1. KMP 2
2. 拓扑排序 3
3. 最长回文子串 5
4. 快速幂求逆元 6
5. 扩展欧几里得 6
6. O（n）求逆元 7
7. 孙子定理 7
8. 同余定理 8
9. 矩阵快速幂 8
10. 卢卡斯定理：大组合数 9
11. 线段树 9
12. AC自动机 13
13. 弱连通分量判断 18
14. 强连通分量判断 20
15. 字典树 23
16. 价值背包 27
17. 多重背包 28

char p[10000],b[10000]; //kmp

int next[10000];

int num=0;

void makenext()

{

int i,j,k=0;

int m=strlen(b);

next[0]=0;

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

{

while(k>0&&b[i]!=b[k])

k=next[k-1];

if(b[i]==b[k])

k++;

next[i]=k;

}

}

void kmp()

{

int i,j,k;

int pp=strlen(p),bb=strlen(b);

for(i=0,j=0;i<pp;i++)

{

if(j>0&&p[i]!=b[j])

j=next[j-1];

if(p[i]==b[j])

j++;

if(j==bb)

{

j=0;

num++;

}

}

}

int main()

{

cin>>b>>p;

makenext();

kmp();

cout<<num<<endl;

return 0;

}

#include<bits/stdc++.h> //拓扑排序链表

using namespace std;

#define mmax(x,y) ((x)>(y)?)(x):(y)

list<int> L[100];

queue<int> q;

int indgree[100]; //入度

int show[100];

void add(int a,int b)

{

L[a].push\_back(b);

indgree[b]++;

show[a]=1;

show[b]=1;

}

void enadd(int i)

{

q.push(i);

show[i]=0;

while(!q.empty())

{

int t=q.front();

cout<<t<<endl;

q.pop();

list<int>::iterator itor;

for(itor=L[t].begin();itor!=L[t].end();++itor)

{

indgree[\*itor]--;

if(indgree[\*itor]==0&&show[\*itor])

q.push(\*itor),show[\*itor]=0;

}

}

}

int main()

{

int n;

while(cin>>n&&n)

{

for(int i=0;i<100;i++)

indgree[i]=0,show[i]=0,L[i].clear();

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

{

int a,b;

cin>>a>>b;

add(a,b);

}

for(int i=0;i<100;i++)

{

if(!indgree[i]&&show[i])

enadd(i);

}

}

return 0;

}

#include <bits/stdc++.h> //拓扑排序dfs

using namespace std;

int G[MAXN][MAXN],c[MAXN],t,topo[MAXN];

int n;

bool dfs(int u)

{

c[u]=-1;

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

if(G[u][v])

{

if(c[v]<0)

return false;

else if(!c[v]&&!dfs(v))

return false;

}

c[u]=1;topo[--t]=u;

return true;

}

bool toposort()

{

memset(c,0,sizeof(c));

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

if(!c[u])

if(!dfs(u))

return false;

return true;

}

int main()

{

cin>>n;

t=MAXN;

while(n--)

{

int u,v;

cin>>u>>v;

G[u][v]=1;

}

n=MAXN-1;

if(toposort())

{

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

cout<<topo[i]<<endl;

}

else

cout<<"wrong"<<endl;

return 0;

}

char str[3000],str1[1005]; //马拉车算法

int mmax,po,MR,i,j,k,RL[3000];

cin>>str1;

str[0]='#';

for(i=0,j=1;str1[i]!='\0';i++,j++)

{

str[j]=str1[i];

str[++j]='#';

}

//cout<<str<<endl<<str1<<endl;

memset(RL,0,sizeof(RL));

po=0,MR=0,k=0,mmax=0;

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

{

if(i<MR)

RL[i]=min(RL[2\*po-i],MR-i);

else

RL[i]=1;

while(i>=RL[i]&&i+RL[i]<j&&str[i-RL[i]]==str[i+RL[i]])

RL[i]++;

if(i+RL[i]-1>MR)

{

MR=i+RL[i]-1;

po=i;

}

mmax=max(mmax,RL[i]);

}

cout<<mmax-1<<endl;

