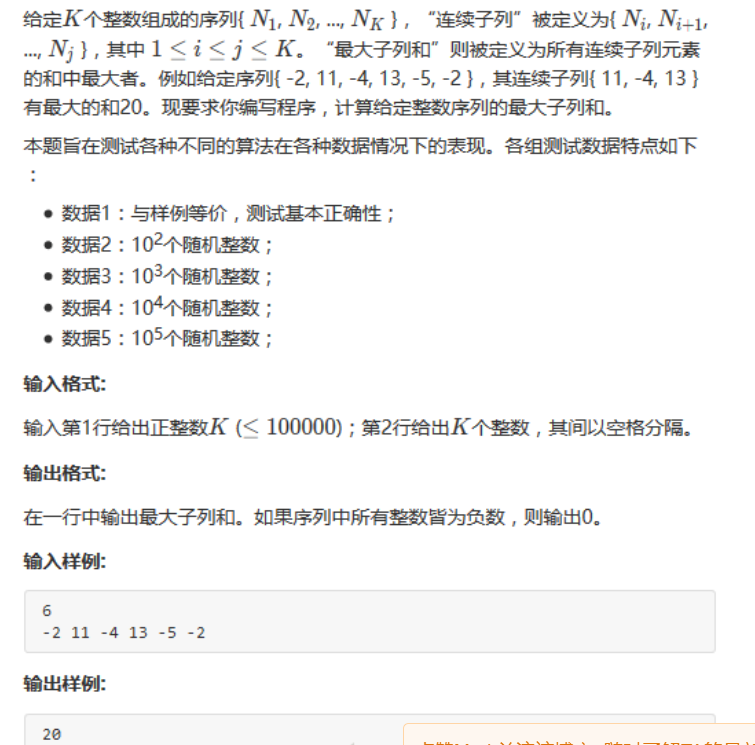
**PTA题目合集**

1. 连续子列与最大子列和
2. 树的同构
3. 判断不同的插入序列是否得到相同的二叉排序树
4. 堆中的路径
5. 列出联通集
6. 六度空间
7. 旅游规划（最短最便宜的路径）
8. 公路村村通（最低成本）
9. 排序（长整型）
10. 哈利波特的考试
11. 电话聊天狂人
12. QQ账户的申请与登录
13. 一元多项式求导
14. 汉诺塔的非递归实现
15. 银行业务队列的简单模拟
16. 求链式线性表的导数第K项
17. 表达式转换
18. 求前缀表达式的值
19. 堆栈模拟队列
20. 还原二叉树
21. 树种统计
22. 朋友圈
23. Windows消息队列
24. 家谱处理
25. 搜索树判断
26. 修理牧场
27. 目录树
28. 笛卡尔树
29. 任务调度的合理性
30. 城市间紧急救援
31. 社交网络中节点的重要性计算
32. 模拟EXCEL排序
33. 寻找大富翁
34. 魔法优惠券
35. PAT排名汇总
36. 整型关键字的散列映射
37. 字符串关键字的散列映射
38. 基于词频的文件相似度
39. 航空公司VIP客户查询
40. 新浪微博热搜话题
41. 打印选课学生名单
42. 银行排队问题之单窗口“夹塞”版
43. 两个有序链表序列的交集
44. 打印学生选课清单



int MaxSubseqSum4(int A[], int N)

{

int ThisSum, MaxSum;

int i;

ThisSum = MaxSum = 0;

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

ThisSum += A[i];//向右累加

if (ThisSum>MaxSum)

MaxSum = ThisSum; // 发现更大和则更新当前结果

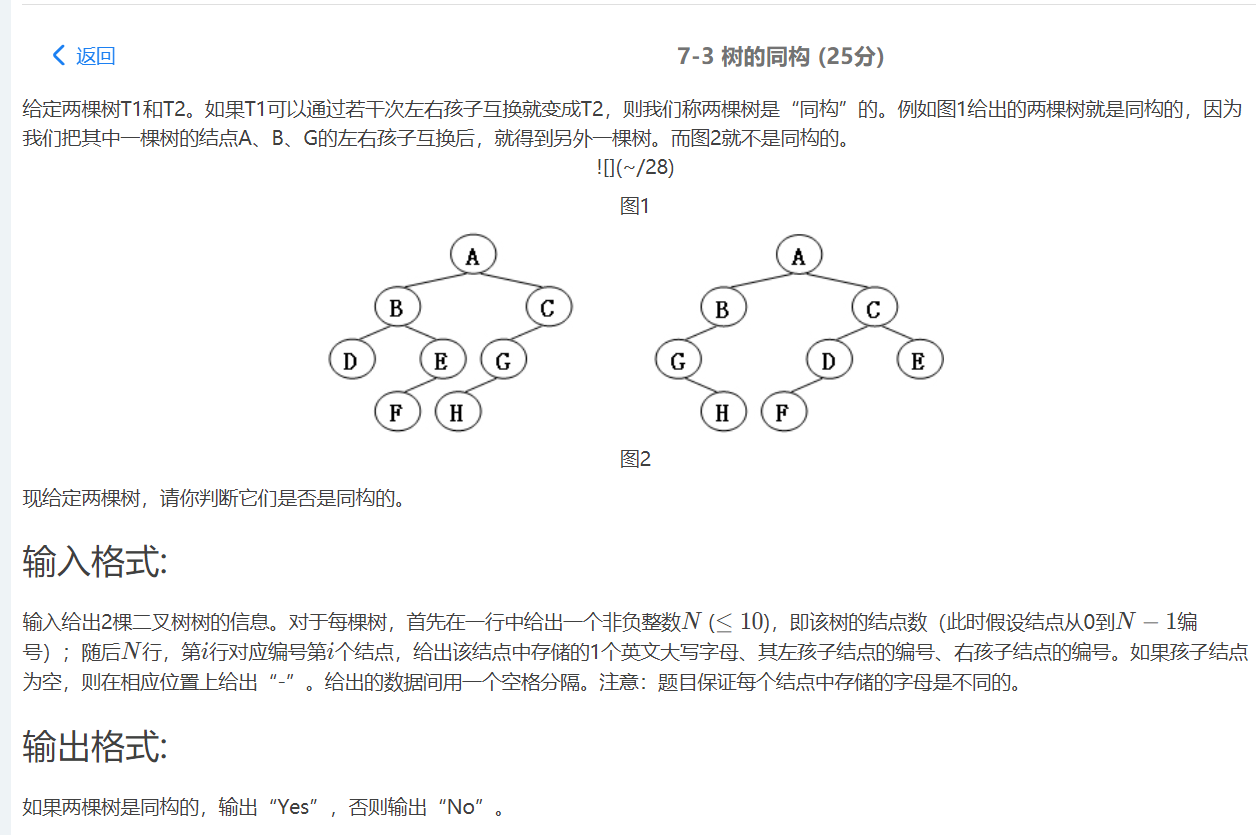
else if (ThisSum < 0) // 如果当前子列和为负数

ThisSum = 0; // 则不可能使后面部分和增大，抛弃之

}

return MaxSum;

}



#include <stdio.h>

#include <stdlib.h>

#define MaxTree 10

#define ElementType char

#define Tree int

#define Null -1

struct TreeNode

{

ElementType e;

Tree left;

Tree right;

}T1[MaxTree],T2[MaxTree];

Tree BuildTree ( struct TreeNode T[]);

int Isomorphic(Tree R1,Tree R2);

int main()

{

Tree R1,R2;

R1 = BuildTree(T1);

R2 = BuildTree(T2);

if( Isomorphic(R1,R2)){

printf("Yes\n");

}

else{

printf("No\n");

}

return 0;

}

Tree BuildTree ( struct TreeNode T[])

{

int i;

int n;

int check[MaxTree];

char cl,cr;

Tree root =Null; //若n为0，返回Null

scanf("%d",&n);

if( n ){

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

check[i] = 0;

}

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

scanf("\n%c %c %c",&T[i].e,&cl,&cr);

if( cl!='-' ){

T[i].left = cl-'0'; //若输入不为'-',那字符减去字符0转换为整型数值

check[T[i].left] = 1; //把在静态链表中出现过的数值标记为1

}

else if( cl=='-' ){

T[i].left = Null;

}

if( cr!='-' ){

T[i].right = cr-'0';

check[T[i].right] = 1;

}

else if( cr=='-' ){

T[i].right = Null;

}

}

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

if( !check[i]){

break;

}

}

root = i;

}

return root;

}

int Isomorphic(Tree R1,Tree R2){

if((R1==Null)&&(R2==Null)) //如果为空树则是同构的

return 1;

if(((R1==Null)&&(R2!=Null))||((R1!=Null)&&(R2==Null)))//如果一个为空一个不为空则不是同构的

return 0;

if((T1[R1].e)!=(T2[R2].e))//如果数据不同则不是同构的

return 0;

//如果左儿子都为空判断右儿子是否同构

if((T1[R1].left==Null)&&(T2[R2].left==Null))

return Isomorphic(T1[R1].right,T2[R2].right);

/\* 如果两棵树左儿子都不为空并且数据还是一样的，对左儿子进行递归\*/

if ( ((T1[R1].left!=Null)&&(T2[R2].left!=Null))&&((T1[T1[R1].left].e)==(T2[T2[R2].left].e)) )

return ( Isomorphic( T1[R1].left, T2[R2].left )&&Isomorphic( T1[R1].right, T2[R2].right ) );

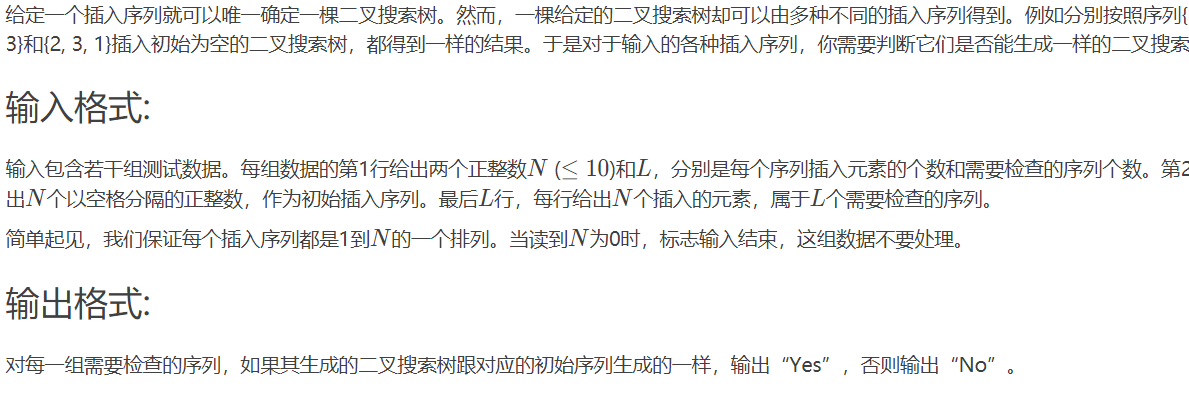
/\* 如果两棵树左儿子（一个空一个不空或者都不空）并且数据不一样，

那么判断第一棵树的左（右）儿子是否跟第二棵树的右（左）儿子同构 \*/

else

return ( Isomorphic( T1[R1].left, T2[R2].right)&&Isomorphic( T1[R1].right, T2[R2].left ) );

}



#include<stdio.h>

#include<stdlib.h>

typedef struct NODE{

int Data;

struct NODE \*Left;

struct NODE \*Right;

} Tree;

typedef enum TYPE{

SOURCR,

JUDGE

} Type;

void InorderTraversal( Tree\* BT ,Type type);

Tree\* Insert( Tree\* root, int X);

int \*sourceDATA;

int sourcePOS=0;

int \*judgeDATA;

int judgePOS=0;

int main() {

while(1){

int N,L;

scanf("%d",&N);

if(N==0){

return 0;

}

scanf("%d",&L);

sourceDATA=(int \*)malloc(sizeof(int)\*N);

judgeDATA=(int \*)malloc(sizeof(int)\*N);

int \*data=(int \*)malloc(sizeof(int)\*N);

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

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

}

Tree \*root=NULL;

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

root=Insert(root,data[i]);

}

InorderTraversal(root,SOURCR);

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

int isDo=1;

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

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

}

Tree \*judge=NULL;

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

judge=Insert(judge,data[i]);

}

InorderTraversal(judge,JUDGE);

int js=1;

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

if(sourceDATA[i]!=judgeDATA[i]){

js=0;

break;

}

}

judgePOS=0;

if(js){

printf("Yes\n");

}else{

printf("No\n");

}

}

sourcePOS=0;

}

return 0;

}

Tree\* Insert( Tree\* root, int X){

if(root==NULL){

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

root->Data=X;

root->Left=NULL;

root->Right=NULL;

return root;

}

Tree \*RT=(Tree\*)malloc(sizeof(Tree));

RT=root;

int flag=0;

while(RT!=NULL){

if(X>RT->Data){

if(RT->Right!=NULL){

RT=RT->Right;

}else{

flag=1;

break;

}

}else{

if(RT->Left!=NULL){

RT=RT->Left;

}else{

flag=2;

break;

}

}

}

if(flag==2){

RT->Left=(Tree\*)malloc(sizeof(Tree));

RT->Left->Data=X;

RT->Left->Left=NULL;

RT->Left->Right=NULL;

}else if(flag==1){

RT->Right=(Tree\*)malloc(sizeof(Tree));

RT->Right->Data=X;

RT->Right->Left=NULL;

RT->Right->Right=NULL;

}

return root;

}

void InorderTraversal( Tree\* BT ,Type type){

if(BT==NULL){

return;

}else{

if(type==SOURCR){

sourceDATA[sourcePOS++]=BT->Data;

}else{

judgeDATA[judgePOS++]=BT->Data;

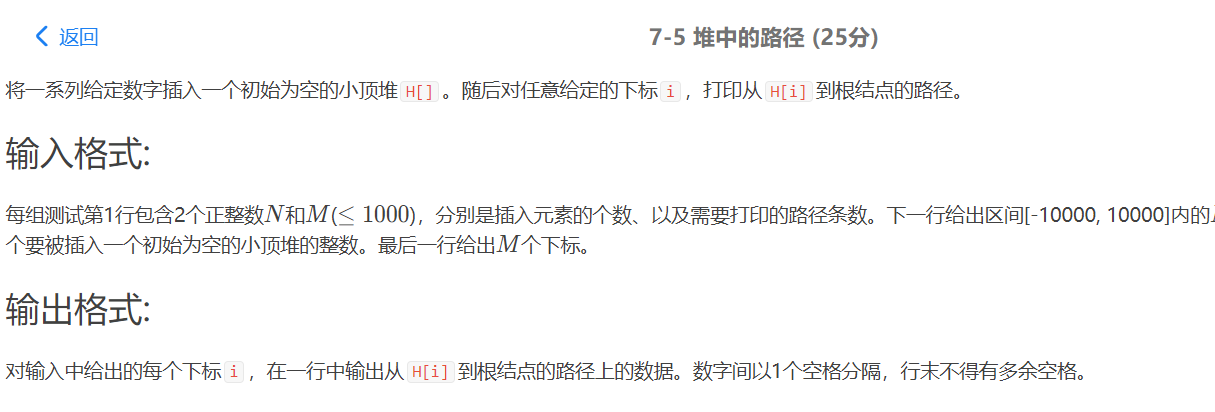
}

InorderTraversal(BT->Left,type);

InorderTraversal(BT->Right,type);

}

}



#include<iostream>

#include<stdlib.h>

#include<stdio.h>

#include<algorithm>

#include<vector>

using namespace std;

const int maxn=10010;

vector<int> a; //stl建堆的容器

int H[maxn]; //我们需要的堆

int N,M,x,t;

/\*打印堆中的路径\*/

void print\_heap(int i,int N)

{

while(i>1)

{

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

i=i/2;

}

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

}

int main()

{

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

{

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

{

scanf("%d",&x);

a.push\_back(x);

make\_heap(a.begin(),a.end(),greater<int>()); //每插入一个元素就建堆一次

}

for(int i=0;i<N;i++) //堆中元素下标应从1开始，但stl中建堆默认从0开始

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

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

{

scanf("%d",&t);

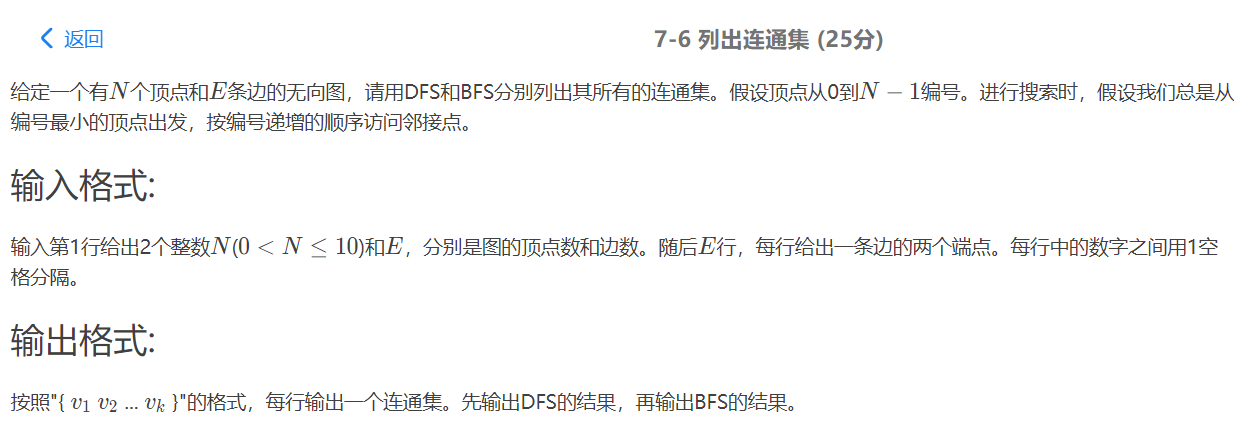
print\_heap(t,N);

}

}

return 0;

}



#include<stdio.h>

#include<stdlib.h>

#define MAXVEX 15

void CreateGraph( );

void DFS( int i);

void DFSTraverse();

void BFSTraverse();

int G[MAXVEX][MAXVEX],Nv,Ne;

int visited[MAXVEX];

int main()

{

CreateGraph();

DFSTraverse();

BFSTraverse();

return 0;

}

void CreateGraph()

{

//用邻接矩阵表示图

int i,j;

int v1,v2;

scanf("%d %d",&Nv,&Ne);

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

{

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

{

G[i][j] = 0; //初始化

}

}

for( i=0; i<Ne; i++) //注意这里是读入边

{

scanf("%d %d",&v1,&v2);

G[v1][v2] = 1;

G[v2][v1]= G[v1][v2]; //无向图对称

}

}

void DFS( int i)

{

int j;

visited[i] = 1;

printf("%d ",i);

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

{

if( G[i][j] && !visited[j])

{

//如果存在ij之间的连线且j并未被访问过

DFS (j);

}

}

}

void DFSTraverse( )

{

int i;

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

{

visited[i] = 0; //初始化访问矩阵

}

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

{

if( !visited[i])

{

printf("{ ");

DFS(i);

printf("}\n");

}

}

}

void BFSTraverse( )

{

int q[MAXVEX]={0}; //用数组表示队列

int rear=-1,front=-1;

int i,j;

int temp;

