1)

x1 到 xl就凑成了一个循环的集合，确定x1，则其它元素的值都被确定了，所以对这样一个集合有f(x1)可以为0~p-1的任意值，所以有p种选择，所以找到所有这样的集合个数num，则ans = p ^ num

如何找到集合个数呢，我用的方法是用x1推xl，xl推xl-1，直到循环，xl = k \* x1 % p

现在把0加回来，对于k可以特判k=0时，有p^p-1种，k=1时，有p^p种。

当k>1时，对于x=0时，我们固定让f(x) = 1，其它值有上述算法即可。

```

#include <bits/stdc++.h>

using namespace std;

const long long MOD=1e9+7;

int main()

{

long long p,k,temp,ans,num;

while(scanf("%I64d%I64d",&p,&k)!=EOF){

if(k==0){

num=p-1;

}

else if(k==1){

num=p;

}

else{

int i;

int temp;

temp=k;

for(i=1;;i++){

if(temp==1){

break;

}

else{

temp=(temp\*k)%p;

}

}

num=(p-1)/i;

}

ans=1;

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

ans=(ans\*p)%MOD;

}

printf("%I64d\n",ans);

}

}

```-

4.容斥原理

---

title: HDU4497 GCD and LCM

date: 2016-05-01

categories:

- 数学

- 组合数学

- 计数原理

- 容斥原理

tags:

- HDU

- 2013 通化邀请赛

toc: true

---

[原题](http://acm.hdu.edu.cn/showproblem.php?pid=4497)

## 题意

已知三个数x,y,z的gcd和lcm，求有多少种可能性。

## 分析

\* 若lcm%gcd！=0，则不存在任何可能性。

\* 若lcm%gcd==0：

令d=lcm/gcd，对d进行唯一分解：

d = p1^a1 \* p2^a2 \* p3^a3 ……

若要满足这样的条件，x,y,z每个数首先要满足gcd的唯一分解。所以对x/g,y/g,z/g进行容斥：

设l/g中某一素数p的指数为cnt，则只有满足这三个数中至少有一个数的p的指数为cnt（否则最小公倍数不是l），至少有一个数的p的指数为0（否则最大公约数不是1）

共三种情况：

①一个数的p的指数为cnt，一个数的p的指数为0，另一个数的p的指数取1~cnt-1，方案数为：C(3,1)\*C(2,1)\*(cnt-1)

②两个数的p的指数为cnt，一个数的p的指数为0，方案数为：C(3,1)

③一个数的p的指数为cnt，两个个数的p的指数为0，方案数为：C(3,1)

\*\*注意：如果存在素数大于10^5，则必定只有一个，且其指数为1\*\*

```

#include <cstdio>

#include <cstring>

#define ll long long

using namespace std;

int vis[1000007],num;

int pri[1000007];

void prime(int n){

memset(vis,0,sizeof(vis));

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

if(vis[i]==0){

vis[i]=1;

pri[num++]=i;

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

vis[i\*j]=1;

}

}

}

}

int main()

{

int t;

ll ans,g,l;

num=0;

prime(1000000);

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

while(t--){

scanf("%I64d%I64d",&g,&l);

if(l%g!=0){

printf("0\n");

}

else{

ans=1;

ll d=l/g;

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

if(d%pri[i]==0){

ll n=0;

while(d%pri[i]==0){

n++;

d/=pri[i];

}

ans\*=6\*n;

}

}

if(d>1){

ans\*=6;

}

printf("%I64d\n",ans);

}

}

}

return 0;

}

```

1. 线性代数

1.FFT和NTT

51nod1028:大数乘法：FFT

递归：

#include <iostream>

#include <cstdio>

#include <cstring>

#include <complex>

using namespace std ;

const double PI = 3.1415926535897932384626 ;

const int MAXN = 262145 ;

const complex<double> I(0,1) ;

complex<double> a[MAXN] , a\_tmp[MAXN] ;

complex<double> save\_w0[2][MAXN] ;

complex<double> MUL( complex<double> z1 , complex<double> z2 ) {

double \_\_rr = z1.real() \* z2.real() ;

double \_\_ii = z1.imag() \* z2.imag() ;

double tmp = (z1.real() + z1.imag()) \* (z2.real() + z2.imag()) ;

complex<double> ret(\_\_rr - \_\_ii, tmp - \_\_rr - \_\_ii) ;

return ret;

}

// Consider the seq. a[x], a[x+s], ..., a[x+s\*(n-1)]

void FFT( int sign , int n , int x , int s ) {

if ( n == 1 ) return ;

if ( n == 2 ) {

a\_tmp[1] = a[x] ;

a\_tmp[2] = a[x+s] ;

a[x] = a\_tmp[1] + a\_tmp[2] ;

a[x+s] = a\_tmp[1] - a\_tmp[2] ;

return ;

}

FFT(sign , n/2 , x , s\*2) ;

FFT(sign , n/2 , x+s , s\*2) ;

int i0 , y0 ;

complex<double> w ;

for ( i0 = 0 , y0 = x , w = 1.0 ; i0 < n / 2 ; i0 ++ , y0 += s\*2 , w = MUL(w,save\_w0[sign][n]) ) {

a\_tmp[i0] = a[y0] ;

a\_tmp[i0+n/2] = MUL(a[y0+s] , w) ;

}

for ( i0 = 0 , y0 = x ; i0 < n/2 ; i0 ++ , y0 += s) {

a[y0] = a\_tmp[i0] + a\_tmp[i0+n/2] ;

a[y0+s\*n/2] = a\_tmp[i0] - a\_tmp[i0+n/2] ;

}

}

void Init() {//打表

for ( int i = 1 ; i <= 262144 ; i ++ ) {

save\_w0[1][i] = exp(2.0 \* PI \* I / (double)i) ;

save\_w0[0][i] = exp(-2.0 \* PI \* I / (double)i) ;