LL pow\_mod(LL a, LL b, LL p){//a的b次方求余p //快速幂求逆元

LL ret = 1;

while(b){

if(b & 1) ret = (ret \* a) % p;

a = (a \* a) % p;

b >>= 1;

}

return ret;

}

LL Fermat(LL a, LL p){//费马求a关于b的逆元

return pow\_mod(a, p-2, p);

}

#include<cstdio> //扩展欧几里得求逆元，比上面的快

typedef long long LL;

void ex\_gcd(LL a, LL b, LL &x, LL &y, LL &d){

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

else{

ex\_gcd(b, a % b, y, x, d);//--- > //那么我们可以得到 a\*x1 + b\*y1 = b\*x2 +(a%b)\*y2 = 1，又因为 a%b = a - a/b\*b，带入前式，

//我们可以得到 a\*x1 + b\*y1 = a\*y2 + b\*(x2-a/b\*y2)，由于a,b是任意的互质正整数，故可由该式得到方程(\*)与方程(\*\*)的解的关系如下：

//x1 = y2，

//y1 = x2-a/b\*y2，即方程(\*)的解可由方程(\*\*)的解得到。

y -= x \* (a / b);

}

}

LL inv(LL t, LL p){//如果不存在，返回-1

LL d, x, y;

ex\_gcd(t, p, x, y, d);

if(d==1)

{

int ans=x%p;

while(ans<=0)

ans+=p;

return ans;

}

return -1;

}

int main(){

LL a, p;

while(~scanf("%lld%lld", &a, &p)){

printf("%lld\n", inv(a, p));

}

}

#include<cstdio> //O(n)求n个数的逆元

const int N = 200000 + 5;

const int MOD = (int)1e9 + 7;

int inv[N];

int init(){

inv[1] = 1;

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

inv[i] = (MOD - MOD / i) \* inv[MOD % i] % MOD;

}

}

int main(){

init();

}

#include<cstdio> //孙子定理

#include<algorithm>

using namespace std;

typedef long long LL;

typedef pair<LL, LL> PLL;

PLL linear(LL A[], LL B[], LL M[], int n) {//求解A[i]x = B[i] (mod M[i]),总共n个线性方程组

LL x = 0, m = 1;

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

LL a = A[i] \* m, b = B[i] - A[i]\*x, d = gcd(M[i], a);

if(b % d != 0) return PLL(0, -1);//答案不存在，返回-1

LL t = b/d \* inv(a/d, M[i]/d)%(M[i]/d);

x = x + m\*t;

m \*= M[i]/d;

}

x = (x % m + m ) % m;

return PLL(x, m);//返回的x就是答案，m是最后的lcm值

}

//n个方程：x=a[i](mod m[i]) (0<=i<n) //同余定理

LL china(int n, LL \*a, LL \*m){

LL M = 1, ret = 0;

for(int i = 0; i < n; i ++) M \*= m[i];

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

LL w = M / m[i];

ret = (ret + w \* inv(w, m[i]) \* a[i]) % M;

}

return (ret + M) % M;

}

#include<bits/stdc++.h> //矩阵快速幂

typedef long long LL;

const int N=2;

const LL mod=1000000007;

using namespace std;

struct Matrix

{

LL m[N][N];

};

Matrix A=

{

1,1,

1,0

};

Matrix I=

{

1,0,

0,1

};

Matrix multi(Matrix a,Matrix b)

{

Matrix c;

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

{

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

{

c.m[i][j]=0;

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

c.m[i][j]+=a.m[i][k]\*b.m[k][j]%(mod-1);

c.m[i][j]%=mod-1;

}

}

return c;

}

Matrix power(Matrix A,LL k)

{

Matrix ans=I,p=A;

int i,j;

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

for(j=0;j<2;j++)

ans.m[i][j]=1;\*/

while(k>0)

{

if(k&1)

{

ans=multi(ans,p);

}

k>>=1;

p=multi(p,p);

}

return ans;

}

卢卡斯定理：大组合数

C(n, m) % p  =  C(n / p, m / p) \* C(n%p, m%p) % p

LL Lucas(LL n, LL m, int p){

return m ? Lucas(n/p, m/p, p) \* comb(n%p, m%p, p) % p : 1;