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

{

visited[i] = 0;

}

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

if( !visited[i]){

printf("{ ");

visited[i] =1;

q[++rear] = i; //入队

while( front<rear ){ //判断队列是否为空

temp =q[++front]; //出队

printf("%d ",temp);

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

if( G[temp][j] && !visited[j]){

visited[j] = 1;

q[++rear] = j;

}

}

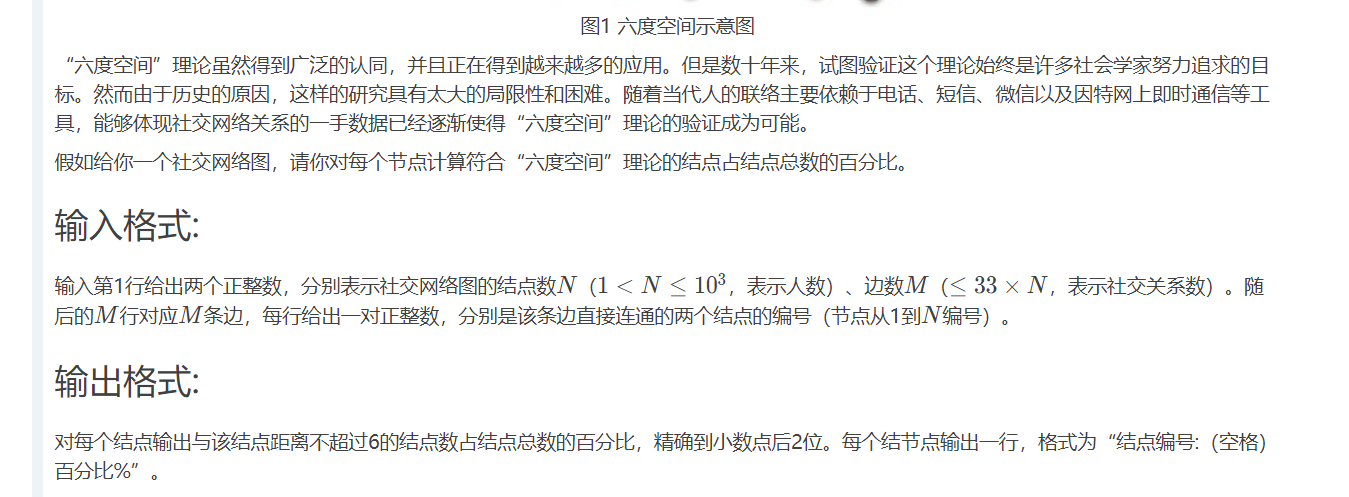
}

printf("}\n");

}

}

}



#include <iostream>

#include <cstdio>

#include <algorithm>

#include <string>

#include <vector>

#include <sstream>

#include <map>

#include <queue>

#include <stack>

#include <cstring>

#include <set>

using namespace std;

int G[10005][10005];

int n,m;

int cnt;

int vis[10005];

void DFS(int cur,int deep)

{

if(deep>6) return ;

cnt++;

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

{

if(G[cur][i]==1 && vis[i]==0)

{

vis[i]=1;

DFS(i,deep+1);

}

}

}

int main()

{

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

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

{

int x,y;

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

G[x][y]=G[y][x]=1;

}

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

{

cnt=0;

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

vis[i]=1;

DFS(i,0);

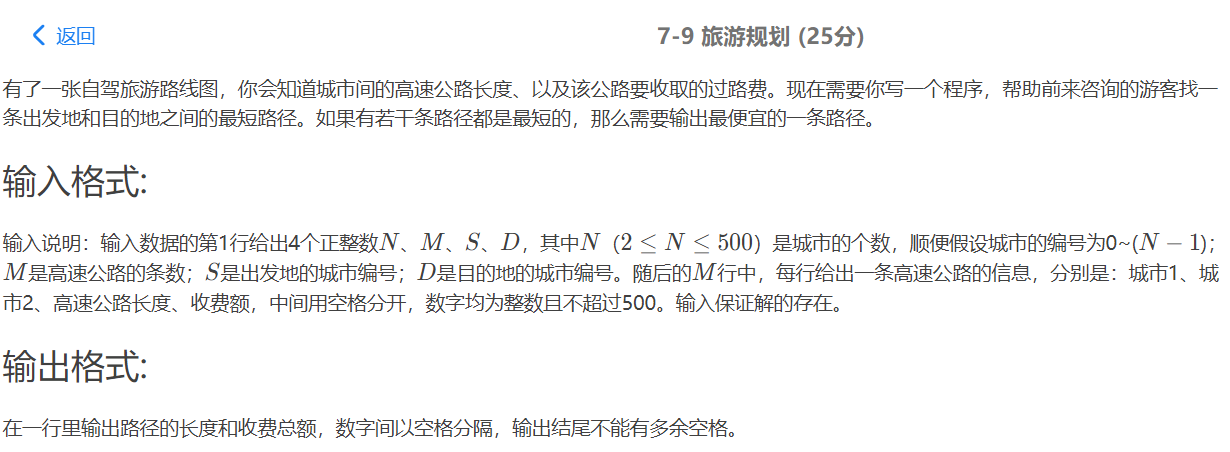
double ans=cnt\*1.0/n;

printf("%d: %.2f%%\n",i,ans\*100);

}

return 0;

}



#include<stdio.h>

#include<stdlib.h>

#define MAX 1000

int matrix[MAX][MAX], cost[MAX][MAX];

int i, j, k, m, n;

int x, y, z, w, a, b;

void flyd(){

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

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

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

if (matrix[i][k] + matrix[k][j] < matrix[i][j]) {

matrix[i][j] = matrix[i][k] + matrix[k][j];

cost[i][j] = cost[i][k] + cost[k][j];

//判断相反时是否也是最短路径

if(matrix[i][n-k-1] + matrix[n-k-1][j] == matrix[i][j]){

//是最短路径再进而判断其花费与原来的谁多，多的要换成少的

if(cost[i][j] > cost[i][n-k-1] + cost[n-k-1][j]){

cost[i][j]=cost[i][n-k-1] + cost[n-k-1][j];

}

}

}

}

}

}

}

int main() {

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

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

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

if (i != j) {

matrix[i][j] = MAX;

cost[i][j] = MAX;

}

}

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

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

matrix[x][y] = z;

matrix[y][x] = z;

cost[x][y] = w;

cost[y][x] = w;

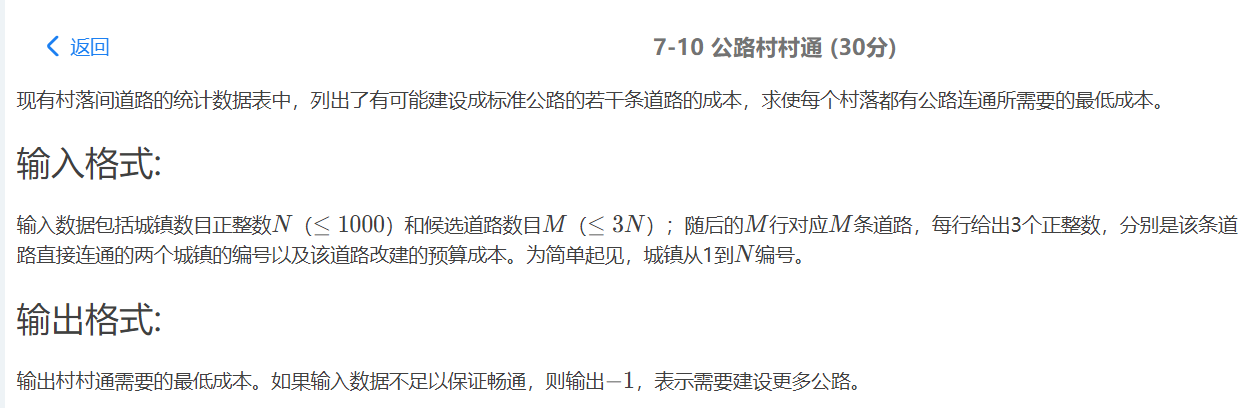
}

flyd();

printf("%d %d\n", matrix[a][b], cost[a][b]);

return 0;

}



#include<iostream>

using namespace std;

#define MAXN 1010

#define MAXM 3010

#define INF 0xffff

typedef struct GNode{

int n;

int e;

int AdjMatrix[MAXN][MAXN];

};

struct GNode g;

int lowcost[MAXN];

int closest[MAXN];

int cost = 0; //记录最小总花费

void Prim(int v)

{

int mincost;

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

{

lowcost[i] = g.AdjMatrix[v][i];

closest[i] = v;

}

for(int i = 1;i<g.n;i++)//寻找n-1个点

{

int mincost = INF;

int k = -1;

for(int j = 1;j<=g.n;j++) //寻找其中的最短边加入到v集合中

{

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

{

mincost = lowcost[j];

k = j;

}

}

if(k!=-1)

{

lowcost[k] = 0; //标记k找到

cost+=mincost; //花费增加

}

else //否则再也找不到这样的k

break;

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

{

if(g.AdjMatrix[k][j]!=0&&g.AdjMatrix[k][j]<lowcost[j])

{

lowcost[j] = g.AdjMatrix[k][j];

closest[j] = k;

}

}

}

}

int main()

{

int n,m;

cin>>n>>m;

g.e = m;

g.n = n;

//邻接矩阵初始化

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

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

{

if(i==j)

g.AdjMatrix[i][j] = 0;

else

g.AdjMatrix[i][j] = INF;

}

//构造邻接矩阵

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

{

int a,b,w;

cin>>a>>b>>w;

g.AdjMatrix[a][b] = g.AdjMatrix[b][a] = w;

}

Prim(1);

int flagAll = 1;

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

{

if(lowcost[i]!=0)

{

flagAll = 0;

break;

}

}

if(flagAll)

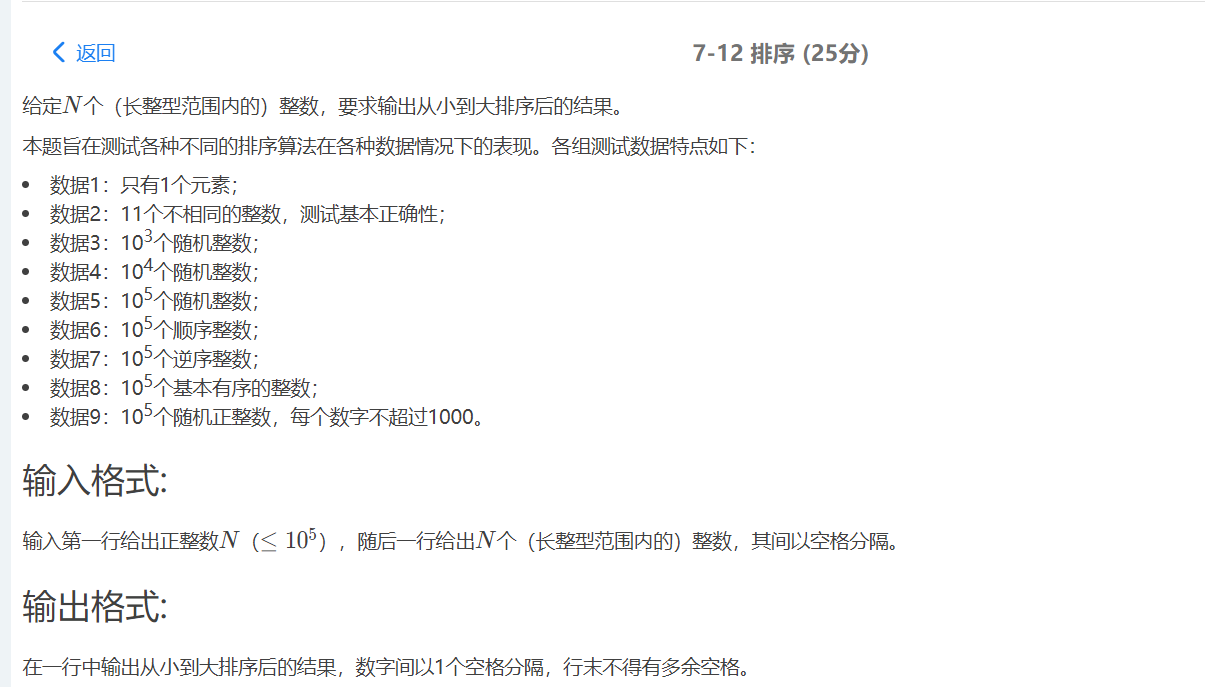
cout<<cost<<endl;

else

cout<<"-1"<<endl;

return 0;

}



#include<iostream>

#include<malloc.h>

using namespace std;

#define MAXN 100010

int num[MAXN];

//将arr[low...mid]和arr[mid+1...high]两个相邻的有序子序列归并为一个有序子序列arr[low...high]

void Merge(int arr[],int low,int high,int mid)

{

int \*tmp;

int i = low,j = mid+1;

int k = 0;

tmp = (int \*)malloc((high-low+1)\*sizeof(int));

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

{

if(arr[i]<arr[j])

{

tmp[k] = arr[i];

i++,k++;

}

else

{

tmp[k] = arr[j];

k++,j++;

}

}

while(i<=mid)

{

tmp[k] = arr[i];

i++,k++;

}

while(j<=high)

{

tmp[k] = arr[j];

j++,k++;

}

for(k = 0,i = low;i<=high;i++,k++)

{

arr[i] = tmp[k];

}

free(tmp);

}

void MergeSort(int arr[],int low,int high)

{

int mid;

if(low<high)

{

mid = (low+high)/2;

MergeSort(arr,low,mid);

MergeSort(arr,mid+1,high);

Merge(arr,low,high,mid);

}

}

int main()

{

int n;

cin>>n;

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

{

cin>>num[i];

}

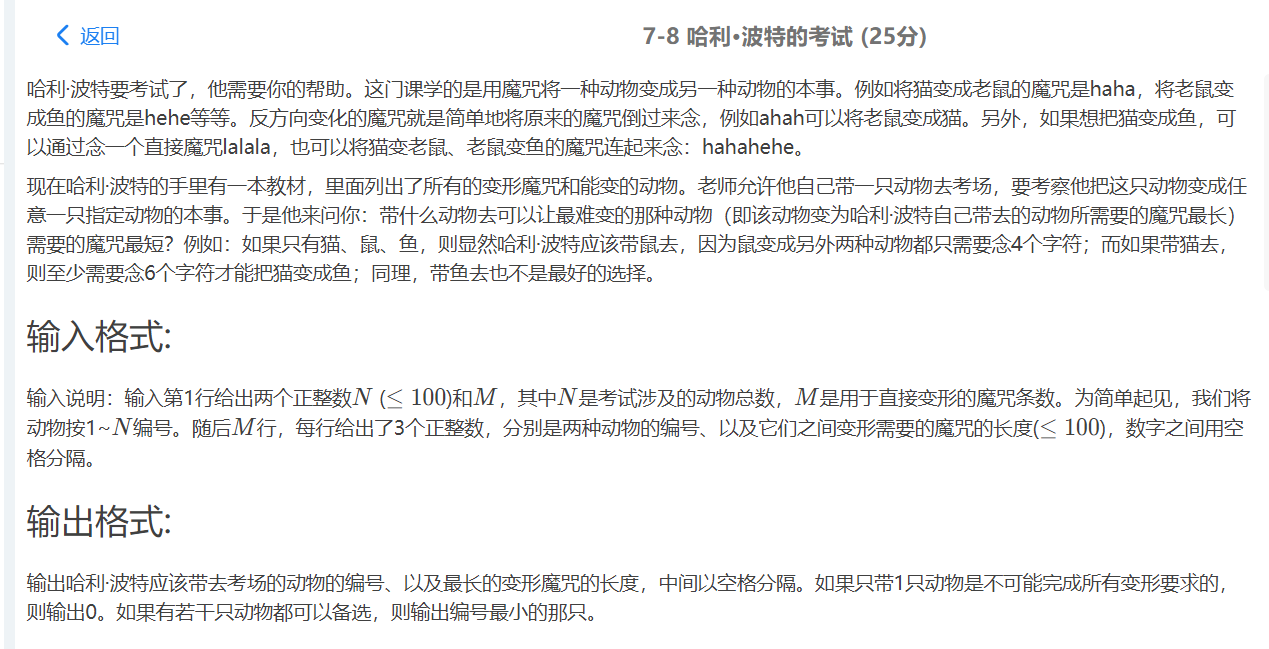
MergeSort(num,0,n-1);

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

cout<<num[i]<<((i==n-1)?(""):" ");

return 0;

}

#include<stdio.h>

#include<stdlib.h>

#define MAXVEX 105

#define INFINITY 65535

void CreateGraph( );

void Floyd();

void FindAnimal();

int FindMax( int i);

int G[MAXVEX][MAXVEX],Nv,Ne;

int D[MAXVEX][MAXVEX]; //存储最短路径矩阵

int main()

{

CreateGraph();

FindAnimal();

return 0;

}

void CreateGraph()

{

//用邻接矩阵表示图

int i,j;

int v1,v2,w;

scanf("%d %d",&Nv,&Ne);

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

{

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

{

if( i==j){

G[i][j] = 0;

}

else G[i][j] = INFINITY; //初始化

}

}

for( i=0; i<Ne; i++) //注意这里是读入边

{

scanf("%d %d %d",&v1,&v2,&w);

G[v1][v2] = w; //读入权值

G[v2][v1]= G[v1][v2]; //无向图对称

}

}

void FindAnimal()

{

int max,min;

int animal;

int i;

Floyd();

min = INFINITY;

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

{

//比较每行最大距离，寻找其中最小值

max = FindMax( i );

if( max == INFINITY)

{

//判断图是否连同通

printf("0\n");

return;

}

if( min>max )

{

min = max;

animal = i;

}

}

printf("%d %d\n",animal,min);

}

int FindMax( int i)

{

int max;

int j;

max = 0;

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

{

if( i!=j && D[i][j]>max)

{

max = D[i][j];

}

}

return max;

}

void Floyd()

{

int i,j,k;

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

{

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

{

D[i][j] = G[i][j];

}

}

//注意动物是从下标1开始编号

for ( k=1; k<=Nv; k++)

{

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

{

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

{

if( D[i][k]+D[k][j] < D[i][j])

{

D[i][j] = D[i][k]+D[k][j];

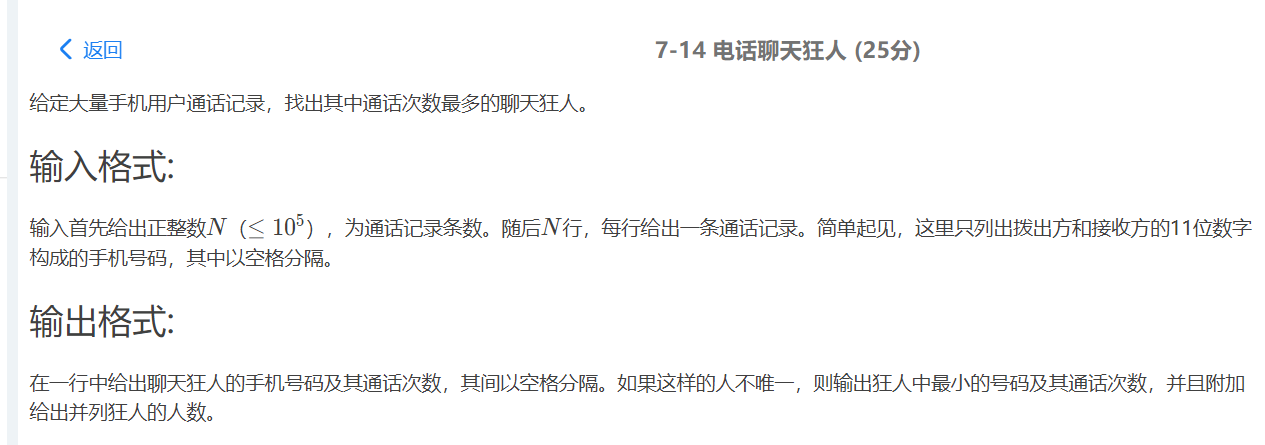
}

}

}

}

}



#include<iostream>

#include<map>

#include<string>

using namespace std;

int main()