}

}

char s1[MAXN] , s2[MAXN] ;

int num1[MAXN] , num2[MAXN] ;

complex<double> b[MAXN] ;

int ans[MAXN] ;

int main() {

Init() ;

while(gets(s1)){

gets(s2) ;

int len1 = strlen(s1) ;

int len2 = strlen(s2) ;

int n = 1 ;

while ( len1 \* 2 > n || len2 \* 2 > n ) n \*= 2 ;

for ( int i = 1 ; i <= n ; i ++ ) num1[i] = 0 , num2[i] = 0 ;

for ( int i = len1 - 1 ; i >= 0 ; i -- ) num1[len1 - i] = (int)(s1[i] - '0') ;

for ( int i = len2 - 1 ; i >= 0 ; i -- ) num2[len2 - i] = (int)(s2[i] - '0') ;

for ( int i = 1 ; i <= n ; i ++ ) a[i] = (double)num1[i] ;

FFT(1 , n , 1 , 1) ;

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

for ( int i = 1 ; i <= n ; i ++ ) a[i] = (double)num2[i] ;

FFT(1 , n , 1 , 1) ;

for ( int i = 1 ; i <= n ; i ++ ) a[i] \*= b[i] ;

FFT(0 , n , 1 , 1) ;

for ( int i = 1 ; i <= n ; i ++ ) ans[i] = (int)(a[i]/(double)n+0.5).real() ;

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

ans[i+1] += ans[i] / 10 ;

ans[i] %= 10 ;

}

for ( int i = n ; i >= 1 ; i -- ) if ( i == 1 || ans[i] != 0 ) {

for ( int j = i ; j >= 1 ; j -- ) printf("%d" , ans[j]) ;

puts("") ;

break ;

}

}

}

1. 字符串
2. 后缀数组

---

title: HDU5769 Fast Matrix Calculation

date: 2016-07-28

categories:

- 字符串

- 后缀数组

tags:

- HDU

- 2016Muti 4

toc: true

---

[原题](http://acm.hdu.edu.cn/showproblem.php?pid=5769)

## 题意

给你一个字符和一个字符串，问这个字符串的所有子串中包含这个字符的字串有多少个。

## 分析

后缀数组处理单个字符串的不同字串个数，然后，发现没办法处理重复的问题，减去height[]数组本身就是为了去重，我再减去这个

后缀中最靠前的字符x的位置的话，肯定多减去了，前面不包含x的前缀重复的部分。

ans+=n-sa[i]-height[i]-(\*it-sa[i]);

所以，完全没辙试了很多种想法，跪完了比赛，比赛完，打开题解就发现，自己就是这个问题解决不了，看到答案一脸懵逼。

ans+=n-max(sa[i]+height[i],\*it);

因为假设对于当前这个后缀来说，最近的x位置大于需要被去重的部分的长度，那么所有包含x的子串都是唯一的，如果最近的x位置小于需要被去重的部分的长度，那么需要被去重的包含x的子串已经存在不需要再被记录，而后面包含x的子串则也是唯一的。

wa：我也是服了我自己了，想打long long一个顺手写了个double活生生wa了快半个小时。。MDZZ

```

#include <vector>

#include <iostream>

#include <string.h>

#include <algorithm>

#include <stdio.h>

using namespace std;

const int MAXN=100007;

int sa[MAXN];//SA数组，表示将S的n个后缀从小到大排序后把排好序的

//的后缀的开头位置顺次放入SA中

int t1[MAXN],t2[MAXN],c[MAXN];//求SA数组需要的中间变量，不需要赋值

int ran[MAXN],height[MAXN];

//待排序的字符串放在s数组中，从s[0]到s[n-1],长度为n,且最大值小于m,

//除s[n-1]外的所有s[i]都大于0，r[n-1]=0

//函数结束以后结果放在sa数组中

void build\_sa(int s[],int n,int m)

{

int i,j,p,\*x=t1,\*y=t2;

//第一轮基数排序，如果s的最大值很大，可改为快速排序

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

for(i=0;i<n;i++)c[x[i]=s[i]]++;

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

for(i=n-1;i>=0;i--)sa[--c[x[i]]]=i;

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

{

p=0;

//直接利用sa数组排序第二关键字

for(i=n-j;i<n;i++)y[p++]=i;//后面的j个数第二关键字为空的最小

for(i=0;i<n;i++)if(sa[i]>=j)y[p++]=sa[i]-j;

//这样数组y保存的就是按照第二关键字排序的结果

//基数排序第一关键字

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

for(i=0;i<n;i++)c[x[y[i]]]++;

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

for(i=n-1;i>=0;i--)sa[--c[x[y[i]]]]=y[i];

//根据sa和x数组计算新的x数组

swap(x,y);

p=1;x[sa[0]]=0;

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

x[sa[i]]=y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+j]==y[sa[i]+j]?p-1:p++;

if(p>=n)break;

m=p;//下次基数排序的最大值

}

}

void getHeight(int s[],int n)

{

int i,j,k=0;

for(i=0;i<=n;i++)ran[sa[i]]=i;

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

{

if(k)k--;

j=sa[ran[i]-1];

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

height[ran[i]]=k;

}

}

char str[MAXN],x[3];

int s[MAXN];

vector<int>locx;

vector<int>::iterator it;

int main()

{

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

int T;

scanf("%d",&T);

int kase=0;

while(T--)

{

locx.clear();

scanf("%s",x);

scanf("%s",str);

int n=strlen(str);

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

if(str[i]==x[0]){

locx.push\_back(i);

}

}

for(int i=0;i<=n;i++)s[i]=str[i];

build\_sa(s,n+1,128);

getHeight(s,n);

long long ans=0;

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

it=lower\_bound(locx.begin(),locx.end(),sa[i]);

if(it!=locx.end()) {

ans+=n-max(sa[i]+height[i],\*it);

}

}

printf("Case #%d: %I64d\n",++kase,ans);

}

return 0;

}

```

1. 计算几何

---

title: UVA 11178 Morley's Theorem

date: 2016-09-01

categories:

- 计算几何

- 向量

tags:

- UVA

toc: true

---

## 题意

给一个三角形，求每个角的两条三等分线相交的三个点的坐标。

## 分析

计算几何基础，已知三个点的坐标，由向量可以求出角度，并直接旋转三条边得到六条三等分线，直接两两求交点就行了。

```

#include <bits/stdc++.h>

#define MaX 500008

#define ll long long

using namespace std;

const double eps=1e-10;

struct Point{

double x,y;

Point(double x=0,double y=0):x(x),y(y){}

};

typedef Point Vector;

//向量加减

Vector operator + (Vector a,Vector b){

return Vector(a.x+b.x,a.y+b.y);

}

Vector operator - (Vector a,Vector b){

return Vector(a.x-b.x,a.y-b.y);

}

//向量乘除数值

Vector operator \* (Vector a,double b){

return Vector(a.x\*b,a.y\*b);

}

Vector operator / (Vector a,double b){

return Vector(a.x/b,a.y/b);

}

//eps控制，重载<和==

int dcmp(double x){

if(fabs(x)<eps){

return 0;

}

else{

return x<0?-1:1;

}

}

bool operator <(const Point& a,const Point& b){

return a.x-b.x<eps||(dcmp(a.x-b.x)==0&&a.y-b.y<eps);

}

bool operator == (const Point& a,const Point& b){

return dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;

}

//计算a的极角

double AngleX(Vector a){

return atan2(a.y,a.x);

}

//向量点乘

double Dot(Vector a,Vector b){

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

}

//向量a的模

double Length(Vector a){

return (sqrt(Dot(a,a)));

}

//两个向量的夹角

double Angle(Vector a,Vector b){

return acos(Dot(a,b)/Length(a)/Length(b));

}

//向量叉乘

double Cross(Vector a,Vector b){

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

}

//三点组成的面积

double Area2(Point a,Point b,Point c){

return Cross(b-a,c-a);

}

//将向量a旋转rad度，rad是弧度

Vector Rotate(Vector a,double rad){

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

}

//计算向量的单位法向量，先判a不是零向量

Vector Normal(Vector a){

double L=Length(a);

return Vector(-a.y/L,a.x/L);

}

//计算两个直线交点，直线方程为p+tv，q+tw，先判Cross(v,w)!=0

Point GetLineInter(Point p,Vector v,Point q,Vector w){

Vector u=p-q;

double t=Cross(w,u)/Cross(v,w);

return p+v\*t;

}

//点p到直线ab的距离

double DisToL(Point p,Point a,Point b){

Vector v1=b-a,v2=p-a;

return fabs(Cross(v1,v2))/Length(v1);//不取绝对值为有向距离

}

//点p到线段ab的距离

double DisToS(Point p,Point a,Point b){

if(a==b){

return Length(p-a);

}

Vector v1=b-a,v2=p-a,v3=p-b;

if(dcmp(Dot(v1,v2)<0)){

return Length(v2);

}

else if(dcmp(Dot(v1,v3))>0){

return Length(v3);

}

else{

return fabs(Cross(v1,v2))/Length(v1);

}

}

//点p在直线ab上的投影

Point GetLineProj(Point p,Point a,Point b){

Vector v=b-a;

return a+v\*(Dot(v,p-a)/Dot(v,v));

}

//线段ab,cd相交判定（交点不能为两个端点）

bool segProInter(Point a,Point b,Point c,Point d){

double p1=Cross(b-a,c-a),p2=Cross(b-a,d-a),

p3=Cross(d-c,a-c),p4=Cross(d-c,b-c);

return dcmp(p1)\*dcmp(p2)<0&&dcmp(p3)\*dcmp(p4)<0;

}

//如果相交允许为端点，直接判一下是否有端点在另一条直线上

bool OnSeg(Point p,Point a,Point b){

return dcmp(Cross(a-p,b-p))==0&&dcmp(Dot(a-p,b-p))<0;

}

int main()

{

int t;

Point a,b,c,d,f,e;

Vector bd,bf,cd,ce,ae,af;

scanf("%d",&t);

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

scanf("%lf%lf%lf%lf%lf%lf",&a.x,&a.y,&b.x,&b.y,&c.x,&c.y);

bd=Rotate((c-b),Angle(c-b,a-b)/3);

bf=Rotate((c-b),2\*(Angle(c-b,a-b)/3));

cd=Rotate((a-c),2\*(Angle(a-c,b-c)/3));

ce=Rotate((a-c),Angle(a-c,b-c)/3);

ae=Rotate((b-a),2\*(Angle(b-a,c-a)/3));

af=Rotate((b-a),Angle(b-a,c-a)/3);

d=GetLineInter(b,bd,c,cd);

e=GetLineInter(a,ae,c,ce);

f=GetLineInter(a,af,b,bf);

printf("%lf %lf %lf %lf %lf %lf\n",d.x,d.y,e.x,e.y,f.x,f.y);

}

}

```

Fish

## 最小树形图\_朱刘算法(poj 3164)

#include <climits>

#include <cstring>

#include <cstdio>

#include <cmath>

#define N 1005

#define type double

#define MAX INT\_MAX

#define MAXN 105

#define MAXM 1005

using namespace std;

struct Point

{

double x, y;

}p[MAXN];

struct Edge{

int u , v;

type cost;

}E[40005];

int pre[N],ID[N],vis[N];

type In[N];

int n,m;

double dis(Point a, Point b)

{

return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));

}

type zhuliu(int root,int NV,int NE) {//根、结点数、边数

type ret = 0;

while(true) {

//1.找最小入边

for(int i=0;i<NV;i++) In[i] = MAX;

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

int u = E[i].u;

int v = E[i].v;

if(E[i].cost < In[v] && u != v) { //这一步可以把自环切掉，找出每个点的最小入边

pre[v] = u;

In[v] = E[i].cost;

}

}

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

if(i == root) continue;

if(In[i] == MAX) return -1;//除了跟以外有点没有入边,则根无法到达它

}

//2.找环

int cntnode = 0;