}

线段树（1）：

#include<cstdio>

using namespace std;

int n,p,a,b,m,x,y,ans;

struct node

{

int l,r,w,f;

}tree[400001];

inline void build(int k,int ll,int rr)//建树

{

tree[k].l=ll,tree[k].r=rr;

if(tree[k].l==tree[k].r)

{

scanf("%d",&tree[k].w);

return;

}

int m=(ll+rr)/2;

build(k\*2,ll,m);

build(k\*2+1,m+1,rr);

tree[k].w=tree[k\*2].w+tree[k\*2+1].w;

}

inline void down(int k)//标记下传

{

tree[k\*2].f+=tree[k].f;

tree[k\*2+1].f+=tree[k].f;

tree[k\*2].w+=tree[k].f\*(tree[k\*2].r-tree[k\*2].l+1);

tree[k\*2+1].w+=tree[k].f\*(tree[k\*2+1].r-tree[k\*2+1].l+1);

tree[k].f=0;

}

inline void ask\_point(int k)//单点查询

{

if(tree[k].l==tree[k].r)

{

ans=tree[k].w;

return ;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(x<=m) ask\_point(k\*2);

else ask\_point(k\*2+1);

}

inline void change\_point(int k)//单点修改

{

if(tree[k].l==tree[k].r)

{

tree[k].w+=y;

return;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(x<=m) change\_point(k\*2);

else change\_point(k\*2+1);

tree[k].w=tree[k\*2].w+tree[k\*2+1].w;

}

inline void ask\_interval(int k)//区间查询

{

if(tree[k].l>=a&&tree[k].r<=b)

{

ans+=tree[k].w;

return;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(a<=m) ask\_interval(k\*2);

if(b>m) ask\_interval(k\*2+1);

}

inline void change\_interval(int k)//区间修改

{

if(tree[k].l>=a&&tree[k].r<=b)

{

tree[k].w+=(tree[k].r-tree[k].l+1)\*y;

tree[k].f+=y;

return;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(a<=m) change\_interval(k\*2);

if(b>m) change\_interval(k\*2+1);

tree[k].w=tree[k\*2].w+tree[k\*2+1].w;

}

int main()

{

scanf("%d",&n);//n个节点

build(1,1,n);//建树

scanf("%d",&m);//m种操作

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

{

scanf("%d",&p);

ans=0;

if(p==1)

{

scanf("%d",&x);

ask\_point(1);//单点查询,输出第x个数

printf("%d",ans);

}

else if(p==2)

{

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

change\_point(1);//单点修改

}

else if(p==3)

{

scanf("%d%d",&a,&b);//区间查询

ask\_interval(1);

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

}

else

{

scanf("%d%d%d",&a,&b,&y);//区间修改

change\_interval(1);

}

}

}

线段树（2）：

#include<iostream>

using namespace std;

#define MAXN 100

#define MAXIND 256 //线段树节点个数

//构建线段树,目的:得到M数组.

void build(int node, int b, int e, int M[], int A[])

{

if (b == e)

M[node] = b; //只有一个元素,只有一个下标

else

{

build(2 \* node, b, (b + e) / 2, M, A);

build(2 \* node + 1, (b + e) / 2 + 1, e, M, A);

if (A[M[2 \* node]] <= A[M[2 \* node + 1]])

M[node] = M[2 \* node];

else

M[node] = M[2 \* node + 1];

}

}

//找出区间 [i, j] 上的最小值的索引

int query(int node, int b, int e, int M[], int A[], int i, int j)

{

int p1, p2;

//查询区间和要求的区间没有交集

if (i > e || j < b)

return -1;

if (b >= i && e <= j)

return M[node];

p1 = query(2 \* node, b, (b + e) / 2, M, A, i, j);

p2 = query(2 \* node + 1, (b + e) / 2 + 1, e, M, A, i, j);

//return the position where the overall

//minimum is

if (p1 == -1)

return M[node] = p2;

if (p2 == -1)

return M[node] = p1;

if (A[p1] <= A[p2])

return M[node] = p1;

return M[node] = p2;

}

int main()

{

int M[MAXIND]; //下标1起才有意义,否则不是二叉树,保存下标编号节点对应区间最小值的下标.