{

map<string ,int >m;

string a;

int n,f=0,fuck=1;

cin>>n;

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

{

cin>>a;

m[a]++;

}

map<string,int>::iterator it;

for (it=m.begin();it!=m.end();it++)

{

if (it->second>f)

{

fuck=1;

f=it->second;

a=it->first;

}

else if (it->second==f)

{

fuck++;

if (it->first<a)

a=it->first;

}

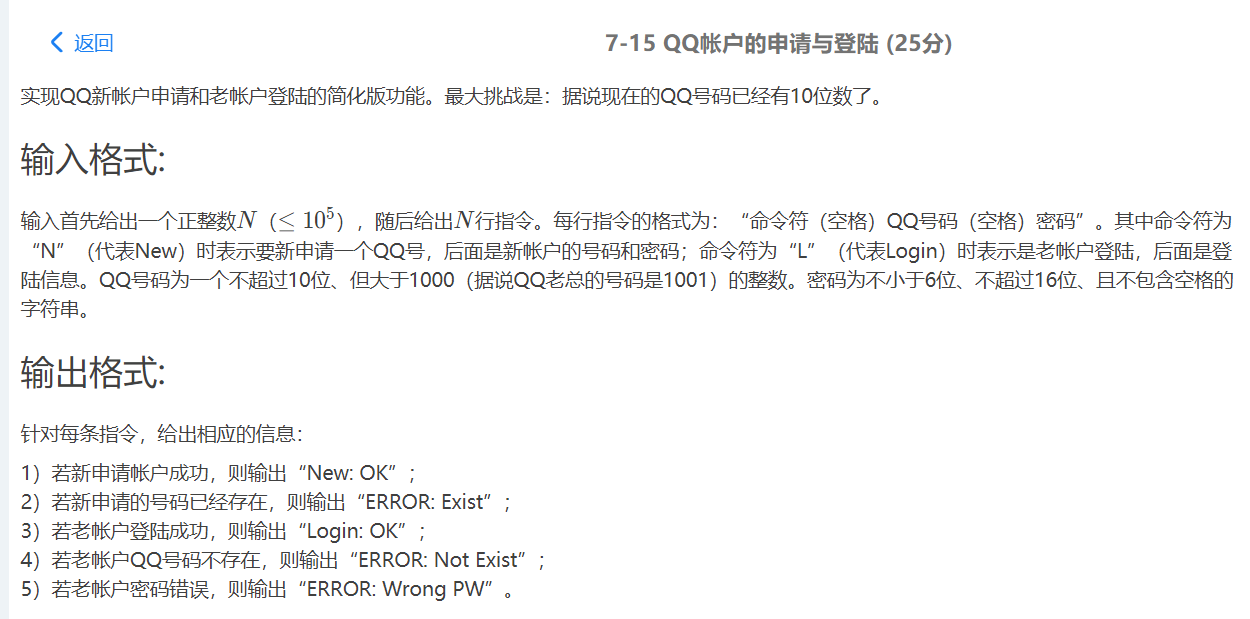
}

cout<<a<<" "<<f;

if (fuck!=1)

cout<<" "<<fuck;

}



//将QQ号的类型从数字型改为字符串型后，成功AC。

#include <iostream>

#include <string>

#include <cstring>

using namespace std;

long N;

typedef struct user

{

char num[20];

char pass[200];

struct user\* next;

}node;

typedef node\* node\_ptr;

node\_ptr\* createtable(unsigned int size)

{

node\_ptr\* T = new node\_ptr[N];

for (unsigned int i = 0; i < size; i++)

{

T[i] = new node;

T[i]->next = NULL;

}

return T;

}

typedef unsigned long long Index;

Index hash\_(char\* key, unsigned int tablesize)

{

Index hash\_val = 0;

while (NULL != \*(key+5))

{

hash\_val = (hash\_val << 5) + \*(key++);

}

return hash\_val % tablesize;

}

void apply(char\* num, char\* s2, node\_ptr\* T)

{

Index hash\_val = hash\_(num, N);

node\_ptr s = T[hash\_val];

while (NULL != s->next)

{

s = s->next;

if (!strcmp(s->num,num))

{

cout << "ERROR: Exist" << endl;

return;

}

}

if (NULL == s->next)

{

node\_ptr temp = new node;

strcat(temp->num,num);

strcpy(temp->pass, s2);

temp->next = NULL;

s->next = temp;

cout << "New: OK" << endl;

return;

}

}

void login(char\* num, char\* s2, node\_ptr\* T)

{

Index hash\_val = hash\_(num, N);

node\_ptr s = T[hash\_val];

while (NULL != s->next)

{

s = s->next;

if (!strcmp(s->num , num))

{

if (!strcmp(s->pass, s2))

{

cout << "Login: OK" << endl;

}

else

cout << "ERROR: Wrong PW" << endl;

return;

}

}

if (NULL == s->next)

{

cout << "ERROR: Not Exist" << endl;

return;

}

}

int main()

{

cin >> N;

char s1[20];

char str2[200];

node\_ptr\* T = createtable(N);

char order;

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

{

cin >> order;

if (order == 'N')

{

cin >> s1;

cin >> str2;

apply(s1, str2, T);

}

else if (order == 'L')

{

cin >> s1;

cin >> str2;

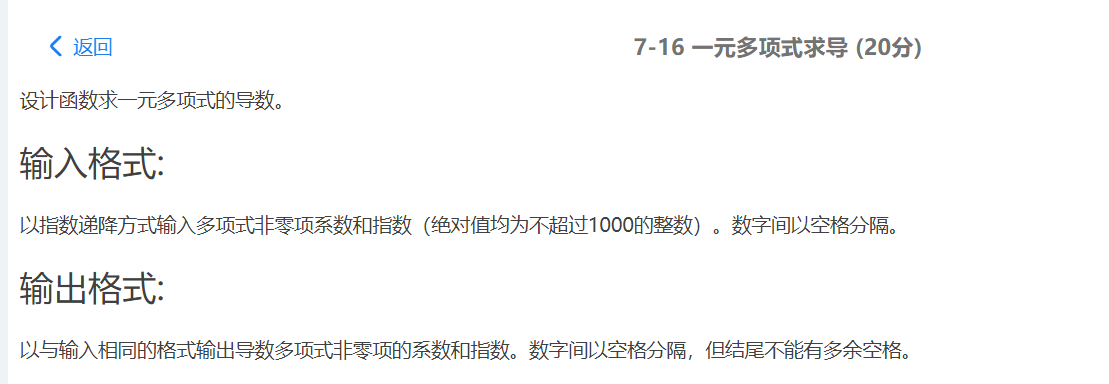
login(s1, str2, T);

}

}

return 0;

}



#include<iostream>

#include<cstdio>

using namespace std;

#define MAXN 1010

int num[MAXN] = {0};

int temp[MAXN] = {0}; //保存求导后的式子

int main()

{

int maxs = 0;

int a,b;

int flag = 0;

while(~scanf("%d %d",&a,&b))

{

if(b==0) continue;

if(a\*b!=0){

cout<<((flag>0)?" ":"")<<b\*a<<" "<<b-1;

flag++;

}

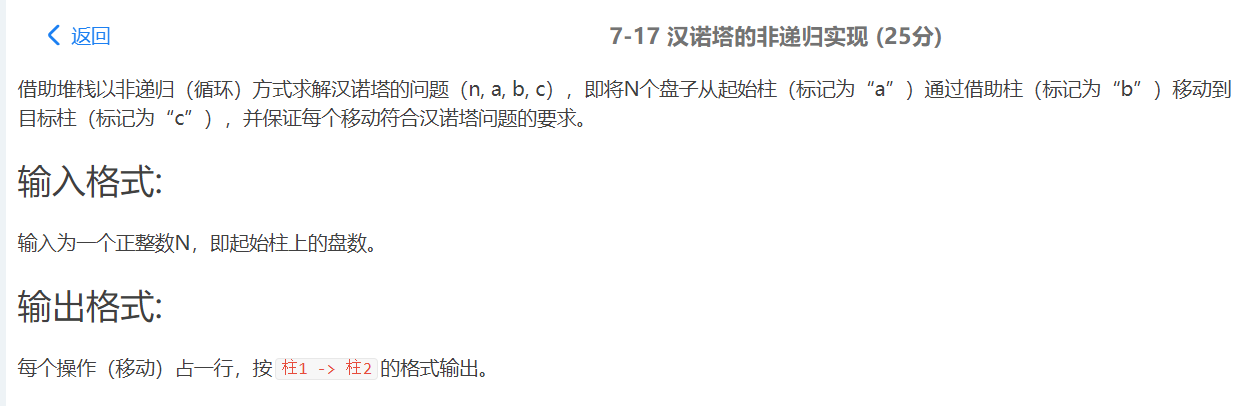
}

if(!flag) //坑！注意这里

cout<<"0 0"<<endl;

return 0;

}



//非递归AC代码

//用cout最后一个测试会超时，改为printf就AC了

#include <iostream>

#include <string>

#include <cstring>

#include <stdio.h>

#include <cmath>

using namespace std;

int n;//输入的盘子数

class stack\_//“三根柱子”的类型

{

public:

stack\_() :r(0){}

~stack\_() {}

void push(int k)

{

h[++r] = k;

}

int pop()

{

if (r <= 0)

return 0;

else

return h[r--];

}

public:

char name;

int r;//指针

int h[10000];

};

stack\_ S[3];//将三根柱子定义为数组，方便操作

bool is\_over(int cnt)//判断移动次数有没有超过2^n-1

{

if (cnt >= pow(2, n)-1)

return true;

else

return false;

}

void move(char a, char b, char c)//移动函数主体

{

int pop\_x,temp1,temp2;

int cnt = 0;//累计移动的次数

int i = 0;//循环的次数

while(cnt < pow(2,n)-1 )//移动次数到达最大时就要退出循环，继续循环会导致错误

{

int k;//中间变量，简化式子

pop\_x = S[i % 3].pop();//随着i的变化，可以实现对S[0],S[1],S[2]轮回判断

if (pop\_x == 1)

{

k = i + 1;

S[k % 3].push(1);

if (!is\_over(cnt))

{

//cout << S[i % 3].name << " -> " << S[k% 3].name << endl;

printf("%c -> %c\n", S[i % 3].name, S[k % 3].name);

cnt++;

}

temp1 = S[(k + 1) % 3].pop();

temp2 = S[(k - 1) % 3].pop();

if ((temp1 != 0 && temp2 != 0)&& (temp1 < temp2) || temp2 == 0 and temp1 != 0)//temp1 移动到 temp2 的情况

{

S[(k - 1) % 3].push(temp2);

S[(k - 1) % 3].push(temp1);

if (!is\_over(cnt))

{

//cout << S[(k + 1) % 3].name << " -> " << S[(k - 1) % 3].name <<endl;

printf("%c -> %c\n", S[(k+1) % 3].name, S[(k-1) % 3].name);

cnt++;

}

}

else if (temp1 == 0 && temp2 == 0)

{

//不移动任何盘子，只要把刚刚出栈的元素重新压回去

S[(k + 1) % 3].push(temp1);

S[(k - 1) % 3].push(temp2);

}

else

{

S[(k + 1) % 3].push(temp1);

S[(k + 1) % 3].push(temp2);

if (!is\_over(cnt))

{

//cout << S[(k - 1) % 3].name << " -> " << S[(k + 1) % 3].name << endl;

printf("%c -> %c\n", S[(k-1)% 3].name, S[(k+1) % 3].name);

cnt++;

}

}

i++;//注意在末尾将i的值加1，实现0，1，2的轮回

}

else

{

S[i % 3].push(pop\_x);//不符合条件，重新压回栈

i++;

}

}

//cout << endl << cnt << endl;

}

void hanoi(int n, char a, char b, char c)//接口

{

S[0].name = a,

S[1].name = b;

S[2].name = c;

for (int i = n; i >= 1; i--)//从大到小将盘子压入栈

{

S[0].push(i);

}

move(a, b, c);//调用move开始进行移动

}

int main()

{

cin >> n;

if (n % 2 == 0)

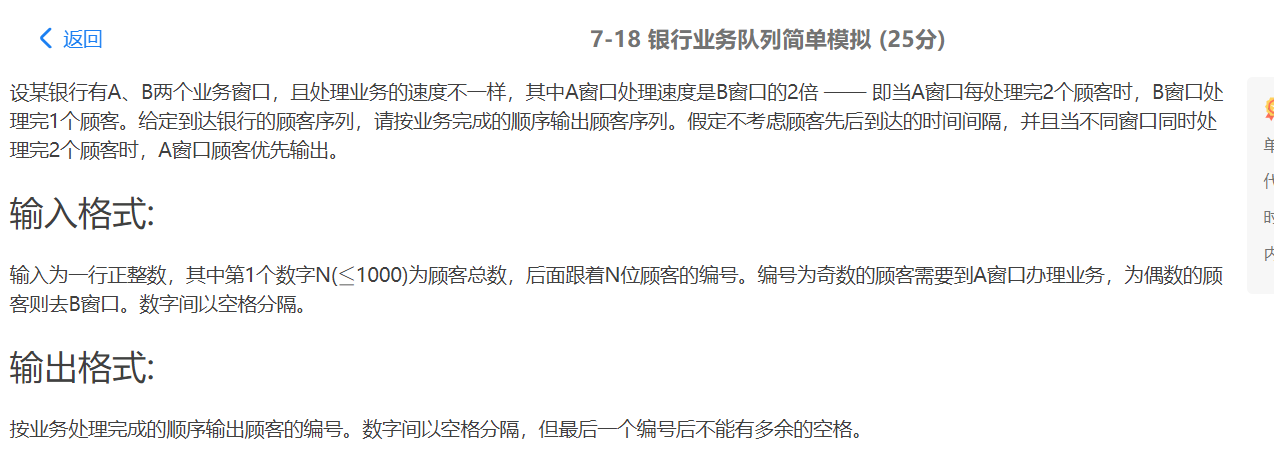
hanoi(n, 'a', 'b', 'c');//偶数的时候按abc顺序

else

hanoi(n, 'a', 'c', 'b');//奇数的时候按acb顺序

return 0 ;

}



#include<iostream>

#include<queue>

using namespace std;

int main()

{

queue<int> a;

queue<int> b;

int n;

cin>>n;

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

{

int num;

cin>>num;

if(num%2==0)

b.push(num);

else

a.push(num);

}

int flag=0;

while(a.size()!=0&&b.size()!=0)

{

cout<<(flag>0?" ":"")<<a.front();

flag++;

a.pop();

if(a.size()==0)

break;

else

{

cout<<" "<<a.front();

}

a.pop();

cout<<" "<<b.front();

b.pop();

}

//剩余全出队

while(a.size()!=0)

{

cout<<(flag>0?" ":"")<<a.front();

flag++;

a.pop();

}

while(b.size()!=0)

{

cout<<(flag>0?" ":"")<<b.front();

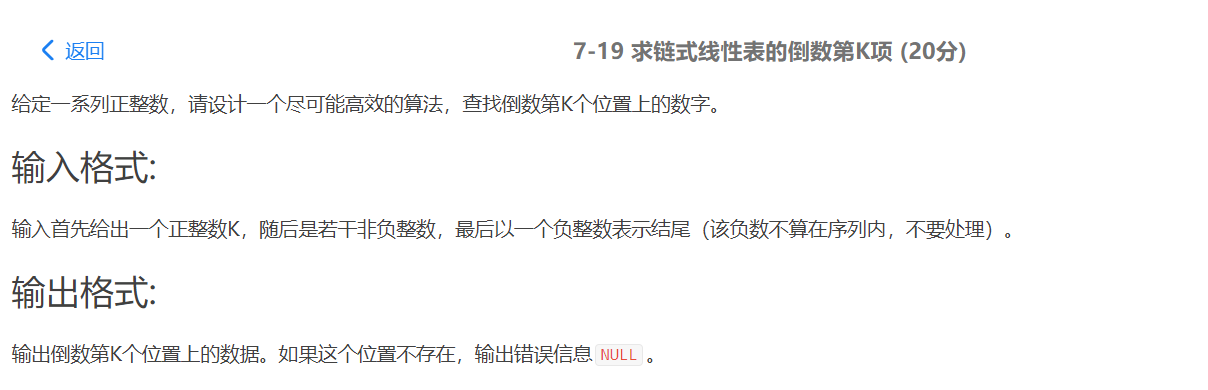
flag++;

b.pop();

}

return 0;

}



#include<stdio.h>

#include<malloc.h>

typedef struct Node{

int data;

Node \*next;

}Node,\*List;

int main()

{

int k;

scanf("%d",&k);

Node \*L,\*p;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

int a;

while(scanf("%d",&a)&&a>=0)

{

p = (Node \*)malloc(sizeof(Node)); // 生成新结点

p->data = a;

p->next = L->next; // 插入到表头

L->next = p;

}

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

{

L = L->next;

}

if(L)

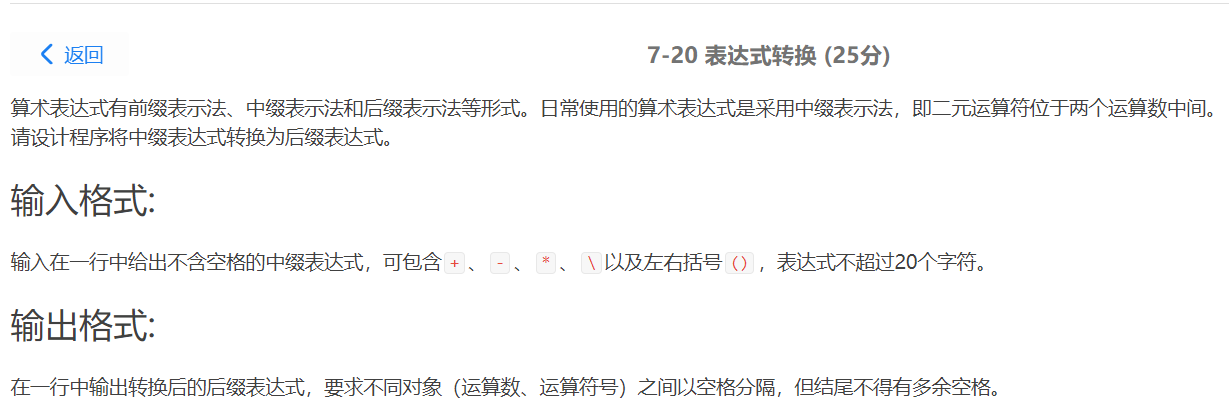
printf("%d\n",L->data);

else

printf("NULL\n");

return 0;