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

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

In[root] = 0;

for(int i=0;i<NV;i++) {//标记每个环

ret += In[i];

int v = i;

while(vis[v] != i && ID[v] == -1 && v != root) {

vis[v] = i; //vis的作用就是把环中各节点都标记为有入边的那个点，如下图，把1,2,3,4,5,6都标记为1

v = pre[v];

}

if(v != root && ID[v] == -1) {

for(int u = pre[v] ; u != v ; u = pre[u]) {

ID[u] = cntnode;

}

ID[v] = cntnode ++;

}

}

if(cntnode == 0) break;//无环

for(int i=0;i<NV;i++) if(ID[i] == -1) {

ID[i] = cntnode ++;

}

//3.缩点,重新标记，调整进入环的边的权值，以便下一轮循环找出最小入边

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

int v = E[i].v;

E[i].u = ID[E[i].u];

E[i].v = ID[E[i].v];

if(E[i].u != E[i].v) {

E[i].cost -= In[v];

}

}

NV = cntnode;

root = ID[root];

}

return ret;

}

int main()

{

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

{

for(int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);

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

{

scanf("%d%d", &E[i].u, &E[i].v);

E[i].u--;

E[i].v--;

if(E[i].u != E[i].v) E[i].cost = dis(p[E[i].u], p[E[i].v]);

else E[i].cost = MAX; //去除自环

}

type ans = zhuliu(0, n, m);

if(ans == -1) printf("poor snoopy\n");

else printf("%.2f\n", ans);

}

return 0;

}

## 斯坦纳树

题目:

斯坦纳树   
Time Limit: 1 Sec Memory Limit: 128 MB   
Description   
现在有一个n\*m的矩阵,某些元素为0,剩下的元素大于0.   
现在你要选择一些元素,使得任意两个为0的元素都能够通过选中的元素四连通.   
(注意,若想达到要求,所有的0自身必须被选中.)   
那么请问选中元素的和的最小值是多少?

Input   
第一行两个整数n,m,表示矩阵的长和宽.   
接下来n行,每行m个整数,第i行第j个整数A[i,j]表示矩阵中第i行第j列的元素大小.

Output   
输出一行一个整数表示选中元素的最小和.

Sample Input   
4 4   
0 1 1 0   
1 9 9 1   
1 9 9 1   
0 1 1 0   
Sample Output   
6

HINT   
1<=n,m<=10,2<=num(0)<=10,0<=A[i,j]<=2^16.

题解：

斯坦纳树模板   
流程：

枚举状态集S   
{   
     枚举S的子集s   
     {   
         更新f[S][1~n]   
     }   
     将 f[S][x]<inf 的x入队   
     spfa(S)   
}

代码：

#include <queue>

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#define N 105

#define P 12

#define inf 0x3f3f3f3f

using namespace std;

const int dx[]={0,0,1,-1};

const int dy[]={1,-1,0,0};

struct Eli

{

int v,n;

}e[N<<2];

int head[N],cnt;

inline void add(int u,int v)

{

e[++cnt].v=v;

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

head[u]=cnt;

}

int n,m;

int id[P][P],val[N];

void build()

{

int i,j,k;

int x,y;

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

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

scanf("%d",&val[id[i][j]=++cnt]);

cnt=0;

for(i=1;i<=n;i++)for(j=1;j<=m;j++)for(k=0;k<4;k++)

if(id[x=i+dx[k]][y=j+dy[k]])add(id[i][j],id[x][y]);

n\*=m,cnt=0;

}

int f[1<<P][N];

queue<int>q;

bool in[N];

void spfa(int S)

{

int i,u,v;

while(!q.empty())

{

u=q.front(),q.pop(),in[u]=0;

for(i=head[u];i;i=e[i].n)

{

if(f[S][v=e[i].v]>f[S][u]+val[v])

{

f[S][v]=f[S][u]+val[v];

if(!in[v])q.push(v);

}

}

}

}

void work()

{

int i,j,k;

int S,s;

memset(f,0x3f3f3f3f,sizeof f);

for(i=1;i<=n;i++)if(!val[i])f[1<<cnt++][i]=0;

for(S=1;S<(1<<cnt);S++)

{

for(s=(S-1)&S;s;s=(s-1)&S)for(i=1;i<=n;i++)

f[S][i]=min(f[S][i],f[s][i]+f[S^s][i]-val[i]);

for(i=1;i<=n;i++)if(f[S][i]<inf&&!in[i])q.push(i),in[i]=1;

spfa(S);

}

int ans=inf;

for(i=1;i<=n;i++)ans=min(ans,f[(1<<cnt)-1][i]);

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

}

int main()

{

freopen("test.in","r",stdin);

build();

work();

return 0;

}

## 最小乘积生成树(bzoj 2359)

题解：

裸最小乘积生成树。

最小乘积生成树定义：

有一张n个点m条边的无向图，每条边有k个权值。

现在要取一个边集M使得其将所有点连通，并使

∏ki=1(∑j∈Mjcost(j,vali)) 最小

即个边集的每一种边权的总和的乘积最小。

比如：

k=1时，就是裸最小生成树。

k=2时，就是要使 [边集的权值1的和]\*[边集的权值2的和] 最小。

最小乘积生成树的一种求法：

广义上的说法（没必要看，或者看完下面的再来看这个就好）

首先我们可以把每种生成树想成一个k维的点，第i维的坐标即那一维上权值的和。

然后我们可以先求出每一维坐标最小的一棵生成树（裸上最小生成树就好），

然后得到一个k-1维的面，然后我们来求一下离这个面最远的点，然后分治下去……据说期望很快……

二维最小乘积生成树的求法：

给每一棵生成树都定义两个权值X、Y，其中X为其包含的所有边的权值x的和，Y为其包含的所有边的权值y的和，那么我们可以把每一种生成树看成一个坐标。

我们先求出坐标x最小的一棵生成树，再求出坐标y最小的一棵生成树。

然后我们可以考虑，最优的点一定在下凸包上【证明一】，然后我们要进行一个不断向左下拓展点的过程：对于两个点A、B形成的直线，我们可以找出在这条直线左下的最远的点C，然后对AC、CB递归做同样的过程，直到找不到一个在左下的点C为止。