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

int a[]={3,4,5,7,2,1,0,3,4,5};

build(1, 0, sizeof(a)/sizeof(a[0])-1, M, a);

cout<<query(1, 0, sizeof(a)/sizeof(a[0])-1, M, a, 0, 5)<<endl;

return 0;

}

#include<bits/stdc++.h> //AC自动机

using namespace std;

const int maxn = 1e7 + 5;

const int MAX = 10000000;

int cnt;

struct node{

node \*next[26];

node \*fail;

int sum;

};

node \*root;

char key[70];

node \*q[MAX];

int head,tail;

node \*newnode;

char pattern[maxn];

int N;

void Insert(char \*s)

{

node \*p = root;

for(int i = 0; s[i]; i++)

{

int x = s[i] - 'a';

if(p->next[x] == NULL)

{

newnode=(struct node \*)malloc(sizeof(struct node));

for(int j=0;j<26;j++) newnode->next[j] = 0;

newnode->sum = 0;newnode->fail = 0;

p->next[x]=newnode;

}

p = p->next[x];

}

p->sum++;

}

void build\_fail\_pointer()

{

head = 0;

tail = 1;

q[head] = root;

node \*p;

node \*temp;

while(head < tail)

{

temp = q[head++];

for(int i = 0; i <= 25; i++)

{

if(temp->next[i])

{

if(temp == root)

{

temp->next[i]->fail = root;

}

else

{

p = temp->fail;

while(p)

{

if(p->next[i])

{

temp->next[i]->fail = p->next[i];

break;

}

p = p->fail;

}

if(p == NULL) temp->next[i]->fail = root;

}

q[tail++] = temp->next[i];

}

}

}

}

void ac\_automation(char \*ch)

{

node \*p = root;

int len = strlen(ch);

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

{

int x = ch[i] - 'a';

while(!p->next[x] && p != root) p = p->fail;

p = p->next[x];

if(!p) p = root;

node \*temp = p;

while(temp != root)

{

if(temp->sum >= 0)

{

cnt += temp->sum;

temp->sum = -1;

}

else break;

temp = temp->fail;

}

}

}

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

root=(struct node \*)malloc(sizeof(struct node));

for(int j=0;j<26;j++) root->next[j] = 0;

root->fail = 0;

root->sum = 0;

scanf("%d",&N);

getchar();

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

{

gets(key);

Insert(key);

}

gets(pattern);

cnt = 0;

build\_fail\_pointer();

ac\_automation(pattern);

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

}

return 0;

}

#include <iostream>ac自动机//快

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <string>

#include <queue>

#include <vector>

using namespace std;

const int N = 100005;

const int M = 26;

queue<int> q;

vector<string> vec;

bool vis[N];

struct Trie {

int trieN;

int ch[N][M], val[N], fail[N];

void init() {

memset(vis,0,sizeof(vis));

trieN = -1;

newnode();

}

int newnode() {

memset(ch[++trieN], 0, sizeof(ch[0]));

val[trieN] = fail[trieN] = 0;

return trieN;

}

void insert(const string &str, int index) {

int cur = 0;

for (int i = 0;str[i];i++) {

int d = str[i] - 'a';

if (!ch[cur][d])

ch[cur][d] = newnode();

cur = ch[cur][d];

}

if (val[cur]) vis[index] = 1;

else val[cur] = index;

}

void build() {

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

if (ch[0][i])

q.push(ch[0][i]);

}

while (!q.empty()) {

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

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

int &next = ch[cur][i];

if (next) {

fail[next] = ch[fail[cur]][i];

q.push(next);

}

else next = ch[fail[cur]][i];

}

}

}

void query(const string &str) {

int cur = 0, tmp;

for (int i = 0;str[i];i++) {

int d = str[i] - 'a';

tmp = cur = ch[cur][d];

while (tmp && ~val[tmp]) {

if (val[tmp] != -1) vis[val[tmp]] = 1;

val[tmp] = -1;

tmp = fail[tmp];