}



#include<stdio.h>

#include<malloc.h>

typedef struct Node{

int data;

Node \*next;

}Node,\*List;

int main()

{

int k;

scanf("%d",&k);

Node \*L,\*p;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

int a;

while(scanf("%d",&a)&&a>=0)

{

p = (Node \*)malloc(sizeof(Node)); // 生成新结点

p->data = a;

p->next = L->next; // 插入到表头

L->next = p;

}

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

{

L = L->next;

}

if(L)

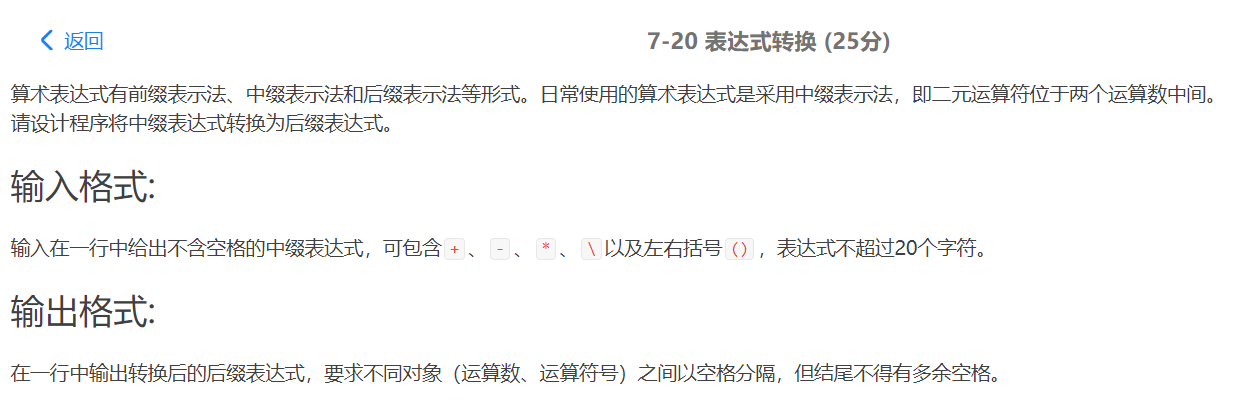
printf("%d\n",L->data);

else

printf("NULL\n");

return 0;

}



#include<stdio.h>

#include<malloc.h>

typedef struct Node{

int data;

Node \*next;

}Node,\*List;

int main()

{

int k;

scanf("%d",&k);

Node \*L,\*p;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

int a;

while(scanf("%d",&a)&&a>=0)

{

p = (Node \*)malloc(sizeof(Node)); // 生成新结点

p->data = a;

p->next = L->next; // 插入到表头

L->next = p;

}

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

{

L = L->next;

}

if(L)

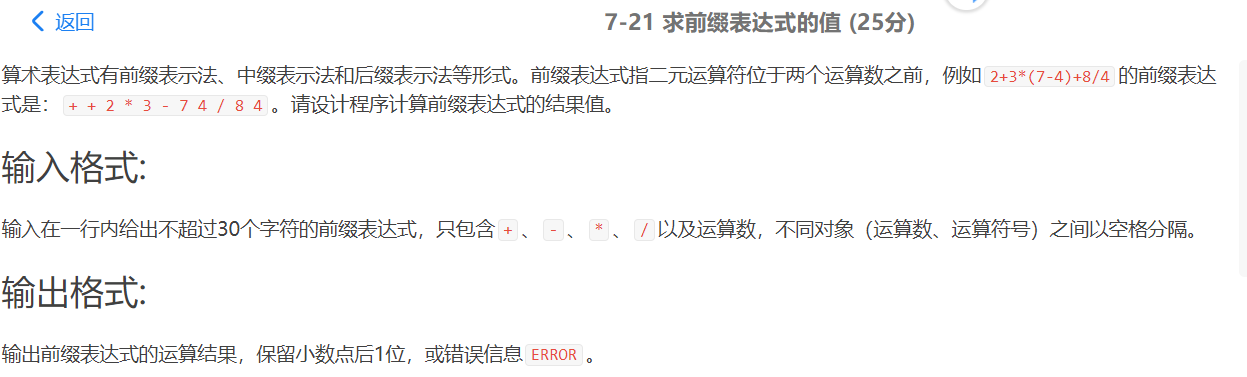
printf("%d\n",L->data);

else

printf("NULL\n");

return 0;

}



#include <iostream>

#include <cstdio>

#include <algorithm>

#include <stack>

#include <string>

#include <cctype>

using namespace std;

stack <double> st;

int main()

{

string s;

getline(cin, s);

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

{

if (isdigit(s[i]))

{

double mul = 10, num = s[i] - '0';

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

{

if (isdigit(s[i]))

{

num += (s[i] - '0') \* mul;

mul \*= 10;

}

else if (s[i] == '.')

{

num /= mul;

mul = 1;

}

else if (s[i] == '-')

num = -num;

else

break;

}

st.push(num);

}

else if (s[i] != ' ') //else

{

double a, b, sum;

a = st.top();

st.pop();

b = st.top();

st.pop();

switch (s[i])

{

case '+':

sum = a + b;

break;

case '-':

sum = a - b;

break;

case '\*':

sum = a \* b;

break;

case '/':

{

if (b == 0)

{

cout << "ERROR";

return 0;

}

sum = a / b;

}

}

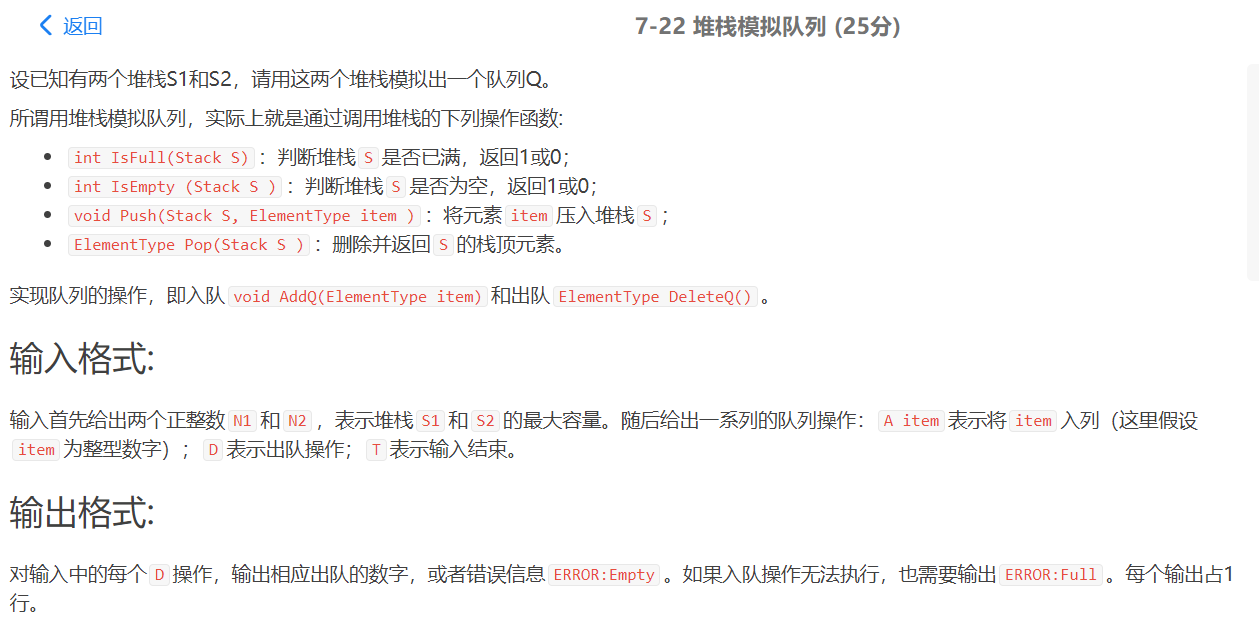
st.push(sum);

}

}

printf("%.1lf", st.top());

}



#include<iostream>

#include<stack>

using namespace std;

#define ElementType int

typedef struct Stack{

stack<int> s1; //输入栈（小）

stack<int> s2; //输出栈（大）

int size1;

int size2;

};

Stack Sta;

int IsFull(Stack S);

int IsEmpty (Stack S);

void Push(Stack &S, ElementType item); //注意这里必须得用引用

ElementType Pop(Stack &S); //注意这里必须得用引用

int main()

{

int size1,size2;

cin>>size1>>size2;

//初始化

if(size1>size2)

{

Sta.size1 = size2;

Sta.size2 = size1;

}

else

{

Sta.size1 = size1;

Sta.size2 = size2;

}

char order;

while(1)

{

char order;

cin>>order;

if(order=='T')

break;

if(order=='A')

{

int data;

cin>>data;

if(IsFull(Sta))

cout<<"ERROR:Full"<<endl;

else

Push(Sta,data);

}

if(order=='D')

{

if(IsEmpty(Sta))

cout<<"ERROR:Empty"<<endl;

else

cout<<Pop(Sta)<<endl;

}

}

}

int IsFull(Stack S)

{

if(S.s1.size()==S.size1&&S.s2.size()!=0)

return 1;

else return 0;

}

int IsEmpty (Stack S)

{

if(S.s1.empty()&&S.s2.empty())

return 1;

else return 0;

}

void Push(Stack &S, ElementType item)

{

if(S.s1.size()<S.size1)

S.s1.push(item);

else

{

for(int i = 0;i<S.size1;i++)

{

S.s2.push(S.s1.top());

S.s1.pop();

}

S.s1.push(item);

}

}

ElementType Pop(Stack &S)

{

if(!S.s2.empty())

{

int ele = S.s2.top();

S.s2.pop();

return ele;

}

else

{

for(int i = 0;i<S.size1;i++)

{

S.s2.push(S.s1.top());

S.s1.pop();

}

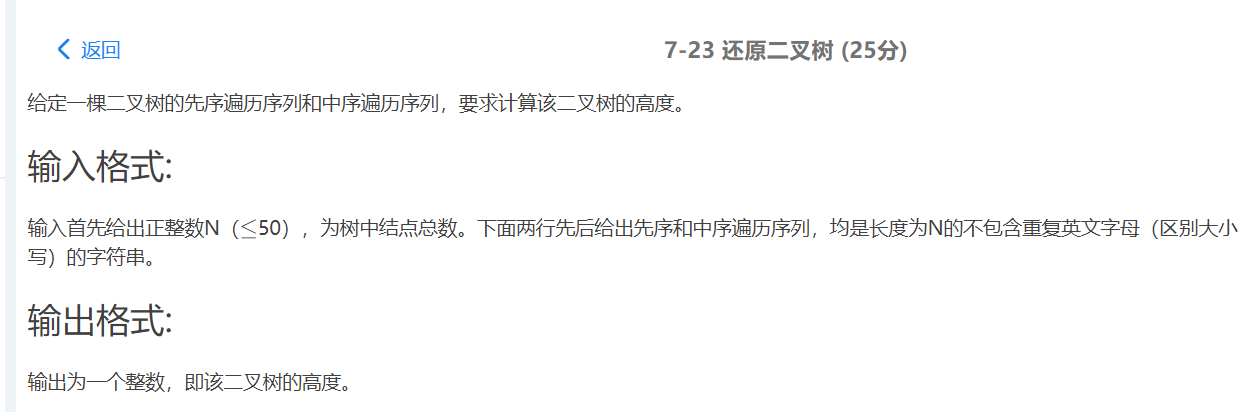
int ele = S.s2.top();

S.s2.pop();

return ele;

}

}



#include<bits/stdc++.h>

using namespace std;

int dfs(char a[],char b[],int n){

int i;

if(n==0)return 0;

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

if(b[i]==a[0]){

break;

}

}

int x=dfs(a+1,b,i)+1;//求左子树的深度

int y=dfs(a+i+1,b+i+1,n-i-1)+1;//求右子树的深度

return x>y?x:y;

}

int main()

{

char a[101];

char b[101];

int n;

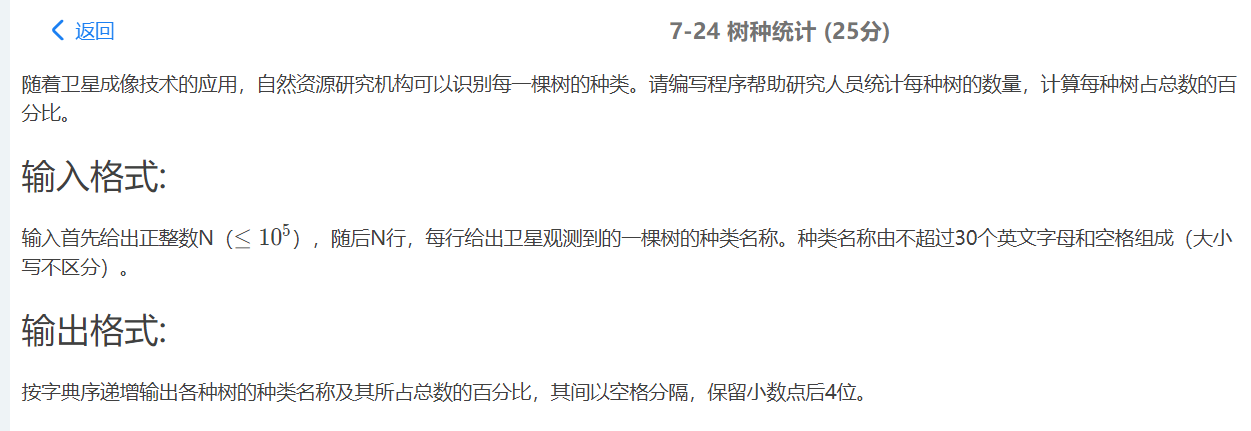
cin>>n;

cin>>a>>b;

int cnt=dfs(a,b,n);

cout<<cnt<<endl;

}



#include<iostream>

#include<map>

#include<string>

#include<algorithm>

#include<cstdio>

using namespace std;

int main()

{

map<string,int> tree;

int n;

cin>>n;

getchar(); //注意一定要用getchar()吸收回车

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

{

string s;

getline(cin,s);

if(tree.find(s)!=tree.end())

tree[s]++;

else

tree[s] = 1;

}

map<string,int>::iterator it;

for(it = tree.begin();it!=tree.end();it++)

{

cout<<(\*it).first<<" ";

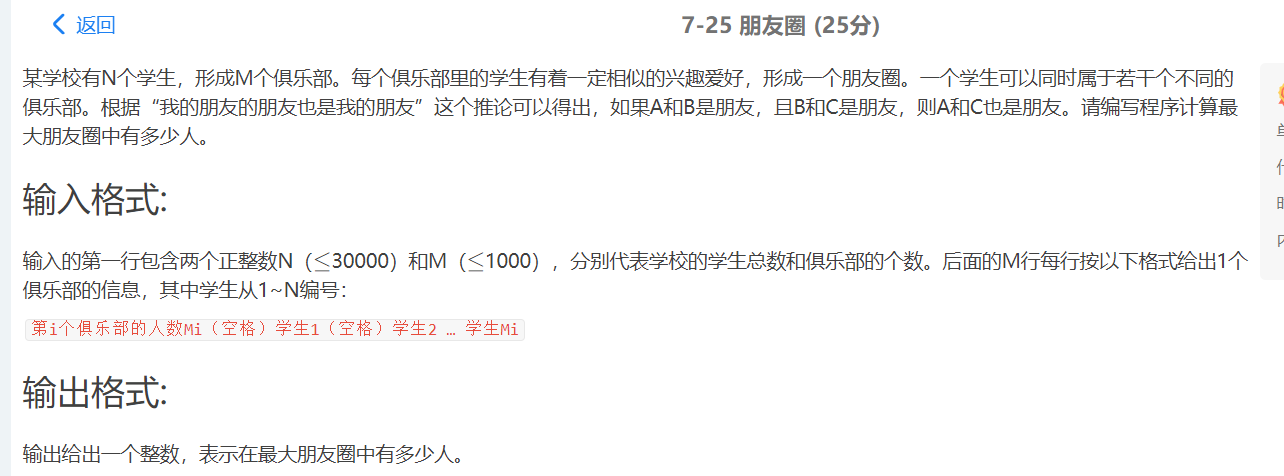
double cnt = (\*it).second\*1.0\*100/n;

printf("%.4lf%%\n",cnt);

}

return 0;

}



#include <iostream>

using namespace std;

int que[30000],book[30000];

void init()

{

for (int i = 1; i <= 30000; i++)

que[i] = i;

}

int getf(int k)

{

return que[k] == k ? k : que[k] = getf(que[k]);

}

void merge(int a, int b)

{

if (getf(a) != getf(b))

que[getf(a)] = getf(b);

}

int main()

{

int n, m, t, a, b, ans=0;

cin >> n >> m;

init();

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

{

cin >> t;

if (t == 0)

continue;

cin >> a;

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

{

cin >> b;

merge(a, b);

}

}

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

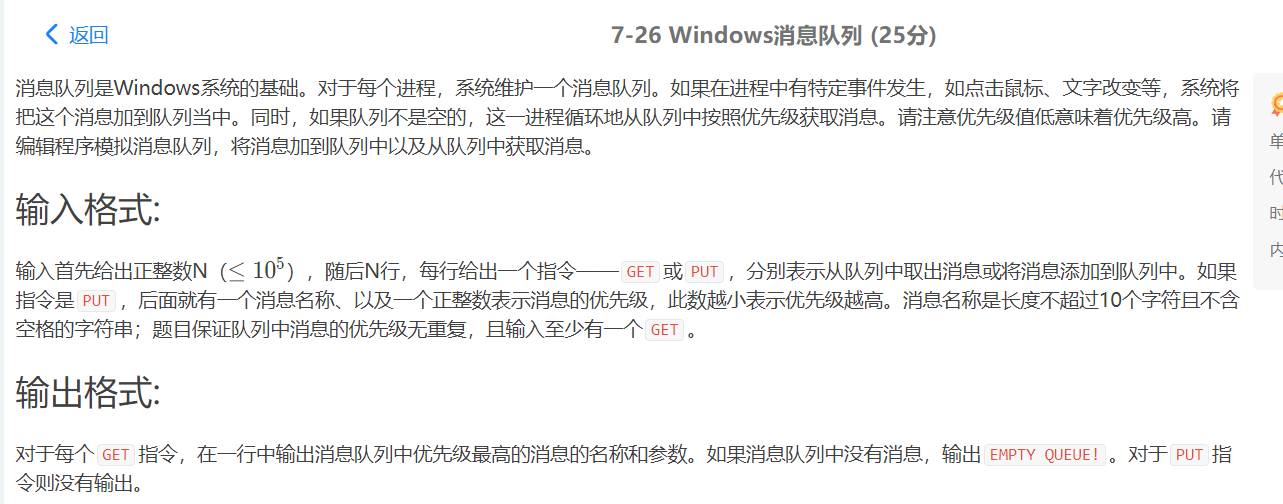
book[getf(i)]++;

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

ans = ans > book[i] ? ans : book[i];

cout << ans;

}



#include <cstdio>

#include <iostream>

#include <queue>

using namespace std;

int n;

char op[4];

struct mes {

char s[11];

int d;

bool operator <(const mes &a) const {

return d > a.d;