然后如何找一个最远的点C呢？

我们可以发现既然有一条边固定，那么不妨把“最远”转化成三角形面积最大，这个用叉积搞一搞，，然后会推出公式面积跟点C有关的部分= 常数A∗x+常数B∗y ，，那么我们把所有边的权值 x 乘上 A ，权值 y 乘上 B 就好了，，，A、B 是啥自己求去！

【题外】：三维的，就是离一个平面最远，转化成体积最大，递归分成三层而非两层，，，，，然后四维，甚至更高维，感觉应该是同理的吧？

关于上文中【证明一】

每个点（xi,yi） 都对应一条函数曲线 ki=xi∗yi，而任意两不同 ki ，它们的函数曲线是不交的（有交的话则存在一点 (xj,yj) 使得 ki=xj∗yj=kj而ki!=kj 成立，显然这是悖论），那么显然最优点肯定不会在凸包内，否则必有凸包上一点比它优。

那么会不会求出这个某种意义上的凸包后，最优点在凸包外，却没被找到呢？

不会。

若有这种情况，此点必然在凸包上某边的左下方，，然后一定会被找出来。。

代码：

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#define N 205

#define M 10100

#define inf 0x3f3f3f3f

using namespace std;

struct Eli

{

int u,v,a,b,c;

void read()

{

scanf("%d%d%d%d",&u,&v,&a,&b);

u++,v++;

}

}e[M];

inline bool cmpa(const Eli &a,const Eli &b){return a.a<b.a;}

inline bool cmpb(const Eli &a,const Eli &b){return a.b<b.b;}

inline bool cmpc(const Eli &a,const Eli &b){return a.c<b.c;}

struct Point

{

int x,y;

void print(){printf("%d %d\n",x,y);}

Point(int \_x=0,int \_y=0):x(\_x),y(\_y){}

bool operator < (const Point &A)const

{

unsigned int p= x;p\*= y;

unsigned int q=A.x;q\*=A.y;

return p==q?x<A.x:p<q;

}

}ans,now,mina,minb;

int f[N],n,m;

int find(int x){return f[x]==x?x:f[x]=find(f[x]);}

Point Kruscal()

{

int i,fa,fb;

now=Point(0,0);

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

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

{

fa=find(e[i].u),fb=find(e[i].v);

if(fa!=fb)

{

f[fb]=fa;

now.x+=e[i].a;

now.y+=e[i].b;

}

}

if(now<ans)ans=now;

return now;

}

inline int xmul(const Point &A,const Point &B,const Point &C)

{return (C.y-A.y)\*(B.x-A.x)-(C.x-A.x)\*(B.y-A.y);}

void work(const Point &A,const Point &B)

{

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

e[i].c=e[i].b\*(A.x-B.x)+e[i].a\*(B.y-A.y);

sort(e+1,e+m+1,cmpc);

Point C=Kruscal();

if(xmul(A,B,C)<=0)return ;

work(A,C),work(C,B);

}

int main()

{

// freopen("test.in","r",stdin);

int i,j,k;

int a,b,c;

ans=Point(inf,inf);

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

for(i=1;i<=m;i++)e[i].read();

sort(e+1,e+m+1,cmpa),mina=Kruscal();

sort(e+1,e+m+1,cmpb),minb=Kruscal();

work(minb,mina),ans.print();

fclose(stdin);

fclose(stdout);

return 0;

}

## fleury算法

#include <iostream>

#include <stdio.h>

#include <string.h>

#define MAXN 200

using namespace std;

struct stack{

int top,node[MAXN];

}s;

int Edge[MAXN][MAXN];

int n;

void dfs(int x){

int i;

s.top++;

s.node[s.top]=x;

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

if(Edge[i][x]>0){

Edge[i][x]=0;

Edge[x][i]=0;

dfs(i);

break;

}

}

}

void Fleury(int x){

int i,b;

s.top=0;s.node[s.top]=x;

while(s.top>=0){

b=0;

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

if(Edge[s.node[s.top]][i]>0){

b=1;break;

}

}

if(b==0){

printf("%d ",s.node[s.top]+1);

s.top--;

}

else{

s.top--;

dfs(s.node[s.top+1]);

}

}

printf("\n");

}

int main()

{

//freopen("in.txt","r",stdin);

int i,j;

int m,s,t;

int degree,num,start;

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

memset(Edge,0,sizeof(Edge));

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

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

Edge[s-1][t-1]=1;

Edge[t-1][s-1]=1;

}

num=0;start=0;

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

degree=0;

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

degree+=Edge[i][j];

if(degree%2==1){

start=i;

num++;

}

}

if(num==0 || num==2) Fleury(start);

else printf("No Euler path\n");

}

return 0;

}

## ISAP最大流

#include <stdio.h>

#include <string.h>

#include <algorithm>

#define clear(A, X) memset (A, X, sizeof A)

#define copy(A, B) memcpy (A, B, sizeof A)

using namespace std;

const int maxE = 1000000;

const int maxN = 100000;

const int maxQ = 1000000;

const int oo = 0x3f3f3f3f;

struct Edge {

int v;//弧尾

int c;//容量

int n;//指向下一条从同一个弧头出发的弧

} edge[maxE];//边组

int adj[maxN], cntE;//前向星的表头

int Q[maxQ], head, tail;//队列

int d[maxN], cur[maxN], pre[maxN], num[maxN];

int sourse, sink, nv;//sourse：源点，sink：汇点，nv：编号修改的上限

int n, m;

void addedge (int u, int v, int c) {//添加边

//正向边

edge[cntE].v = v;

edge[cntE].c = c;//正向弧的容量为c

edge[cntE].n = adj[u];

adj[u] = cntE++;

//反向边

edge[cntE].v = u;

edge[cntE].c = 0;//反向弧的容量为0

edge[cntE].n = adj[v];

adj[v] = cntE++;

}

void rev\_bfs () {//反向BFS标号

clear (num, 0);

clear (d, -1);//没标过号则为-1

d[sink] = 0;//汇点默认为标过号

num[0] = 1;

head = tail = 0;