}

}

}

}ac;

int main(){

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

cin.tie(0);

int t;

cin >> t;

while(t--){

ac.init(); //初始化

int n;

string str;

cin >> n;

vec.resize(n+1);

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

cin >> vec[i];

ac.insert(vec[i],i);

}

ac.build();

cin>>str;

ac.query(str); //查询是否在str里出现过

bool flag = 1;

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

if (!vis[i]) flag = 0;

}

if (flag) cout <<"Yes"<< '\n';

else cout << "No\n";

}

return 0;

}

#include<iostream>弱连通分量判断

#include<cstdio>

#include<vector>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn=1000+10;

int instack[maxn],dfn[maxn],low[maxn],belong[maxn],in[maxn],out[maxn];

vector<vector<int>> G(maxn);

vector<int> s;

int n,m;

int cnt=0,idex=0;

void Tarjan(int u)

{

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

s.push\_back(u);

instack[u]=1;

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

{

int v=G[u][i];

if(!dfn[v])

{

Tarjan(v);

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

}

else if(instack[v])

{

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

}

}

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

{

cnt++;

int p;

do{

p=s[s.size()-1];

instack[p]=0;

belong[p]=cnt;

s.pop\_back();

}while(u!=p);

}

}

bool Count()

{

memset(in,0,sizeof(in));

memset(out,0,sizeof(out));

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

{

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

{

int v=G[i][j];

if(belong[i]!=belong[v])

{

out[belong[i]]++;

in[belong[v]]++;

}

}

}

int ans=0;

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

{

if(!out[i]) ans++;

if(!in[i]) ans++;

}

if(ans>2) return false;

return true;

}

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

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

cnt=0;idex=0;

s.clear();

for(int i=1;i<=n;i++) G[i].clear();

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

memset(low,0,sizeof(low));

memset(instack,0,sizeof(instack));

memset(belong,0,sizeof(belong));

int a,b;

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

{

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

G[a].push\_back(b);

}

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

{

if(!dfn[i])

{

Tarjan(i);

}

}

if(Count()) printf("Yes\n");

else printf("No\n");

}

return 0;

}

#include <stdio.h> //强联通分量判断

#include <string.h>

#include <vector>

#include <stack>

using namespace std;

#define MIN(a,b) ((a)<(b)?(a):(b))

#define N 10005 // 题目中可能的最大点数

stack<int>sta; // 存储已遍历的结点

vector<int>gra[N]; // 邻接表表示图

int dfn[N]; // 深度优先搜索访问次序

int low[N]; // 能追溯到的最早的次序

int InStack[N]; // 检查是否在栈中(2为在栈中，1为已访问，且不在栈中，0为不在)

vector<int> Component[N]; // 获得强连通分量结果

int InComponent[N]; // 记录每个点在第几号强连通分量里

int index,ComponentNumber; // 索引号，强连通分量个数

int n, m; // 点数，边数

void init(void)

{

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

memset(low, 0, sizeof(low));

memset(InStack, 0, sizeof(InStack));

index = ComponentNumber = 0;

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

{

gra[i].clear();

Component[i].clear();

}

while(!sta.empty())

sta.pop();

}

void tarjan(int u)

{

InStack[u] = 2;

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

sta.push(u);

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

{

int t = gra[u][i];

if (dfn[t] == 0)

{

tarjan(t);

low[u] = MIN(low[u], low[t]);

}

else if (InStack[t] == 2)

{

low[u] = MIN(low[u], dfn[t]);

}

}

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

{

++ ComponentNumber; //；连通分量数目

while (!sta.empty())

{

int j = sta.top();

sta.pop();

InStack[j] = 1;

Component[ComponentNumber].push\_back(j);

InComponent[j]=ComponentNumber;

if (j == u)

break;

}

}

}

void input(void)

{

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

{

int a,b;

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

gra[a].push\_back(b);

}

}

void solve(void)

{

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

if(!dfn[i])

tarjan(i);

if(ComponentNumber>1)

puts("No");

else

puts("Yes");