}

}temp;

int main() {

scanf("%d",&n);

priority\_queue<mes> q;

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

scanf("%s",op);

if(op[0] == 'P') {

scanf("%s%d",temp.s,&temp.d);

q.push(temp);

}

else {

if(q.empty()) printf("EMPTY QUEUE!\n");

else {

printf("%s\n",q.top().s);

q.pop();

}

}

}

}



#include<iostream>

#include<map>

#include<string>

#include<cstring>

using namespace std;

int N, M;

map<string, int> stringToInt; //名字编号，从0开始编号

char intToString[100][11]; //编号对应的名字，和stringToInt编号一致，也是从0开始编号

int spaces[100]; //每行对应的空格数，从0行开始计数

bool x\_child\_y(int x, int y);

bool x\_parent\_y(int x, int y);

bool x\_sibling\_y(int x, int y);

bool x\_descendant\_y(int x, int y);

bool x\_ancestor\_y(int x, int y);

int main() {

scanf("%d %d", &N, &M);

getchar();

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

char c;

int space = 0;

while((c = getchar()) == ' ') {

space++;

}

char name[11];

name[0] = c;

int index = 1;

while((c = getchar()) != '\n') {

name[index++] = c;

}

name[index] = '\0';

stringToInt[name] = i;

strcpy(intToString[i], name);

spaces[i] = space;

}

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

char str1[11], str2[3], str3[4], str4[11], str5[3], str6[11];

scanf("%s %s %s %s %s %s", str1, str2, str3, str4, str5, str6);

int line1 = stringToInt[str1];

int line2 = stringToInt[str6];

if(strcmp(str4, "child") == 0){

if(x\_child\_y(line1, line2)){

printf("True\n");

}else{

printf("False\n");

}

}else if(strcmp(str4, "parent") == 0){

if(x\_parent\_y(line1, line2)){

printf("True\n");

}else{

printf("False\n");

}

}else if(strcmp(str4, "sibling") == 0){

if(x\_sibling\_y(line1, line2)){

printf("True\n");

}else{

printf("False\n");

}

}else if(strcmp(str4, "descendant") == 0){

if(x\_descendant\_y(line1, line2)){

printf("True\n");

}else{

printf("False\n");

}

}else if(strcmp(str4, "ancestor") == 0){

if(x\_ancestor\_y(line1, line2)){

printf("True\n");

}else{

printf("False\n");

}

}

}

return 0;

}

bool x\_child\_y(int x, int y){

if(spaces[x] - 2 != spaces[y]){

return false;

}

if(x < y){

return false;

}

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

if(spaces[i] < spaces[x]){

return false;

}

}

return true;

}

bool x\_parent\_y(int x, int y){

if(spaces[x] + 2 != spaces[y]){

return false;

}

if(x > y){

return false;

}

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

if(spaces[i] < spaces[y]){

return false;

}

}

return true;

}

bool x\_sibling\_y(int x, int y){

if(spaces[x] != spaces[y]){

return false;

}

if(x > y){

swap(x, y);

}

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

if(spaces[i] < spaces[y]){

return false;

}

}

return true;

}

bool x\_descendant\_y(int x, int y){

if(spaces[x] < spaces[y]){

return false;

}

if(x < y){

return false;

}

int index = y + 1;

while(true){

if(spaces[index] == spaces[y]){

return false;

}

if(index == x){

return true;

}

index++;

}

}

bool x\_ancestor\_y(int x, int y){

if(spaces[x] > spaces[y]){

return false;

}

if(x > y){

return false;

}

int index = x + 1;

while(true){

if(spaces[index] == spaces[x]){

return false;

}

if(index == y){

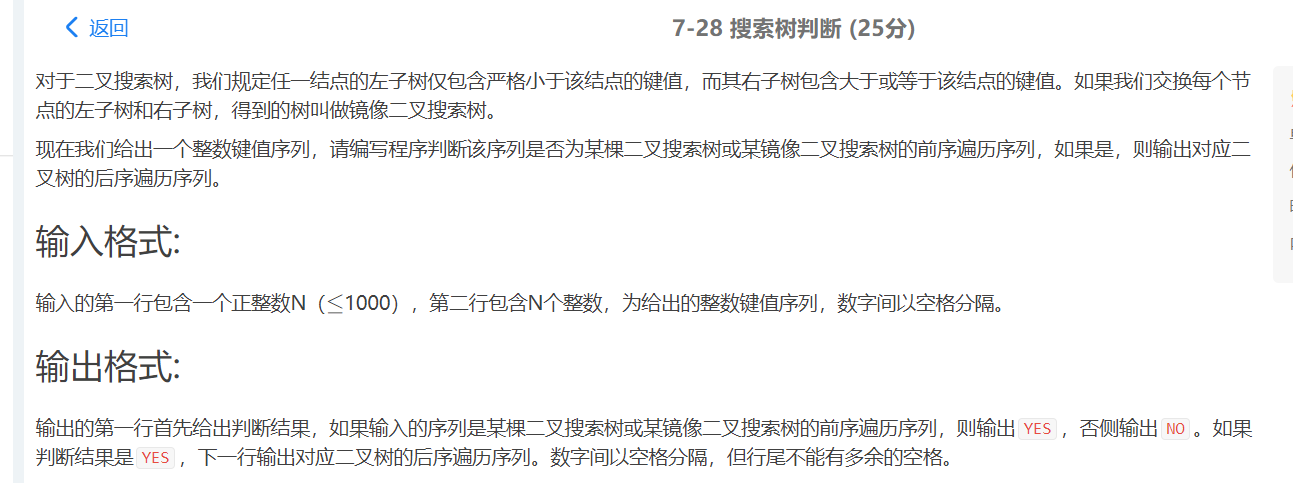
return true;

}

index++;

}

}



#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct TNode \*tree;

struct TNode

{

int data;

tree lchild;

tree rchild;

};

int flag = 0; //控制最后一个输出后面没有空格

int flag1 ; //如果不是二叉搜索树返回1

int flag2 ; //如果不是二叉镜像搜索树返回1

void Print( tree t);

tree Find ( int pre[],int len);

tree FindMirror( int pre[],int len);

int main()

{

int len;

int pre[1005];

int i;

tree t,tm;

scanf("%d",&len);

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

{

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

}

t = Find( pre,len);

tm = FindMirror( pre,len );

if( t && !flag1)

{

//树不为空并且是二叉搜索树

printf("YES\n");

Print( t );

printf("\n");

}

else if( tm && !flag2)

{

//树不为空并且是二叉镜像搜索树

printf("YES\n");

Print( tm );

printf("\n");

}

else printf("NO\n");

return 0;

}

tree Find ( int pre[],int len)

{

int i,j;

if( !len ) return NULL;

tree temp = (tree) malloc( sizeof( struct TNode));

temp->data = \*pre;

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

{

if( pre[i] >= temp->data)

//寻找右子树

break;

}

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

{

if( pre[j] < temp->data)

{

//右子树中有小于根结点的值，不是二叉搜索树

flag1 = 1;

return NULL;

}

}

temp->lchild = Find( pre+1, i-1);

temp->rchild = Find( pre+i, len-i);

return temp;

}

tree FindMirror( int pre[],int len)

{

//镜像树，左子树大于根大于右子树

int i,j;

if( !len ) return NULL;

tree temp = (tree) malloc( sizeof( struct TNode));

temp->data = \*pre;

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

{

if( pre[i] < temp->data)

//寻找右子树

break;

}

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

{

if( pre[j] >= temp->data)

{

//右子树中有大于等于根结点的值，不是二叉搜索树

flag2 = 1;

return NULL;

}

}

temp->lchild = FindMirror( pre+1, i-1);

temp->rchild = FindMirror( pre+i, len-i);

return temp;

}

void Print( tree t)

{

if( t )

{

//后序遍历

Print(t->lchild);

Print(t->rchild);

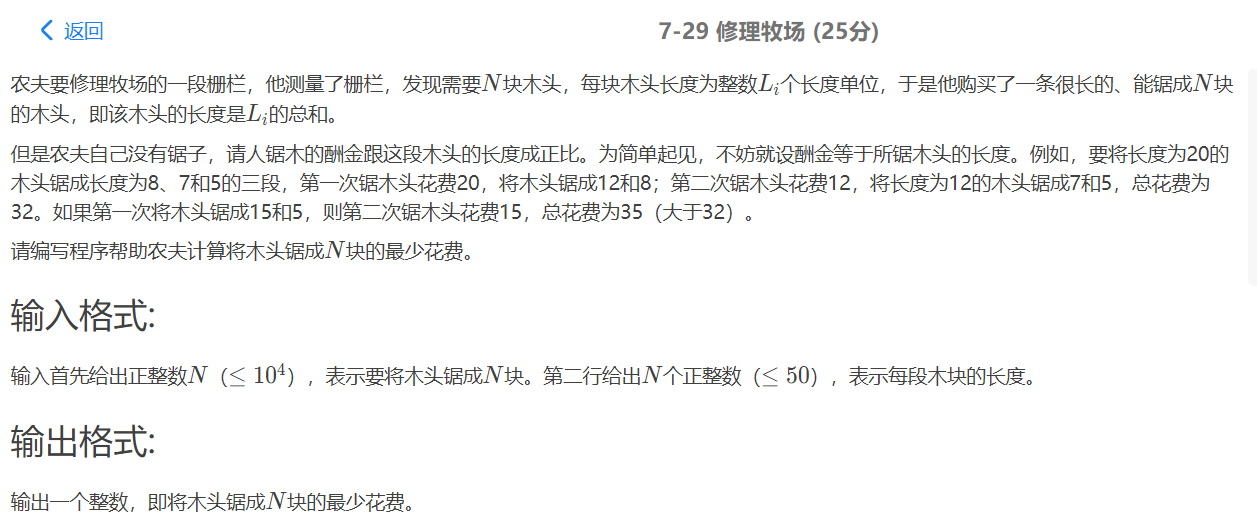
if( !flag ) flag = 1;

else printf(" ");

printf("%d",t->data);

}

}



#include<iostream>

#include<queue>

using namespace std;

priority\_queue<int,vector<int>,greater<int> >q;

int main()

{

int a,b,n,t,ans=0;

cin>>n;

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

{

cin>>t;

q.push(t);

}

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

{

a=q.top();

q.pop();

b=q.top();

q.pop();

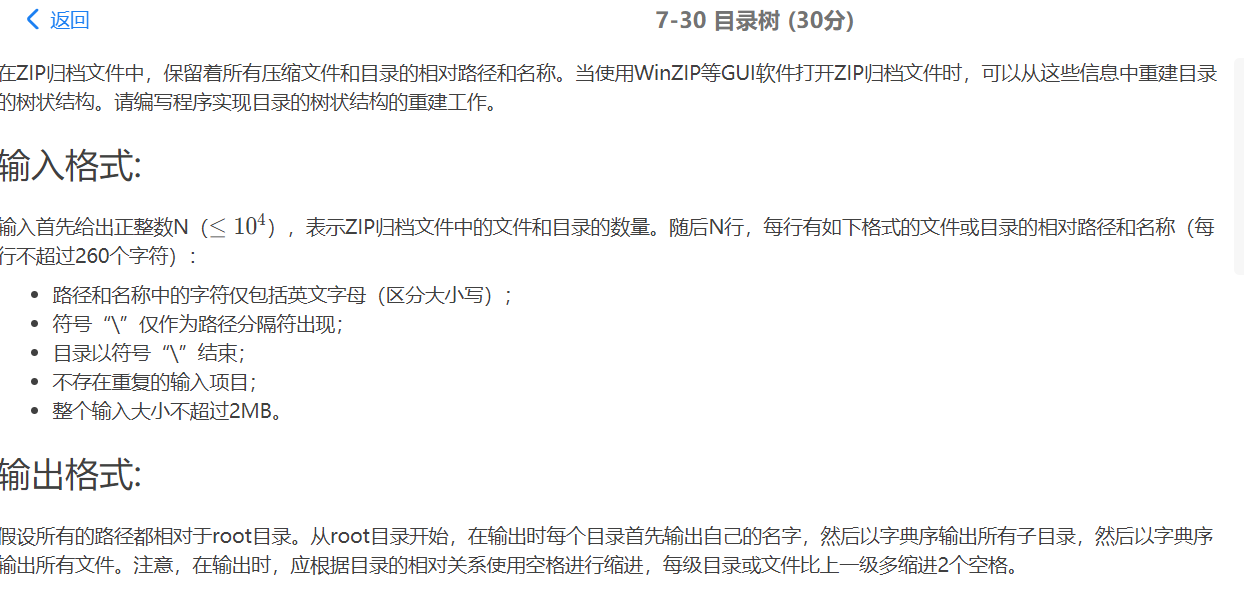
q.push(a+b);

ans+=a+b;

}

cout<<ans;

}



#include <stdio.h>

#include<stdlib.h>

#include <string.h>

#define True 1

#define False 0

typedef int Bool;

typedef struct node\* Node;

struct node {

char\*Name;

//先判断是不是目录,是目录才有File和Catalog，否则只可能有Brother

Bool isCatalog;

//指示本目录的子目录

Node File;

//指示本目录的子文件

Node Catalog;

//指示和本目录或文件平行的目录或文件

Node Brother;

} Head;

void Print(Node, int);

void Read();

Node New(char\*);

Node InsertCatalog(Node, char\*);

Node InsertFile(Node, char\*);

int main() {

int n;

scanf("%d", &n);

Head.Name = (char\*) malloc(sizeof(char) \* 5);

strcpy(Head.Name, "root");

Head.File = NULL;

Head.Catalog = NULL;

Head.Brother = NULL;

Head.isCatalog = True;

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

getchar();

Read();

}

Print(&Head, 0);

return 0;

}

void Read() {

char FileName[261];

Node HTemp = &Head;

scanf("%s", FileName);

char words[261];

//L记录的是目录(或文件名)名的第一个字符的起点位置

int j, L = 0;

for (int i = 0; i < strlen(FileName); i++) {

//第一个'\'字符前的名字

if (FileName[i] == '\\') {

for (j = L; j < i; j++) {

//获取名

words[j - L] = FileName[j];

}

words[j - L] = '\0';

//插入目录(有'\'的前面一定是目录名，没有就不是目录名，而是文件名)

HTemp->Catalog = InsertCatalog(HTemp->Catalog, words);

//一个'\'就是一个目录的结束，执行完以后还要再继续读取下一个目录，而下一个目录是建立在上一个目录

//所以这里要进行跳跃

HTemp = HTemp->Catalog;

//上面我们知道，此刻我们到达了下一个要插入的目录(或文件)的上一级目录

//但是，我们这个时候到达的是字典顺序排好的第一个目录，，，不一定是上面我们刚刚插入的目录

//所以这里要加个循环来寻找到达的插入的目录

while (strcmp(HTemp->Name, words))

HTemp = HTemp->Brother;

L = i + 1;

}

}

//显然，如果L==strlen(FileName)那么目录结构输入时应该是:XXX\XXX\XXX\ 最后一个一定是'\'字符，最后插入的是目录名

//但如果L<strlen(FileName)那么目录结构输入时应该是:XXX\XXX\xxx 最后一个不是'\'字符，最后插入的是文件名

if (L < strlen(FileName)) {

for (j = L; j <= strlen(FileName); j++) {

words[j - L] = FileName[j];

}

HTemp->File = InsertFile(HTemp->File, words);

}

}

Node InsertCatalog(Node Catalog, char\*InsertCatalogName) {

//如果文件目录不存在或者字典要插入的目录名字典顺序比当前第一个小

//那就应该插在第一个位置

if (!Catalog || strcmp(Catalog->Name, InsertCatalogName) > 0) {

Node temp = New(InsertCatalogName);

temp->Brother = Catalog;

return temp;

}

//如果目录已经存在直接返回

if (strcmp(Catalog->Name, InsertCatalogName) == 0)

return Catalog;

//如果不满足上面的条件，则进入与H目录同层的目录进行判断

//显然，H的同层下(H层以上的目录不在范围内容)目录结构发生改变，则全局就将发视变化

Catalog->Brother = InsertCatalog(Catalog->Brother, InsertCatalogName);

return Catalog;

}

Node InsertFile(Node File, char\*InsertFileName) {

//同上InsertCatalog

if (!File || strcmp(File->Name, InsertFileName) > 0) {

Node Insert = New(InsertFileName);

Insert->isCatalog = False;

Insert->Brother = File;

return Insert;

}

//这里我其实觉得也应该添加一个判断，毕竟也有可能出现重复文件名

// if (strcmp(File->Name, InsertFileName) == 0)

// return File;

//但仔细一思考，该题是目录清单，清单所确定的目录下不可能出现同一文件，，，至于为啥可能出现同一文件夹名字呢？

//这是因为目录清单每个条例只能确定一个文件路径或者文件夹路径，但同一文件夹下可能有多个文件或者多个目录，所以可能出现同一文件夹名字

File->Brother = InsertFile(File->Brother, InsertFileName);

return File;

}

Node New(char\*Name) {

Node newNode = (Node) malloc(sizeof(struct node));

newNode->Name = (char\*) malloc(sizeof(char) \* (strlen(Name) + 1));

strcpy(newNode->Name, Name);

newNode->Brother = NULL;

newNode->File = NULL;

newNode->Catalog = NULL;

//默认是在建目录

newNode->isCatalog = True;

return newNode;

}

void Print(Node H, int Space) {

if (H) {

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

printf(" ");

//打印顺序

//1-1 根目录名

// 1-2 子目录名

// 1-3 文件名

//1-4 异根目录名

//1-1

printf("%s\n", H->Name);

//1-2

if (H->isCatalog == 1)

Print(H->Catalog, Space + 2);

//1-3

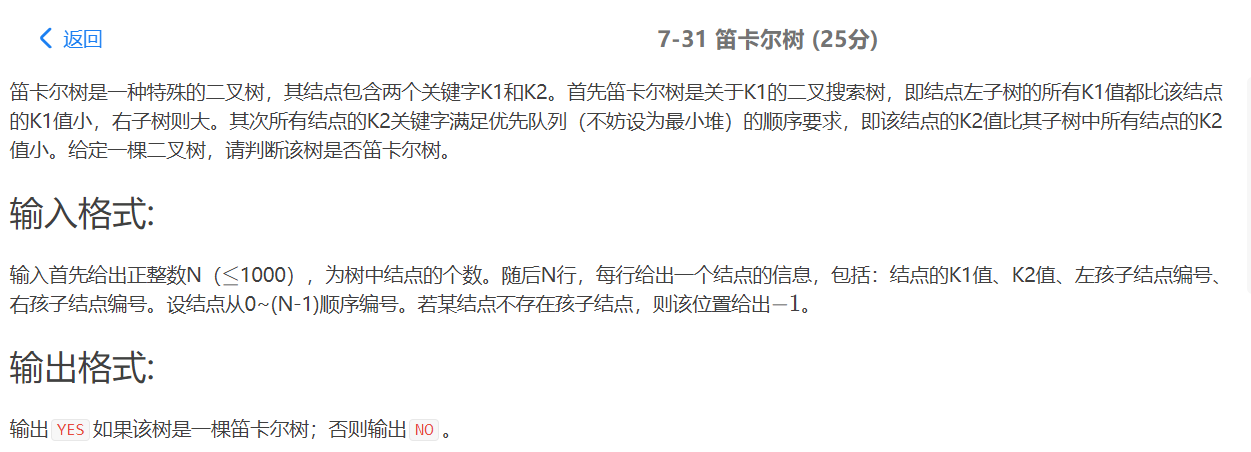
Print(H->File, Space + 2);