Q[tail++] = sink;

while (head != tail) {

int u = Q[head++];

for (int i = adj[u]; ~i; i = edge[i].n) {

int v = edge[i].v;

if (~d[v]) continue;//已经标过号

d[v] = d[u] + 1;//标号

Q[tail++] = v;

num[d[v]]++;

}

}

}

int ISAP() {

copy (cur, adj);//复制，当前弧优化

rev\_bfs ();//只用标号一次就够了，重标号在ISAP主函数中进行就行了

int flow = 0, u = pre[sourse] = sourse, i;

while (d[sink] < nv) {//最长也就是一条链，其中最大的标号只会是nv - 1，如果大于等于nv了说明中间已经断层了。

if (u == sink) {//如果已经找到了一条增广路，则沿着增广路修改流量

int f = oo, neck;

for (i = sourse; i != sink; i = edge[cur[i]].v) {

if (f > edge[cur[i]].c){

f = edge[cur[i]].c;//不断更新需要减少的流量

neck = i;//记录回退点，目的是为了不用再回到起点重新找

}

}

for (i = sourse; i != sink; i = edge[cur[i]].v) {//修改流量

edge[cur[i]].c -= f;

edge[cur[i] ^ 1].c += f;

}

flow += f;//更新

u = neck;//回退

}

for (i = cur[u]; ~i; i = edge[i].n) if (d[edge[i].v] + 1 == d[u] && edge[i].c) break;

if (~i) {//如果存在可行增广路，更新

cur[u] = i;//修改当前弧

pre[edge[i].v] = u;

u = edge[i].v;

}

else {//否则回退，重新找增广路

if (0 == (--num[d[u]])) break;//GAP间隙优化，如果出现断层，可以知道一定不会再有增广路了

int mind = nv;

for (i = adj[u]; ~i; i = edge[i].n) {

if (edge[i].c && mind > d[edge[i].v]) {//寻找可以增广的最小标号

cur[u] = i;//修改当前弧

mind = d[edge[i].v];

}

}

d[u] = mind + 1;

num[d[u]]++;

u = pre[u];//回退

}

}

return flow;

}

void init () {//初始化

clear (adj, -1);

cntE = 0;

}

void work () {

int u, v, c;

init ();

for (int i = 0; i < m; ++ i) scanf ("%d%d%d", &u, &v, &c), addedge (u, v, c);

sourse = 1; sink = n; nv = sink + 1;

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

}

int main() {

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

return 0;

}

## Dinic费用流(poj2112)

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <algorithm>

#include <cstring>

using namespace std;

const int MAXPT = 255;

const int MAXED = 100005;

const int INF = 0x3fffffff;

struct node{

int u, v, flow;

int opp;

int next;

};

struct Dinic{

node arc[MAXED];

int vn, en, head[MAXPT]; //vn点个数(包括源点汇点),en边个数

int cur[MAXPT]; //当前弧

int q[MAXPT]; //bfs建层次图时的队列

int path[MAXED], top; //存dfs当前最短路径的栈

int dep[MAXPT]; //各节点层次

void init(int n){

vn = n;

en = 0;

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

}

void insert\_flow(int u, int v, int flow){

arc[en].u = u;

arc[en].v = v;

arc[en].flow = flow;

arc[en].opp = en + 1;

arc[en].next = head[u];

head[u] = en ++;

arc[en].u = v;

arc[en].v = u;

arc[en].flow = 0; //反向弧

arc[en].opp = en - 1;

arc[en].next = head[v];

head[v] = en ++;

}

bool bfs(int s, int t){

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

int lq = 0, rq = 1;

dep[s] = 0;

q[lq] = s;

while(lq < rq){

int u = q[lq ++];

if (u == t){

return true;

}

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

int v = arc[i].v;

if (dep[v] == -1 && arc[i].flow > 0){

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

q[rq ++] = v;

}

}

}

return false;

}

int solve(int s, int t){

int maxflow = 0;

while(bfs(s, t)){

int i,j,k;

for (i = 1; i <= vn; i ++) cur[i] = head[i];

for (i = s, top = 0;;){

if (i == t){

int mink;

int minflow = 0x3fffffff;

for (k = 0; k < top; k ++)

if (minflow > arc[path[k]].flow){

minflow = arc[path[k]].flow;

mink = k;

}

for (k = 0; k < top; k ++)

arc[path[k]].flow -= minflow, arc[arc[path[k]].opp].flow += minflow;

maxflow += minflow;

top = mink; //arc[mink]这条边流量变为0, 则直接回溯到该边的起点即可(这条边将不再包含在增广路内).

i = arc[path[top]].u;

}

for (j = cur[i]; j != -1; cur[i] = j = arc[j].next){

int v = arc[j].v;

if (arc[j].flow && dep[v] == dep[i] + 1)

break;

}

if (j != -1){

path[top ++] = j;

i = arc[j].v;

}

else{

if (top == 0) break;

dep[i] = -1;

i = arc[path[-- top]].u;

}

}

}

return maxflow;

}

}dinic;

int map[250][250];

void floyd(int n){

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

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

if(map[i][k] == INF) continue;

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

if (map[j][k] == INF) continue;

if (map[i][j] > map[i][k] + map[k][j])

map[i][j] = map[i][k] + map[k][j];

}

}

}

return ;

}

int go(int mid, int k, int c, int m)

{

int i;

dinic.init(k+c+2);

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

dinic.insert\_flow(k+c+1, i, m);

}

for (i = k+1; i <= k+c; i ++){

dinic.insert\_flow(i, k+c+2, 1);

}

for (i = 0; i < k; i ++){

for (int j = k; j < k+c; j ++){

if (map[i][j] <= mid){

dinic.insert\_flow(i+1, j+1, 1);

}

}

}

return dinic.solve(k+c+1, k+c+2);

}

int BS(int k, int c, int m){

int l = 0, r = 10000000;

while(l < r){

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

if (go(mid, k, c, m) == c){

r = mid;