}

int main()

{

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

{

init();

input();

solve();

}

}

#include <iostream>

#include<cstdlib>

#define MAX 26

using namespace std;

typedef struct TrieNode //Trie结点声明

{

bool isStr; //标记该结点处是否构成单词

struct TrieNode \*next[MAX]; //儿子分支

}Trie;

void insert(Trie \*root,const char \*s) //将单词s插入到字典树中

{

if(root==NULL||\*s=='\0')

return;

int i;

Trie \*p=root;

while(\*s!='\0')

{

if(p->next[\*s-'a']==NULL) //如果不存在，则建立结点

{

Trie \*temp=(Trie \*)malloc(sizeof(Trie));

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

{

temp->next[i]=NULL;

}

temp->isStr=false;

p->next[\*s-'a']=temp;

p=p->next[\*s-'a'];

}

else

{

p=p->next[\*s-'a'];

}

s++;

}

p->isStr=true; //单词结束的地方标记此处可以构成一个单词

}

int search(Trie \*root,const char \*s) //查找某个单词是否已经存在

{

Trie \*p=root;

while(p!=NULL&&\*s!='\0')

{

p=p->next[\*s-'a'];

s++;

}

return (p!=NULL&&p->isStr==true); //在单词结束处的标记为true时，单词才存在

}

void del(Trie \*root) //释放整个字典树占的堆区空间

{

int i;

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

{

if(root->next[i]!=NULL)

{

del(root->next[i]);

}

}

free(root);

}

int main(int argc, char \*argv[])

{

int i;

int n,m; //n为建立Trie树输入的单词数，m为要查找的单词数

char s[100];

Trie \*root= (Trie \*)malloc(sizeof(Trie));

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

{

root->next[i]=NULL;

}

root->isStr=false;

scanf("%d",&n);

getchar();

for(i=0;i<n;i++) //先建立字典树

{

scanf("%s",s);

insert(root,s);

}

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

{

for(i=0;i<m;i++) //查找

{

scanf("%s",s);

if(search(root,s)==1)

printf("YES\n");

else

printf("NO\n");

}

printf("\n");

}

del(root); //释放空间很重要

return 0;

}

#include<iostream> //字典树

#include<cstring>

#include<cstdio>

using namespace std;

typedef long long LL;

const LL mod=1e9+7;

const int MAX=11;

const char o='0';

char a[10005][11];

int flag;

struct trie{

bool isstr; //根据需要可设置其它变量

trie \*next[MAX];

trie()

{

isstr=0;

memset(next,0,sizeof(next));

}

}t[100000];

int num;

void inst(trie \*root,char \*s){

trie \*p=root;

int i=0;

while(s[i]){

int k=s[i]-o;

if(p->next[k]==NULL)

p->next[k]=&t[num++];

else{

if(p->next[k]->isstr==1)

flag=1;

else if(s[i+1]=='\0')

flag=1;

}

i++;

p=p->next[k];

}

p->isstr=true;

}

//bool ask(trie \*root,char \*s){

// trie \*p=root;

// int i=0;

// while(s[i]){ //单词查询

// int k=s[i++]-o;

// p=p->next[k];

// if(p->isstr&&s[i])

// return 0;

// }

// return 1;

//}

int main()

{

int n,m; //n个单词，m个查询，复杂度O（n\*len）

int T;

cin>>T;

while(T--){

cin>>n;

memset(t,0,sizeof(t));

trie \*root=&t[0]; //分配空间

num=1;

flag=0;

int i,j,k;

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

{

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

inst(root,a[i]);

}

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

// if(ask(root,a[i]))

// continue;

// flag=1;

// break;

// }

if(flag)

cout<<"NO\n";

else

cout<<"YES\n";

}

return 0;

}

#include<bits/stdc++.h>完全背包

using namespace std;

typedef long long LL;

LL F[1000005];

LL wi[1000005];

LL vi[1000005];

int main()

{

int m,n;

int i,j,k;

while(cin>>m>>n){

memset(F,0,sizeof(F));

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

cin>>wi[i]>>vi[i];