//1-4

Print(H->Brother, Space);

}

}



#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef struct N{

int K1;

int K2;

int Left;

int Right;

} Node;

typedef Node\* TreeNode;

void Ergodic(TreeNode tNode[],int pos,int parentK1);

void MidErgodic(TreeNode tNode[],int pos);

int SearchTree[1000]={0};

int Spos=0;

int main(){

int n;

scanf("%d",&n);

TreeNode \*tNode=(TreeNode\*)malloc(sizeof(TreeNode)\*n);

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

tNode[i]=(TreeNode)malloc(sizeof(Node));

tNode[i]->K1=0;

tNode[i]->K2=0;

tNode[i]->Left=-1;

tNode[i]->Right=-1;

}

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

scanf("%d %d %d %d",&tNode[i]->K1,&tNode[i]->K2,&tNode[i]->Left,&tNode[i]->Right);

}

int \*isV=(int \*)malloc(sizeof(int)\*n);

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

isV[i]=0;

}

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

if(tNode[i]->Left!=-1){

isV[tNode[i]->Left]=1;

}

if(tNode[i]->Right!=-1){

isV[tNode[i]->Right]=1;

}

}

int First=0;

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

if(!isV[i]){

First=i;

break;

}

}

Ergodic(tNode,First,10000);

MidErgodic(tNode,First);

int flag=1;

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

if(SearchTree[i]<SearchTree[i-1]){

printf("NO\n");

flag=0;

break;

}

}

if(flag)

printf("YES\n");

return 0;

}

void Ergodic(TreeNode tNode[],int pos,int parentK1){

TreeNode root=tNode[pos];

TreeNode Left=NULL;

if(tNode[pos]->Left!=-1){

Left=tNode[tNode[pos]->Left];

}

TreeNode Right=NULL;

if(tNode[pos]->Right!=-1){

Right=tNode[tNode[pos]->Right];

}

if(Left!=NULL && Right!=NULL){

if((root->K1 > Left->K1 && root->K2 < Left->K2)

&& (root->K1 < Right->K1 && root->K2 < Right->K2)

){

Ergodic(tNode,root->Left,root->K1);

Ergodic(tNode,root->Right,root->K1);

}else{

printf("NO\n");

exit(0);

}

}else if(Left!=NULL && Right==NULL){

if(root->K1 > Left->K1 && root->K2 < Left->K2){

Ergodic(tNode,root->Left,root->K1);

}else{

printf("NO\n");

exit(0);

}

}else if(Left==NULL && Right != NULL){

if(root->K1 < Right->K1 && root->K2 < Right->K2){

Ergodic(tNode,root->Right,root->K1);

}else{

printf("NO\n");

exit(0);

}

}

}

void MidErgodic(TreeNode tNode[],int pos){

if(pos!=-1){

TreeNode root=tNode[pos];

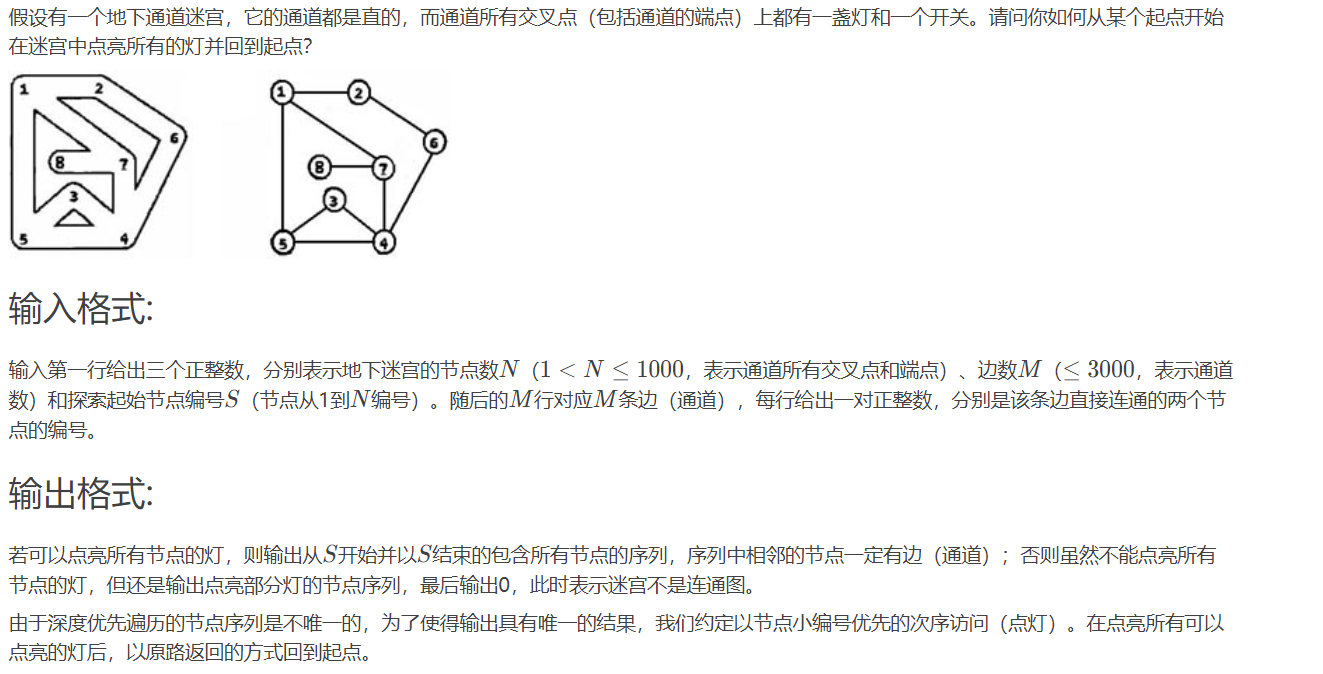
MidErgodic(tNode,root->Left);

SearchTree[Spos++]=root->K1;

MidErgodic(tNode,root->Right);

}

}



#include<iostream>

using namespace std;

#define MAXN 1010

int AdjMatrix[MAXN][MAXN] = {0};

int visited[MAXN] = {0};

int flag = 0;

int cnt = 0;

int n;

void DFS(int s)

{

if(flag)

cout<<" ";

flag++;

cout<<s;

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

if(!visited[i]&&AdjMatrix[s][i])

{

visited[i] = 1;

cnt++;

DFS(i);

cout<<" "<<s;

}

}

int main()

{

int m,s;

cin>>n>>m>>s;

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

{

int a,b;

cin>>a>>b;

AdjMatrix[a][b] = AdjMatrix[b][a] = 1;

}

visited[s] = 1;

cnt++;

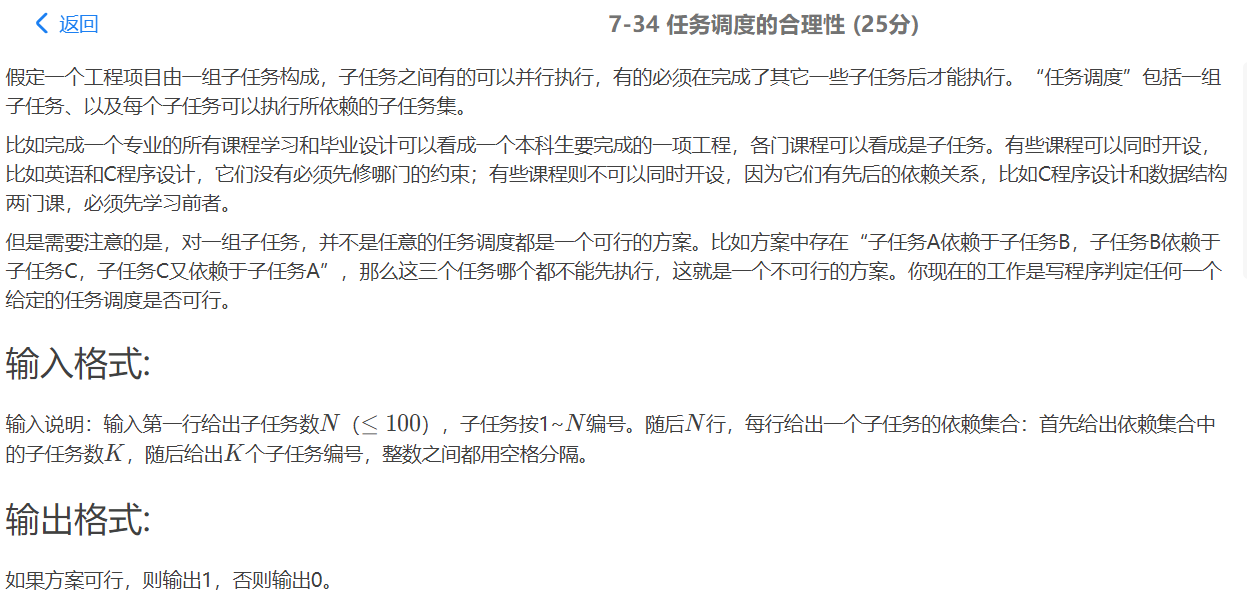
DFS(s);

if(cnt<n)

cout<<" 0";

return 0;

}



#include<iostream>

using namespace std;

int G[101][101], Indegree[101] = {0};

//G[i][j]为1表示j依赖于i,Indegree[i]表示i依赖的任务数，即G中第i列1的个数

int main()

{

int N, K;

cin>>N;

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

cin>>K;

Indegree[i] = K;//入度为依赖集合中子任务个数

int ID;

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

cin>>ID;

G[ID][i] = 1;//i依赖于ID，即ID完成才能执行i

}

}

int flag = 0, V;//flag为0表示未找到目前可执行的任务，V记录入度为0的任务编号

for(int i=1;i<=N;i++){//每次找到一个任务执行

flag = 0;

for(V=1;V<=N;V++)//找到编号最小的入度为0的顶点

if(Indegree[V]==0){

Indegree[V] = -1;//表示已执行，下次不会再执行

flag = 1;//找到入度为0的顶点

break;//结束查找

}

if(flag==0) break;//没有入度为0的顶点表明任务安排不合理，退出循环

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

if(G[V][j]==1) Indegree[j]--;//对每个依赖V的子任务，因为V已执行，其入度-1

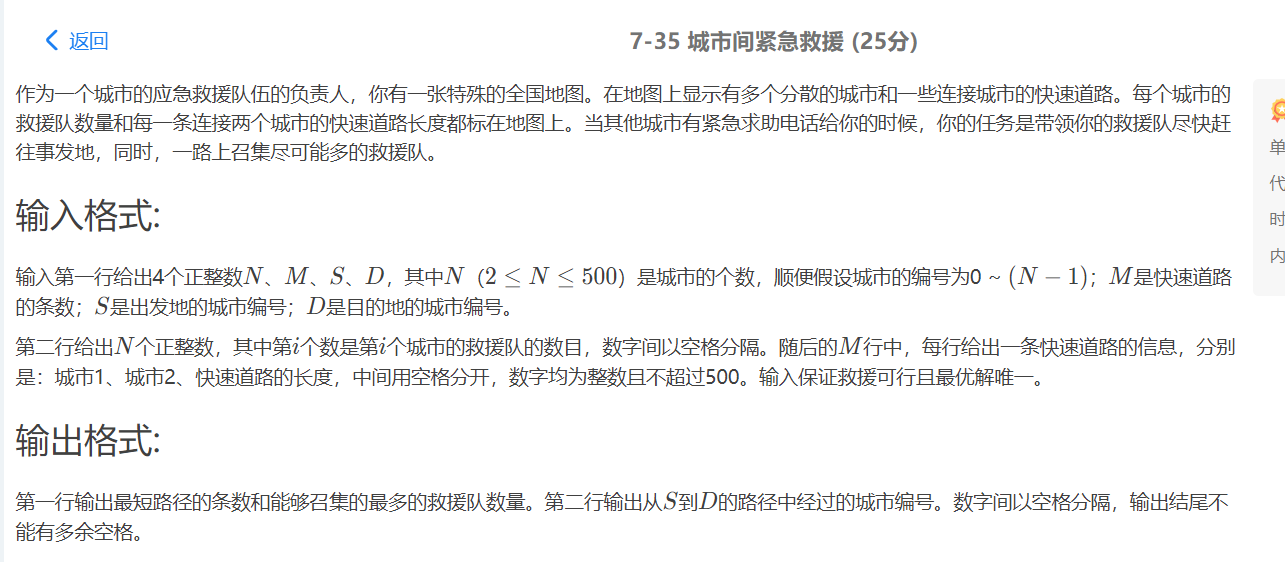
}

if(flag) cout<<"1"<<endl;

else cout<<0<<endl;

return 0;

}



//好好研究一下dijstra算法

//这个最短路径条数是指能到达目的地的不同最短路径的数量，而不是最优解中通过几条路径到达目的地

#include<stdio.h>

#include<vector>

#include<stack>

using namespace std;

struct line{

int next;

int dis;

};

vector<line> edge[501];

stack<int> s;

int people[501];int path[501];int dis[501];bool mark[501];

int count[501];

int com[501];//表示集结起来的人数

int main(){

int N,M,S,D;

scanf("%d%d%d%d",&N,&M,&S,&D);

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

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

com[i]=people[i];

}

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

int a,b,d;

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

line tmp;

tmp.next=b;tmp.dis=d;

edge[a].push\_back(tmp);

tmp.next=a;

edge[b].push\_back(tmp);

}

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

mark[i]=false;

dis[i]=-1;

}

dis[S]=0;

count[S]=1;

mark[S]=true;

int newp=S;

for(int i=0;i<N;i++){//循环n-1次 ，选取剩余n-1个结点加入集合

for(int j=0;j<edge[newp].size();j++){//根据每次新加入的结点更新距离信息

int t=edge[newp][j].next;

int d=edge[newp][j].dis;

if(dis[t]==-1||dis[t]>dis[newp]+d)

count[t]=count[newp];

if(dis[t]==dis[newp]+d)

count[t]+=count[newp];

if(mark[t]==true) continue;

if(dis[t]==-1||dis[t]>dis[newp]+d||(dis[t]==dis[newp]+d)&&com[t]<com[newp]+people[t]){

dis[t]=dis[newp]+d;

com[t]=com[newp]+people[t];

path[t]=newp;//存储前一结点信息

// printf("%d %d \n",t,path[t]);

}

}

int mind=123123123;int maxp=0;

for(int j=0;j<N;j++){//选取未加入集合的结点中最短路径||相同路径最大人数作为下一个加入集合的结点

if(mark[j]==true) continue;

if(dis[j]==-1) continue;

if(dis[j]<mind||(dis[j]==mind&&com[j]>maxp)){

mind=dis[j];

maxp=com[j];

newp=j;

}

}

mark[newp]=true;

}

// printf("%d ",dis[D]);

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

// printf("%d ",path[i]);

// }

// printf("\n");

int tmp=D;s.push(D);

while(tmp!=S){

tmp=path[tmp];

s.push(tmp);

}

printf("%d %d\n",count[D],com[D]);

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

// printf("%d ",count[i]);

while(s.size()>1){

printf("%d ",s.top());

s.pop();

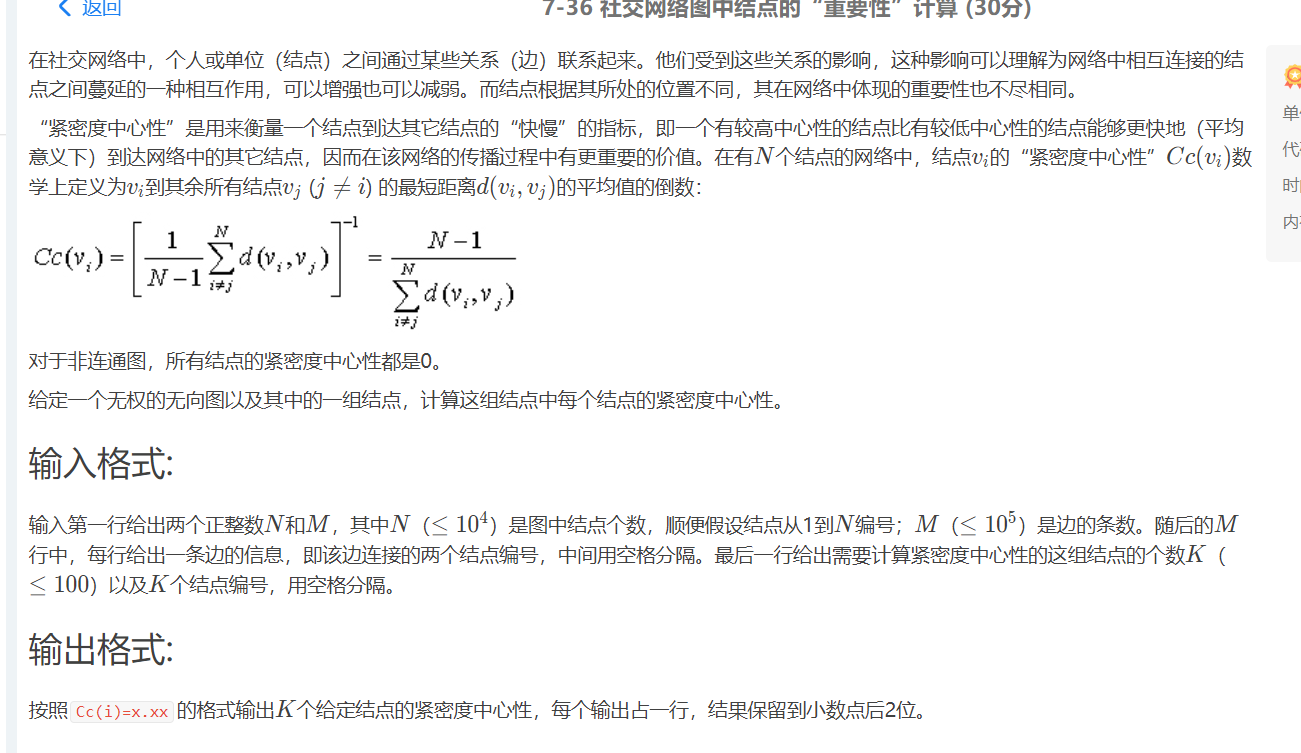
}

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

s.pop();

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#define Max 10000

#define Inf 1E5

typedef struct GNode\* Graph;

struct GNode{

int V;

int L;

int GR[Max][Max];

};

Graph CreateGraph(int V);

Graph InsertEdge(Graph G,int N);

void Cc(Graph G,int n);

int main(){

int N,M,K,i,num;

scanf("%d",&N);

Graph G=CreateGraph(N);

scanf("%d",&M);

G=InsertEdge(G,M);

scanf("%d",&K);

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

scanf("%d",&num);

Cc(G,num);

}

return 0;

}

Graph CreateGraph(int V){

Graph G=(Graph)malloc(sizeof(struct GNode));

G->V=V;

int i,j;

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

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

G->GR[i][j]=0;

return G;

}

Graph InsertEdge(Graph G,int N){

int i,v1,v2,w;