}

else{

l = mid + 1;

}

}

return r;

}

int main ()

{

int k, c, m;

scanf("%d %d %d", &k, &c, &m);

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

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

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

if (map[i][j] == 0)

map[i][j] = INF;

}

}

floyd(k+c);

printf("%d\n", BS(k, c, m));

return 0;

}

## KM算法

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

const int MAXN = 305;

const int INF = 0x3f3f3f3f;

int love[MAXN][MAXN]; // 记录每个妹子和每个男生的好感度

int ex\_girl[MAXN]; // 每个妹子的期望值

int ex\_boy[MAXN]; // 每个男生的期望值

bool vis\_girl[MAXN]; // 记录每一轮匹配匹配过的女生

bool vis\_boy[MAXN]; // 记录每一轮匹配匹配过的男生

int match[MAXN]; // 记录每个男生匹配到的妹子 如果没有则为-1

int slack[MAXN]; // 记录每个汉子如果能被妹子倾心最少还需要多少期望值

int N;

bool dfs(int girl)

{

vis\_girl[girl] = true;

for (int boy = 0; boy < N; ++boy) {

if (vis\_boy[boy]) continue; // 每一轮匹配 每个男生只尝试一次

int gap = ex\_girl[girl] + ex\_boy[boy] - love[girl][boy];

if (gap == 0) { // 如果符合要求

vis\_boy[boy] = true;

if (match[boy] == -1 || dfs( match[boy] )) { // 找到一个没有匹配的男生 或者该男生的妹子可以找到其他人

match[boy] = girl;

return true;

}

} else {

slack[boy] = min(slack[boy], gap); // slack 可以理解为该男生要得到女生的倾心 还需多少期望值 取最小值 备胎的样子【捂脸

}

}

return false;

}

int KM()

{

memset(match, -1, sizeof match); // 初始每个男生都没有匹配的女生

memset(ex\_boy, 0, sizeof ex\_boy); // 初始每个男生的期望值为0

// 每个女生的初始期望值是与她相连的男生最大的好感度

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

ex\_girl[i] = love[i][0];

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

ex\_girl[i] = max(ex\_girl[i], love[i][j]);

}

}

// 尝试为每一个女生解决归宿问题

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

fill(slack, slack + N, INF); // 因为要取最小值 初始化为无穷大

while (1) {

// 为每个女生解决归宿问题的方法是 ：如果找不到就降低期望值，直到找到为止

// 记录每轮匹配中男生女生是否被尝试匹配过

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

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

if (dfs(i)) break; // 找到归宿 退出

// 如果不能找到 就降低期望值

// 最小可降低的期望值

int d = INF;

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

if (!vis\_boy[j]) d = min(d, slack[j]);

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

// 所有访问过的女生降低期望值

if (vis\_girl[j]) ex\_girl[j] -= d;

// 所有访问过的男生增加期望值

if (vis\_boy[j]) ex\_boy[j] += d;

// 没有访问过的boy 因为girl们的期望值降低，距离得到女生倾心又进了一步！

else slack[j] -= d;

}

}

}

// 匹配完成 求出所有配对的好感度的和

int res = 0;

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

res += love[ match[i] ][i];

return res;

}

int main()

{

while (~scanf("%d", &N)) {

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

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

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

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

}

return 0;

}

## Tarjan(求桥割点，点从1-n)

#include<iostream>

using namespace std;

#include<cstdio>

#include<cstring>

#include<vector>

#define N 201

vector<int>G[N];

int n,m,low[N],dfn[N];

bool is\_cut[N];

int father[N];

int tim=0;

void input()

{

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

int a,b;

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

{

scanf("%d%d",&a,&b);

G[a].push\_back(b);/\*邻接表储存无向边\*/

G[b].push\_back(a);

}

}

void Tarjan(int i,int Father)

{

father[i]=Father;/\*记录每一个点的父亲\*/

dfn[i]=low[i]=tim++;

for(int j=0;j<G[i].size();++j)

{

int k=G[i][j];

if(dfn[k]==-1)

{

Tarjan(k,i);

low[i]=min(low[i],low[k]);

}

else if(Father!=k)/\*假如k是i的父亲的话，那么这就是无向边中的重边，有重边那么一定不是桥\*/

low[i]=min(low[i],low[k]);

}

}

void count()

{

int rootson=0;

Tarjan(1,0);

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

{

int v=father[i];

if(v==1)

rootson++;/\*统计根节点子树的个数，根节点的子树个数>=2,就是割点\*/

else{

if(low[i]>=dfn[v])/\*割点的条件\*/

is\_cut[v]=true;

}

}

if(rootson>1)

is\_cut[1]=true;

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

if(is\_cut[i])

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

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

{

int v=father[i];

if(v>0&&low[i]>dfn[v])/\*桥的条件\*/

printf("%d,%d\n",v,i);

}

}

int main()

{

input();

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

memset(father,0,sizeof(father));

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

memset(is\_cut,false,sizeof(is\_cut));

count();

return 0;

}

## Tarjan(SCC)

#include <queue>

#include <stack>

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

const int N = 2e3+5;

const int M = 1e4+5;

struct Vertex

{

int head;

}V[N];

struct Edge

{

int v,next;

}E[M];

int top,scc,id,dfn[N],low[N],belong[N];

bool in[N];

stack<int> S;

void init()

{

top = 0;

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

}

void add\_edge(int u,int v)

{

E[top].v = v;

E[top].next = V[u].head;

V[u].head = top++;

}

void tarjan(int u)

{

dfn[u] = low[u] = ++id;

S.push(u);

in[u] = true;

for(int i=V[u].head;~i;i=E[i].next)

{

int v = E[i].v;

if(!dfn[v])

{

tarjan(v);

low[u] = min(low[u],low[v]);

}

else if(in[v])

low[u] = min(low[u],dfn[v]);

}

if(dfn[u] == low[u])

{

int v;

do

{

v = S.top();

S.pop();

in[v] = false;

belong[v] = scc;

}while(u != v);

scc++;

}

}

void get\_scc(int n)