}

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

for(j=wi[i];j<=m;j++)

F[j]=max(F[j],F[j-wi[i]]+vi[i]);

cout<<F[m]<<endl;

}

}

#include<cstdio>价值背包

#include<cstdlib>

#include<iostream>

#include<algorithm>

#include<cmath>

#include<cstring>

#include<map>

#define max(a,b)(a>b?a:b)

#define min(a,b)(a<b?a:b)

typedef long long ll;

using namespace std;

#define N 5100

ll dp[N],w[N]; ///此时的dp[i]表示的是；价值为i时的最小容量为dp[i];

int v[N];

int main()

{

int T,i,n,sum;

ll V;

scanf("%d",&T);

while(T--)

{

scanf("%d%I64d",&n,&V);

sum=0;

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

{

scanf("%I64d%d",&w[i],&v[i]);

sum=sum+v[i];

}

memset(dp,1000000010,sizeof(dp)); ///要求最小容量，初始化为最大值；

dp[0]=0;

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

{

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

dp[j]=min(dp[j],dp[j-v[i]]+w[i]);

}

for(i=sum;i>=0;i--)

{

if(dp[i]<=V)

{

printf("%d\n",i); ///此处输出i，即为满足条件的最大价值

break;

}

}

}

return 0;

}

#include<bits/stdc++.h>多重背包

typedef long long LL;

const int N=2;

const LL mod=1000000007;

using namespace std;

struct nc{

LL wei;

LL pr;

};

nc c[1000005];

LL ans[1000005]={0};

int main()

{

int n,m;

while(cin>>m>>n){

int i,j,k;

int dpr,dwei,snum;

int sum=0;

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

int k=1;

cin>>dwei>>dpr>>snum;

while(snum-k>0){ //把物品拆开来

snum-=k;

c[++sum].wei=k\*dwei;

c[sum].pr=k\*dpr;

k\*=2;

}

c[++sum].wei = dwei\*snum;

c[sum].pr = dpr\*snum;

}

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

ans[j]=0;

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

for(j=m;j>=c[i].wei;j--)

ans[j]=max(ans[j],ans[j-c[i].wei]+c[i].pr);

cout<<ans[m]<<endl;

}

return 0;

}

#include<bits/stdc++.h> //矩阵快速幂

typedef long long LL;

const int N=2;

const LL mod=1000000007;

using namespace std;

struct Matrix

{

LL m[N][N];

};

Matrix A= //系数矩阵

{

1,1,

1,0

};

Matrix I= //单位矩阵||初始矩阵

{

1,0,

0,1

};

Matrix multi(Matrix a,Matrix b)

{

Matrix c;

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

{

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

{

c.m[i][j]=0;

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

c.m[i][j]+=a.m[i][k]\*b.m[k][j]%(mod);

c.m[i][j]%=mod;

}

}

return c;

}

Matrix power(Matrix A,LL k)

{

Matrix ans=I,p=A;

while(k>0)

{

if(k&1)

{

ans=multi(ans,p);

}

k>>=1;

p=multi(p,p);

}

return ans;

}

JAVA

1.大浮点数运算的位数规定：

MathContext mc = new MathContext(2, RoundingMode.HALF\_DOWN);  
//精度为2，舍入模式为大于0.5进1，否则舍弃。   
BigDecimal a = new BigDecimal(0.5);  
BigDecimal b = new BigDecimal(0.2);  
System.out.println(a.divide(b,mc)); //保留两位数字计算

2.进制转换

int a=7;

String st=Integer.toString(a,2); //把数字a当成10进制转换成2进制的字符串；

System.out.println(st);

int num=Integer.parseInt(st,2); //把字符串当成2进制转换成10进制的数字；

System.out.println(num);

String st="10000000";

String ss = new BigInteger(st,2).toString(10);

System.out.println(ss); // st是字符串，2是st的原进制，10是目标进制

3. 排序

函数：Arrays.sort()；

例程：

import java.io.\*;

import java.math.\*;

import java.util.\*;

import java.text.\*;

public class Main

{

    public static void main(String[] args)