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

scanf("%d %d",&v1,&v2);

G->GR[v1-1][v2-1]=1;

G->GR[v2-1][v1-1]=1;

}

return G;

}

void Cc(Graph G,int n){

n--;

int i;

int path=0;

int dis[Max];

int a[Max];

int k;

int head=0;

int tail=0;

for(i=0;i<G->V;i++){

dis[i]=Inf;

}

a[head]=n;

dis[n]=0;

while(head<=tail){

k=a[head];

head++;

for(i=0;i<G->V;i++){

if(G->GR[k][i]&&dis[i]>dis[k]+1){

dis[i]=dis[k]+1;

tail++;

a[tail]=i;

}

}

}

for(i=0;i<G->V;i++){

if(dis[i]!=Inf)

path+=dis[i];

else{

path=Inf;

break;

}

}

if(path!=Inf){

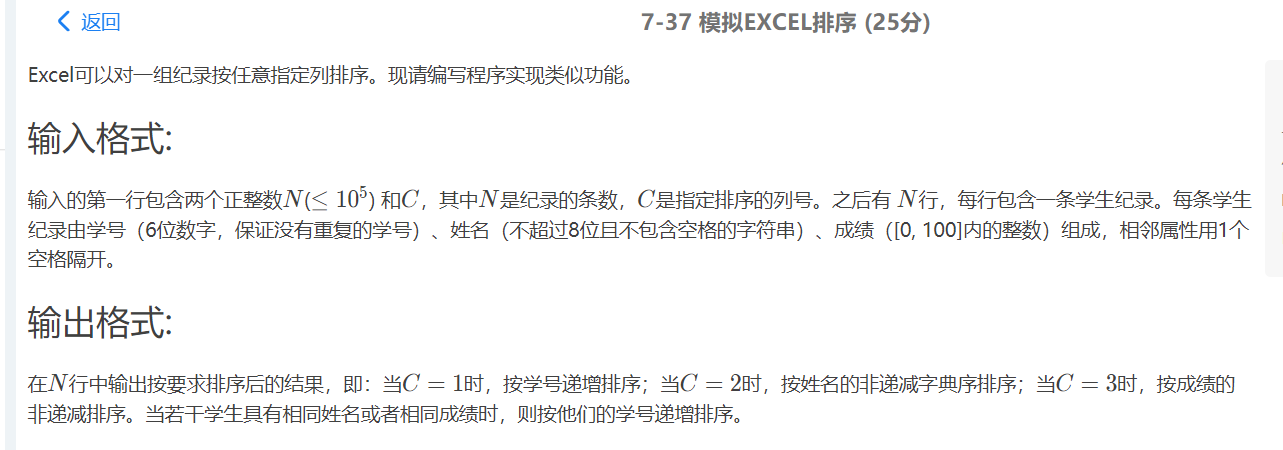
printf("Cc(%d)=%.2f\n",n+1,(G->V-1)\*1.0/path);

}

else

printf("Cc(%d)=0.00\n",n+1);

}



#include<iostream>

#include<vector>

#include<set>

#include<map>

#include<algorithm>

using namespace std;

#define MAXN 100010

typedef struct Stu{

string stuNum;

string name;

int grade;

};

struct Stu student[MAXN];

bool cmp1(Stu a,Stu b)

{

return a.stuNum<b.stuNum;

}

bool cmp2(Stu a,Stu b)

{

if(a.name == b.name)

return a.stuNum<b.stuNum;

else return a.name<b.name;

}

bool cmp3(Stu a,Stu b)

{

if(a.grade == b.grade)

return a.stuNum<b.stuNum;

else return a.grade<b.grade;

}

int main()

{

int n,a;

cin>>n>>a;

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

{

cin>>student[i].stuNum>>student[i].name>>student[i].grade;

}

if(a==1)

sort(student,student+n,cmp1);

else if(a==2)

sort(student,student+n,cmp2);

else

sort(student,student+n,cmp3);

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

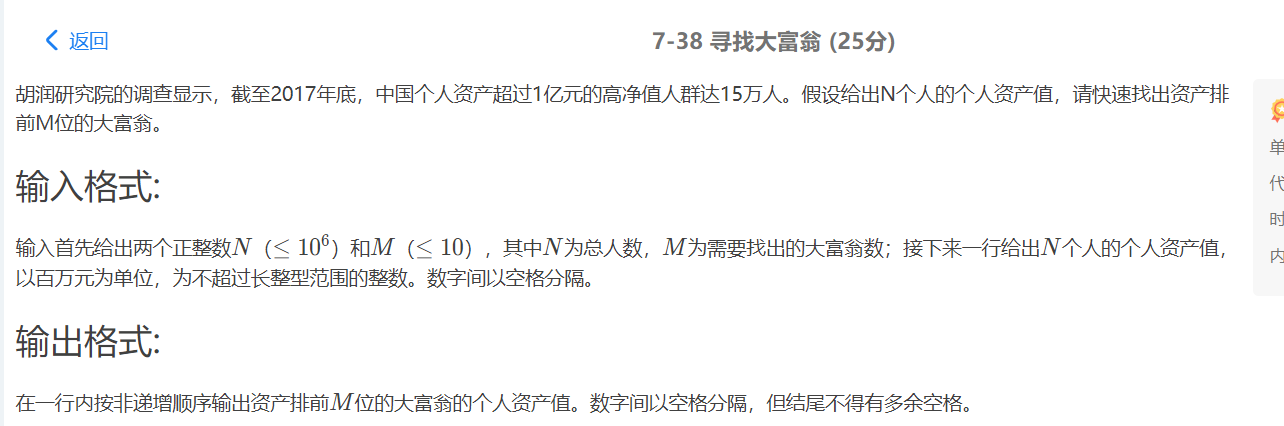
{

cout<<student[i].stuNum<<" "<<student[i].name<<" "<<student[i].grade<<endl;

}

return 0;

}



#include<iostream>

#include<algorithm>

using namespace std;

int num[1000100];

int main()

{

int n,m;

cin>>n>>m;

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

{

cin>>num[i];

}

sort(num,num+n,greater<int>)

if(n<m)

m==n;

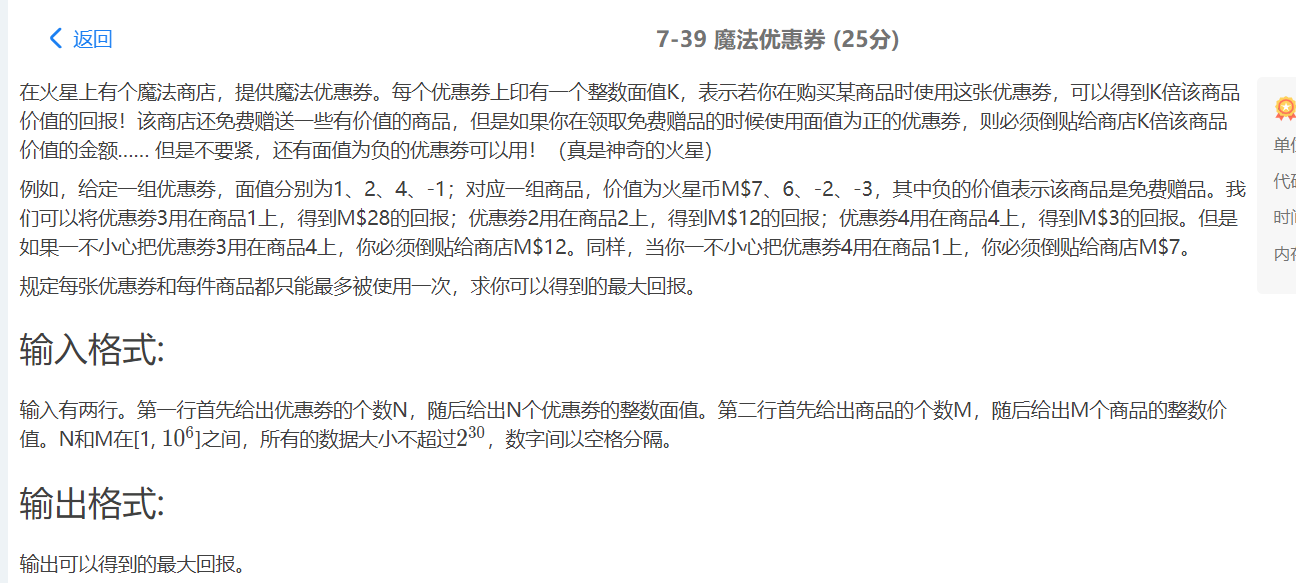
for(int i = 0;i<m - 1;i++)

cout<<num[i]<<" ";

cout<<num[m-1];

return 0;

}



#include<iostream>

#include<algorithm>

using namespace std;

#define MAXN 1000010

int youhui[MAXN];

int goods[MAXN];

int main()

{

int n,m;

cin>>n;

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

{

cin>>youhui[i];

}

cin>>m;

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

{

cin>>goods[j];

}

sort(youhui,youhui+n,greater<int>());

sort(goods,goods+m,greater<int>());

int head = 0;

int tail1 = n-1,tail2 = m-1;

int sum = 0;

while(youhui[head]\*goods[head]>0)

{

sum+=youhui[head]\*goods[head];

head++;

}

while(youhui[tail1]\*goods[tail2]>0&&tail1>head&&tail2>head)

{

sum+=youhui[tail1]\*goods[tail2];

tail1--;

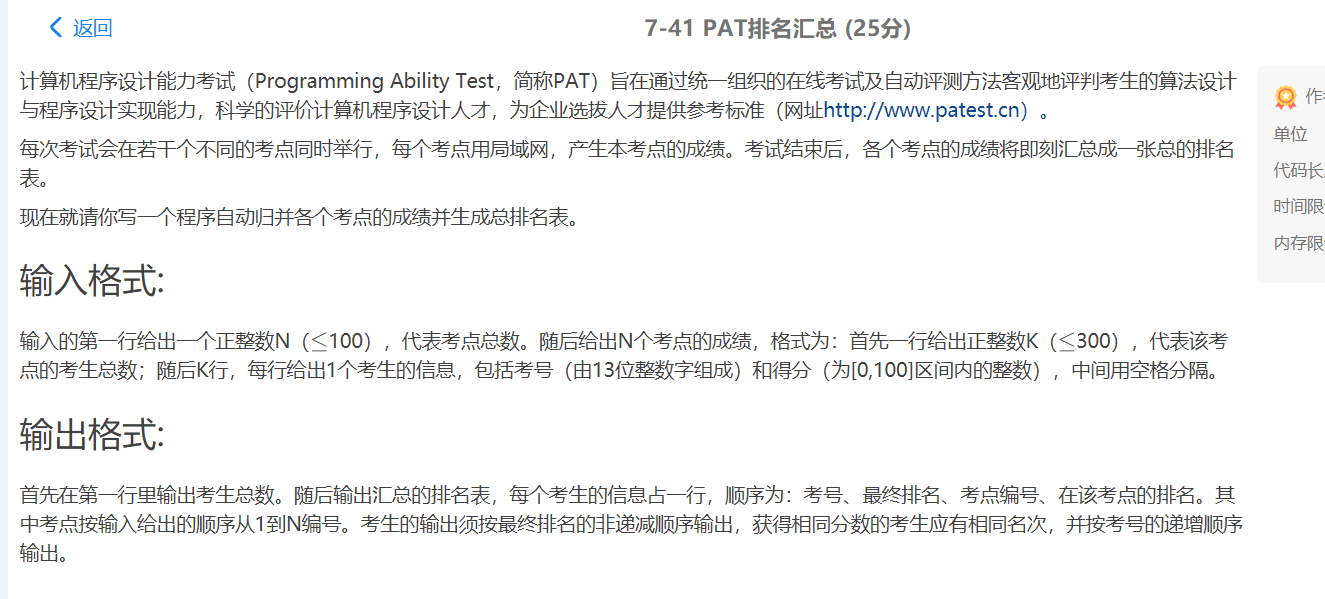
tail2--;

}

cout<<sum<<endl;

return 0;

}



#include<iostream>

#include<string>

#include<algorithm>

#include<cstdio>

using namespace std;

#define MAXN 30010

typedef struct Stu{

string id;

int score;

int allRank;

int placeRank;

int placeId;

};

Stu stu[MAXN];

bool cmp(Stu a,Stu b)

{

if (a.score!=b.score)

return a.score>b.score;

else

return a.id<b.id;

}

int main()

{

int n;

cin>>n;

int num = 0;

int k;

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

{

cin>>k;

for(int j = num;j<num+k;j++)

{

cin >> stu[j].id >> stu[j].score;

stu[j].placeId = i+1;

}

sort(stu+num,stu+num+k,cmp); //处理每一个考场的排名问题

int level = 1;

for(int j = num;j<num+k;j++) //注意考虑重分的人

{

if(j==num)

stu[j].placeRank = level;

else

{

if(stu[j].score==stu[j-1].score)

stu[j].placeRank = stu[j-1].placeRank;

else stu[j].placeRank = level;

}

level++;

}

num+=k; //注意num要加k

}

sort(stu,stu+num,cmp);

int level = 1;

for(int j = 0;j<num;j++) //注意考虑重分的人

{

if(j==0)

stu[j].allRank = level;

else

{

if(stu[j].score==stu[j-1].score)

stu[j].allRank = stu[j-1].allRank;

else stu[j].allRank = level;

}

level++;

}

cout<<num<<endl;

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

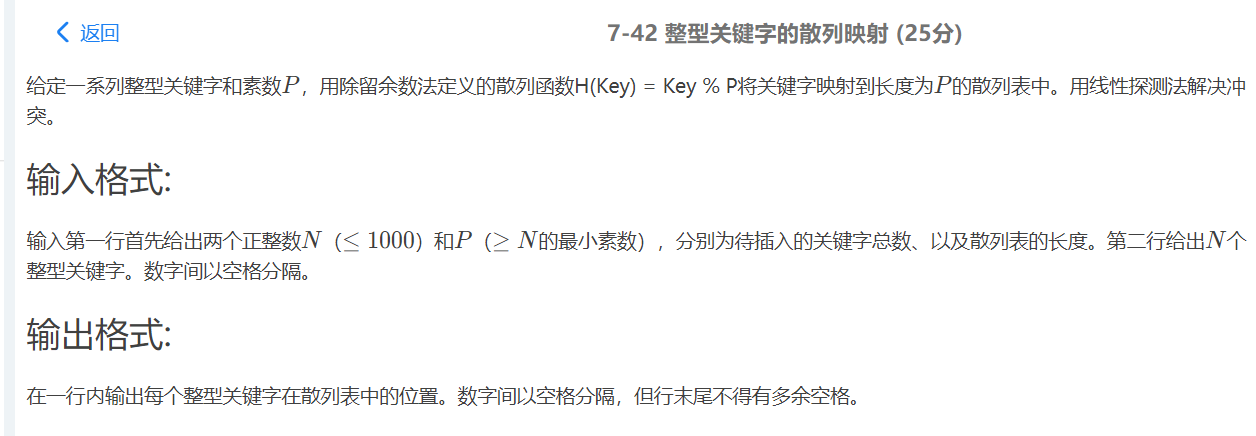
{

cout<<stu[i].id<<" "<<stu[i].allRank<<" "<<stu[i].placeId<<" "<<stu[i].placeRank<<endl;

}

return 0;

}



#include<iostream>

#include<cstring>

#include<cstdio>

using namespace std;

int N,P,flag = 0,address[1010],number[1010];

int Hash(int x)

{

return x%P;

}

int Search(int v)

{

for(int i = v;i < P;i++){

if(!address[i]){

return i;

}

else if(i + 1 == P){

i = -1;

}

}

}

int main()

{

cin >> N >> P;

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

int temp,v;

bool f = false;

scanf("%d",&temp);

v = Hash(temp);

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

if(number[j] == temp){

printf(" %d",j);

f = true;

break;

}

}

if(f)

continue;

if(!address[v]){

address[v] = 1;

number[v] = temp;

printf("%s%d",flag++?" ":"",v);

}

else{

v = Search(v);

address[v] = 1;

number[v] = temp;

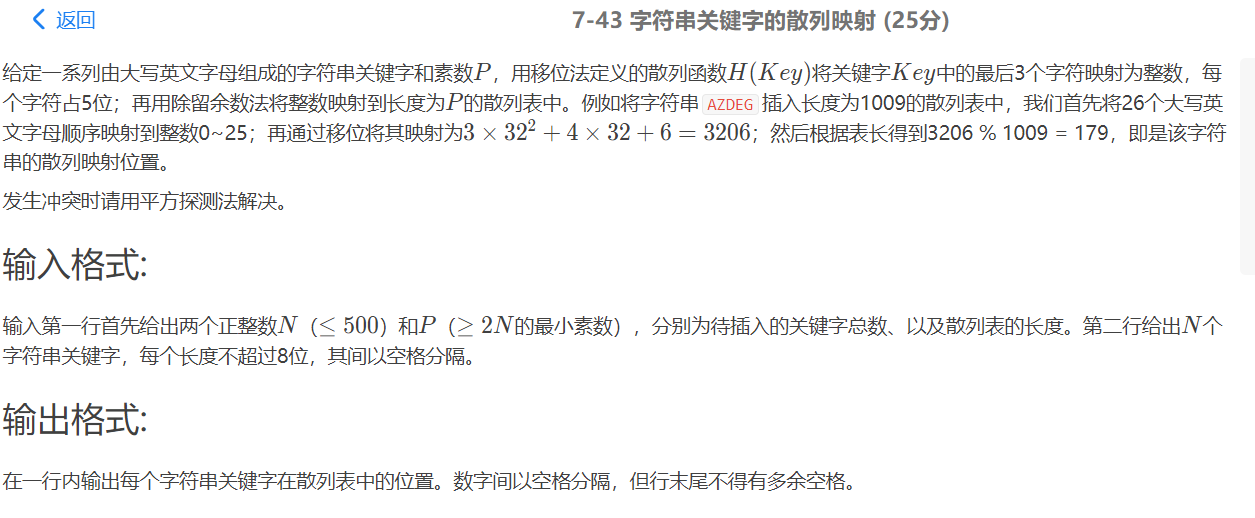
printf("%s%d",flag++?" ":"",v);

}

}

return 0;

}



#include<iostream>

#include<cstdio>

#include<algorithm>

#include<cstring>

#include<set>

#include<map>

using namespace std;

#define maxn 1000007

#define INF 0xffffff

#define mod = 1e6+7;

map<string, int> mp;

int vis[maxn] = {0};

int main() {

string s;

int n, m;

cin>>n>>m;

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

cin >> s;

int len = s.size();

int num = 0;

if(len == 1) {

num += (s[0]-'A');

}

else if(len == 2) {

num = 32\*(s[0]-'A') + (s[1]-'A');

}

else {

for(int j = 3; j >= 1; --j) {

int pos = len - j;

num = num \* 32 + s[pos] - 'A';

}

}

num %= m;

if(mp[s] == 0 && vis[num]) {

for(int t = 1; t < maxn; ++t) { //平方探测处理冲突

if(!vis[(num+t\*t)%m]) {

num = (num+t\*t)%m;

vis[num] = true;

mp[s] = num;

cout << num;

break;

}

else if(!vis[(num-t\*t+m)%m]) {

num = (num-t\*t+m)%m;

vis[num] = true;

mp[s] = num;

cout << num;

break;