{

scc = id = 0;

memset(dfn,0,sizeof(dfn));

memset(in,false,sizeof(in));

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

if(!dfn[i])

tarjan(i);

}

## Tarjan(BCC)

#include <cstdio>

#include <cstring>

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int MAXNODE = 10005;

const int MAXEDGE = 100010;

const int INF = 0x3f3f3f3f;

struct Edge{

int v, id, next;

bool bridge;

Edge() {}

Edge(int v, int id, int next): v(v), id(id), next(next), bridge(false){}

}E[MAXEDGE \* 2], cut[MAXEDGE];

struct Node {

int u, v;

Node() {}

Node(int u, int v): u(u), v(v) {}

};

stack<Node> Stack;

//bcc表示的是一个bcc里面的点

vector<int> bcc[MAXNODE];

//pre纪录的是时间戳,lowlink纪录的是该点及其该子孙节点所能返回的最早时间戳是多少,bccno纪录的是该点当前是属于哪个bcc的

int head[MAXNODE], pre[MAXNODE], lowlink[MAXNODE], bccno[MAXNODE];

int n, m, tot, bcc\_cnt, dfs\_clock, cut\_cnt;

bool iscut[MAXNODE];

//双向边的添加

void AddEdge(int u, int v, int id) {

E[tot] = Edge(v, id, head[u]);

head[u] = tot++;

E[tot] = Edge(u, id, head[v]);

head[v] = tot++;

}

void init() {

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

tot = 0;

}

//点双连通

void dfs(int u, int fa) {

pre[u] = lowlink[u] = ++dfs\_clock;

int child = 0;//纪录当前节点有多少个子节点

for (int i = head[u]; ~i; i = E[i].next) {

int v = E[i].v;

if (!pre[v]) {

Stack.push(Node(u, v));

child++;

dfs(v, u);

lowlink[u] = min(lowlink[u], lowlink[v]);//更新

//子节点最多返回到该点

if (lowlink[v] >= pre[u]) {

//该边为桥

if (lowlink[v] > pre[u]) {

E[i].bridge = E[i ^ 1].bridge = true;

cut[cut\_cnt++] = E[i];

}

iscut[u] = true;

bcc\_cnt++; bcc[bcc\_cnt].clear();

while (1) {

Node x = Stack.top(); Stack.pop();

if (bccno[x.u] != bcc\_cnt) {

bcc[bcc\_cnt].push\_back(x.u);

bccno[x.u] = bcc\_cnt;

}

if (bccno[x.v] != bcc\_cnt) {

bcc[bcc\_cnt].push\_back(x.v);

bccno[x.v] = bcc\_cnt;

}

if (x.u == u && x.v == v) break;

}

}

}

else if (v != fa && pre[v] < pre[u]) {//反向边

Stack.push(Node(u, v));

lowlink[u] = min(lowlink[u], pre[v]);

}

}

//u是根结点，且只有一个孩子，那就不是割点了

if (fa < 0 && child == 1) iscut[u] = 0;

}

void find\_bcc() {

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

memset(bccno, 0, sizeof(bccno));

dfs\_clock = bcc\_cnt = cut\_cnt = 0;

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

if (!pre[i]) dfs(i, -1);

}

int main() {

init();

return 0;

}

## 2-sat算法(hdu 3062)

#include<stdio.h>

#include<string.h>

const int maxn = 3000;

struct node

{

int v,next;

}eg[maxn\*maxn];

int head[maxn];

int dfn[maxn],low[maxn],sta[maxn],insta[maxn],belong[maxn];

int tot,color,top,Index,n;

void add(int a,int b)

{

eg[tot].v= b;

eg[tot].next = head[a];

head[a] = tot++;

}

void tarjan(int u)

{

dfn[u] = low[u] = ++Index;

sta[top++] = u;

insta[u] = 1;

for(int i=head[u];i+1;i= eg[i].next)

{

int v= eg[i].v;

if(!dfn[v])

{

tarjan(v);

low[u] = low[u] < low[v] ? low[u] : low[v];

}else if(insta[v])

low[u] = low[u] < dfn[v] ? low[u] : dfn[v];

}

if(low[u] == dfn[u])

{

int v;

color++;

do

{

v= sta[--top];

insta[v] = 0;

belong[v] =color;

}while(v != u);

}

}

void init()

{

tot = color = top = Index =0;

memset(dfn,0,sizeof(dfn));

memset(insta,0,sizeof(insta));

memset(belong,0,sizeof(belong));

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

}

void work()

{

int i;

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

{

if(!dfn[i])tarjan(i);

}

int flag = 0;

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

{

if(belong[i\*2] == belong[i\*2+1])

{

break;

}

}

if(i == n) printf("YES\n");

else printf("NO\n");

}

int main()

{

int m,a1,a2,b1,b2;

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

{

init();

scanf("%d",&m);

while(m--)

{

scanf("%d%d%d%d",&a1,&a2,&b1,&b2);

// printf("%d %d\n",a1\*2+a2,b1\*2 + !b2);

// printf("%d %d\n",b1\*2 + b2,a1\*2+!a2);

add(a1\*2 + b1,a2\*2 + 1 -b2);

add(a2\*2 + b2,a1\*2 + 1 -b1);

}

work();

}

return 0;

}

## 最大团(hdu 1530)

#include<iostream>

#include<cstring>

#include<cstdio>

using namespace std;

int n;

int map[55][55],cn,maxt,mark[500];

void dfs(int x)

{

if(x>n)

{

maxt=cn;

return;

}

bool flag=true;

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

{

if(mark[i]&&!map[x][i])

{

flag=false;

break;

}

}

if(flag)

{

cn++;

mark[x]=true;

dfs(x+1);

cn--;

}

if(cn+n-x+1>maxt)

{

mark[x]=false;

dfs(x+1);

}

}

int main()

{

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

{

memset(map,0,sizeof(map));

memset(mark,0,sizeof(mark));

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

{

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

{

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

}

}

cn=maxt=0;

dfs(1);

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

}

return 0;

}