    {

        Scanner cin = new Scanner (new BufferedInputStream(System.in));

        int n = cin.nextInt();

        int a[] = new int [n];

        for (int i = 0; i < n; i++) a[i] = cin.nextInt();

        Arrays.sort(a);

        for (int i = 0; i < n; i++) System.out.print(a[i] + " ");

    }

}

7. 结构体排序：

例子：一个结构体有两个元素String x，int y，排序，如果x相等y升序，否者x升序。

一、Comparator

强行对某个对象collection进行整体排序的比较函数，可以将Comparator传递给Collections.sort或Arrays.sort。

接口方法：这里也给出了两种方法，

import java.util.\*;

class structSort{

    String x;

    int y;

}

class cmp implements Comparator<structSort>{

public int compare(structSort o1, structSort o2) {

if(o1.x.compareTo(o2.x) == 0){//这个相当于c/c++中strcmp（o1.x , o2,x）

return o1.y - o2.y;

}

return o1.x.compareTo(o2.x);

}

}

public class Main {

public static void main(String[] args) {

 Comparator<structSort> comparator = new Comparator<structSort>(){

public int compare(structSort o1, structSort o2) {

if(o1.x.compareTo(o2.x) == 0){

return o1.y - o2.y;

}

return o1.x.compareTo(o2.x);

}

 };

 Scanner cin = new Scanner(System.in);

 int n = cin.nextInt();

 structSort a[] = new structSort[10];

 for (int i = 0; i < n; i++) {

a[i] = new structSort();

a[i].x = cin.next();

a[i].y = cin.nextInt();

}

 Arrays.sort(a,0,n,comparator);//这个直接使用Comparator

 Arrays.sort(a,0,n,new cmp());//这个实现Comparator，就就跟c++中的sort函数调用就差不多了

 for (int i = 0; i < n; i++) {

System.out.println(a[i].x+" "+a[i].y);

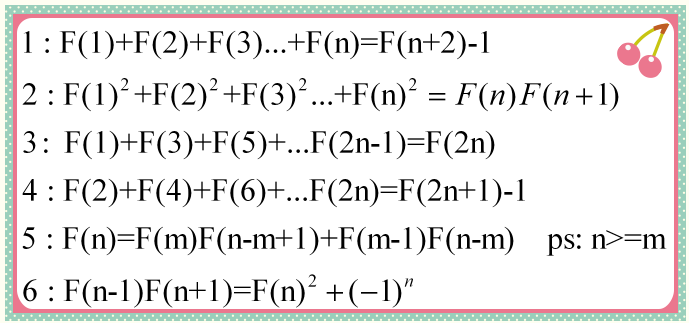
}

}

}

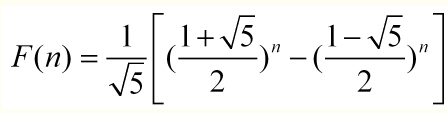
结论篇

1.判断组合数C(n,m)的奇偶性当n&m==m为奇数，反之就是偶数

2.







3.f(0)-f⑴+f⑵-…+(-1)^n·f(n)=(-1)^n·[f(n+1)-f(n)]-1

4.f(n+m)=f(n+1)·f(m)+f(n)·f(m-1)

5.随着数列项数的增加，前一项与后一项之比越来越逼近黄金分割的数值0.6180339887..…

从第二项开始，每个奇数项的平方都比前后两项之积多1，每个偶数项的平方都比前后两项之积少1。

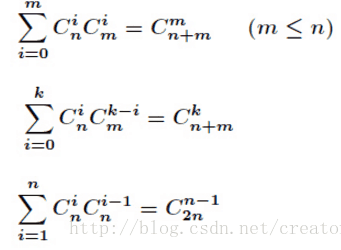
斐波那契数列的第n项同时也代表了集合{1,2,...,n}中所有不包含相邻正整数的子集个数。

6.如果fib(k)能被x整除，则fib(k\*i)都可以被x整除

7.f(2n-1)=[f(n)]^2-[f(n-2)]^2

8.3f(n)=f(n+2)+f(n-2)

9.f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m)

10.

11