}

}

}

else {

num %= m;

mp[s] = num;

cout << num;

vis[num] = true;

}

if(i < n)

cout << " ";

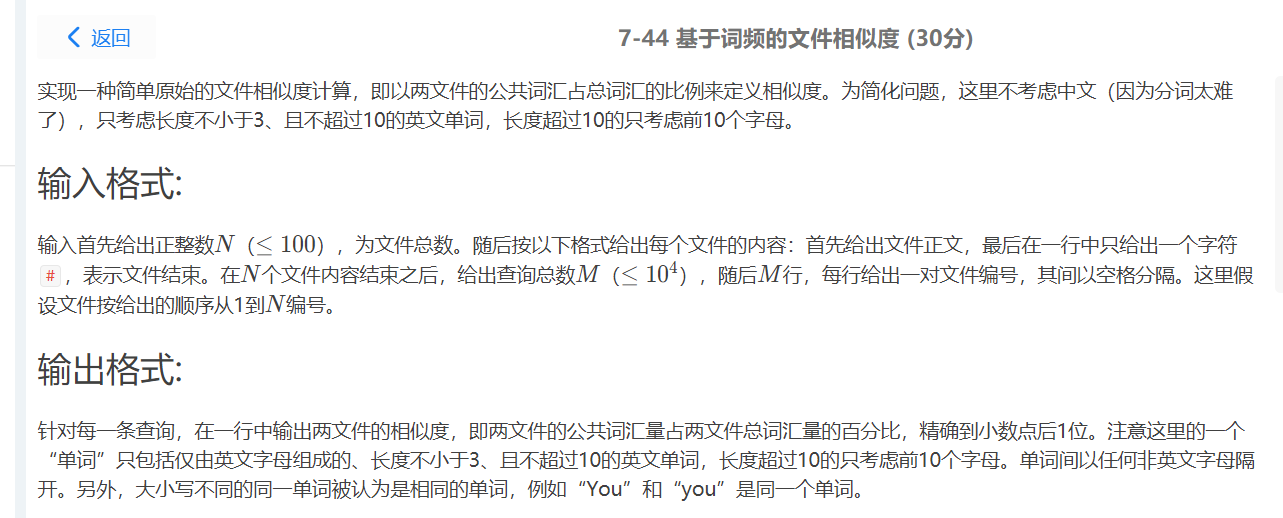
else

cout << endl;

}

return 0;

}



#include<bits/stdc++.h>

#include<cmath>

#define mem(a,b) memset(a,b,sizeof a);

#define INF 0x3f3f3f3f

using namespace std;

typedef long long ll;

set<string> st[110];

int main()

{

int n;

char ts[15],s[100];

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

{

int kase=1;

for(int i=0;i<110;i++) st[i].clear();

while(~scanf("%s",s))

{

int len=strlen(s),k=0;

if(s[0]=='#' && len==1)

{

if(kase++==n) break;

}

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

{

if(s[i]>='a'&&s[i]<='z') s[i]=toupper(s[i]);

}

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

{

char c=s[i];

if(isupper(c))

{

if(f) continue;

ts[k++]=c;

if(k>=10) // 超过10，有字符分隔else会处理，没有字符分隔这边处理

{

ts[k++]='\0';

f=1;

k=0; // 避免与for外面的判断混淆

st[kase].insert(ts);

}

}

else // 有字符分隔

{

if(!f) // 没超过10

{

if(k>=3)

{

ts[k++]='\0';

st[kase].insert(ts);

}

}

else; // 超过10

f=0;

k=0;

}

}

if(k>=3)

{

ts[k++]='\0';

st[kase].insert(ts);

}

}

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

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

{

up=0;

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

int mi=min(st[a].size(),st[b].size());

if(st[a].size()==mi)

{

for(set<string>::iterator it=st[a].begin();it!=st[a].end();it++)

if(st[b].count(\*it)==1) up++;

}

else

{

for(set<string>::iterator it=st[b].begin();it!=st[b].end();it++)

if(st[a].count(\*it)==1) up++;

}

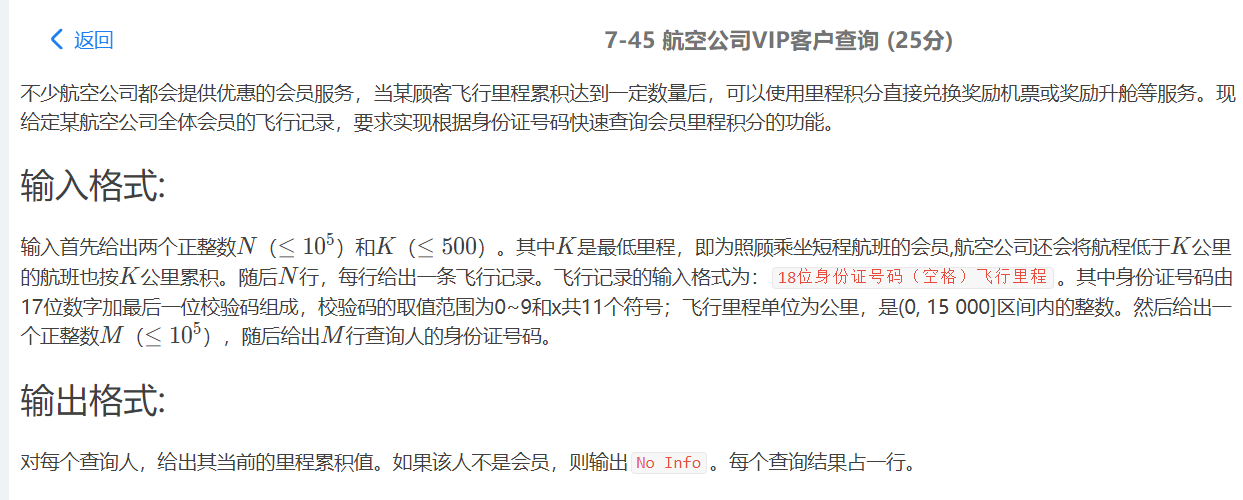
printf("%.1f%%\n",up\*1.0/(st[a].size()+st[b].size()-up)\*100);

}

}

return 0;

}



#include <cstdio>

#include <iostream>

#include <cstring>

#include <map>

using namespace std;

int n,k;

map<string ,int> vip;

int main()

{

char x[17];

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

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

{

int y;

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

if(y<k)

y = k;

vip[x]+=y;

}

int m;

scanf("%d",&m);

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

{

scanf("%s",x);

if(vip.find(x)!=vip.end())

printf("%d\n",vip[x]);

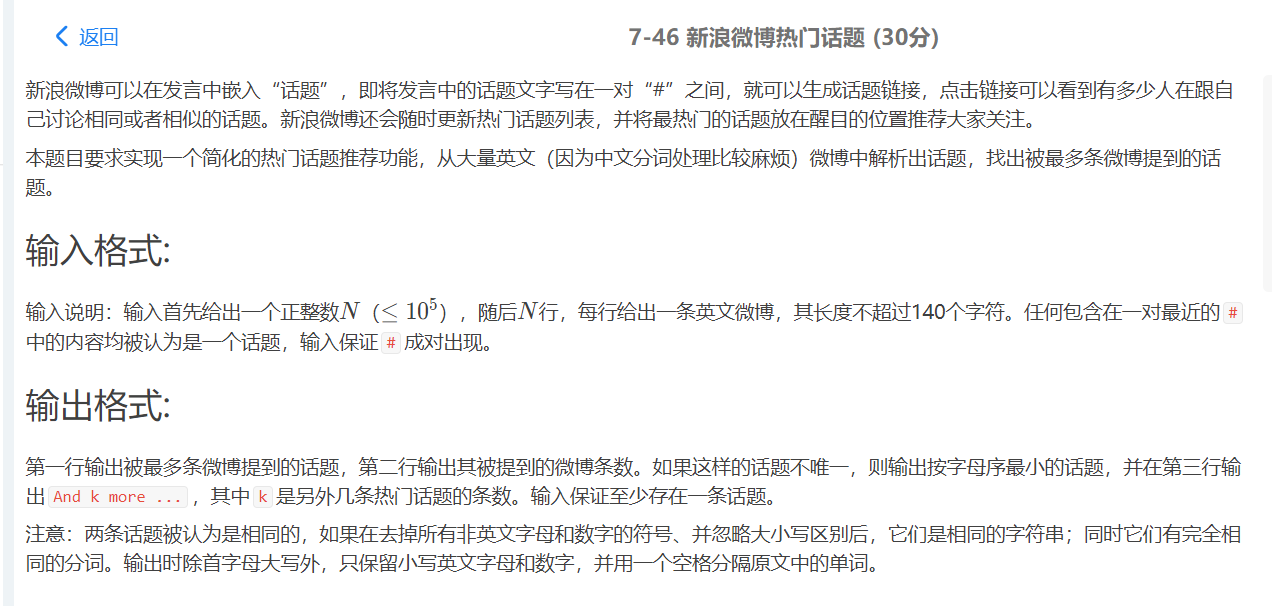
else

printf("No Info\n");

}

return 0;

}



#include<bits/stdc++.h>

#include<cmath>

#define mem(a,b) memset(a,b,sizeof a);

#define INF 0x3f3f3f3f

using namespace std;

typedef long long ll;

typedef pair<string,int> psi;

set<string> st; // 解决每一行重复给 ump 添加的问题

unordered\_map<string,int> ump;

int cmp(psi p1,psi p2)

{

if(p1.second==p2.second) return p1.first<p2.first;

return p1.second>p2.second;

}

int main()

{

string s;

char ts[150];

int n;

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

{

getchar();

ump.clear();

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

{

st.clear();

getline(cin,s);

char c;

int len=s.length(),jl=0,k=0,blank=1;

while(jl<len && s[jl]!='#') jl++;

for(int j=jl+1;j<len;j++)

{

c=s[j];

if(c=='#')

{

// 首位空格处理

int l=0,r=k-1;

while(ts[l]==' ') l++;

while(ts[r]==' ') r--;

strncpy(ts,ts+l,r-l+1);

ts[r-l+1]=0;

k=0; // 开始新的#...#话题计数

if(st.insert(ts).second)

ump[ts]++;

jl=j+1;

while(jl<len && s[jl]!='#') jl++;

j=jl; // 不用加 1，因为马上就 j++

}

if(isupper(c)) ts[k++]=tolower(c), blank=0;

else if(islower(c)||isdigit(c)) ts[k++]=c, blank=0;

else

{

if(blank==0)

ts[k++]=' ',blank=1;

}

}

}

vector<psi> vec(ump.begin(),ump.end());

sort(vec.begin(),vec.end(),cmp);

vec[0].first[0]=toupper(vec[0].first[0]);

printf("%s\n",vec[0].first.c\_str());

int jde=vec[0].second,cnt=0;

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

for(int i=1;i<vec.size();i++)

if(vec[i].second==jde) cnt++;

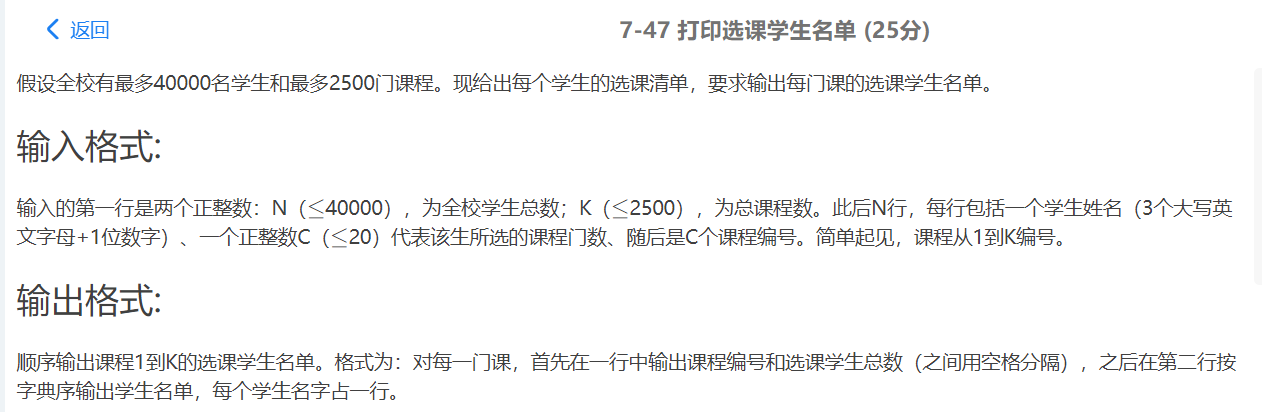
else break;

if(cnt>0) printf("And %d more ...\n",cnt);

}

return 0;

}



#include <iostream>

#include <vector>

#include <string>

#include <cmath>

#include <algorithm>

#include <queue>

#include <cstdio>

#include <cctype>

#include <climits>

#include <unordered\_map>

#include <map>

#include <set>

#include <cstring>

using namespace std;

int main(){

int n, k, id, m;

string name;

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

vector<vector<string>> V(k+1);

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

cin>>name;

scanf("%d", &m);

while(m--){

scanf("%d", &id);

V[id].push\_back(name);

}

}

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

sort(V[i].begin(), V[i].end());

printf("%d %d\n", i, V[i].size());

for(auto s: V[i]){

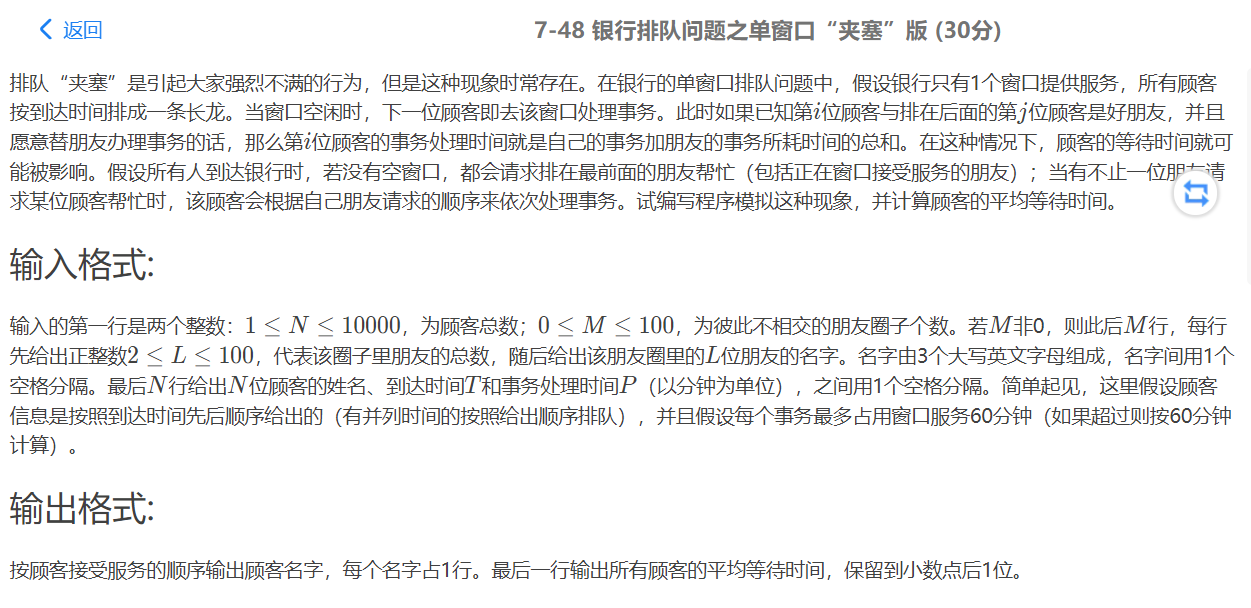
printf("%s\n", s.c\_str());

}

}

return 0;

}



#include <iostream>

#include <vector>

#include <map>

#include <iomanip>

using namespace std;

struct Customer {

string name;

int T,

P,

wait,

vis;

};

map<string, int>mp;

vector<Customer>customer;

vector<string>ans;

int N, M, L, now = 0, sum = 0, vis[10000];

int main() {

cin >> N >> M;

customer.resize(N);

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

cin >> L;

string name;

while (L--) {

cin >> name;

mp[name] = i;

}

}

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

cin >> customer[i].name >> customer[i].T >> customer[i].P;

customer[i].P = min(customer[i].P, 60);

}

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

if (!customer[i].vis) {

now = max(now, customer[i].T);

customer[i].wait = now - customer[i].T;

now += customer[i].P;

ans.push\_back(customer[i].name);

customer[i].vis = 1;

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

if (!customer[j].vis && mp[customer[i].name] == mp[customer[j].name] && customer[j].T <= now) {

customer[j].wait = now - customer[j].T;

now += customer[j].P;

ans.push\_back(customer[j].name);

customer[j].vis = 1;

}

}

for (auto& it : ans)

cout << it << endl;

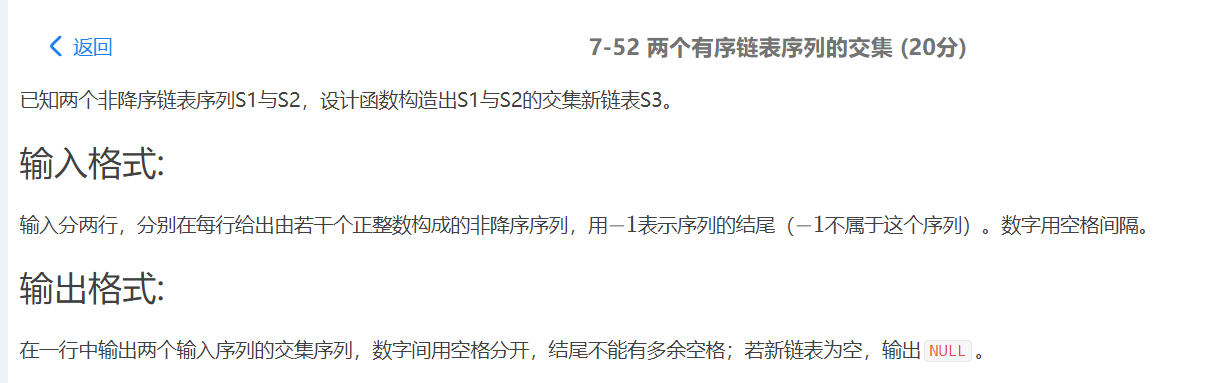
for (auto& it : customer)

sum += it.wait;

cout << fixed << setprecision(1) << 1.0 \* sum / N;

return 0;

}



#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

typedef struct Node{

int Data;

Node \*Next;

} \*List;

List Read();//读入链表

List Common(List L1, List L2);//找出L1,L2交集

int main()

{

List L1, L2, L3;

L1 = Read();

L2 = Read();

L3 = Common(L1, L2);

L3 = L3->Next;//跳过表头

int flag=0;

if(!L3) printf("NULL\n");

while(L3){

if(flag) printf(" ");//之前已有输出则输出空格

printf("%d",L3->Data);

flag = 1;

L3 = L3->Next;

}

return 0;

}

List Read(){

List L, tmp, LL;

int Data;

L = (List)malloc(sizeof(List));

L->Data = -1;

L->Next = NULL;

LL = L;

scanf("%d",&Data);

while(Data!=-1){

tmp = (List)malloc(sizeof(List));

tmp->Data = Data;

tmp->Next = NULL;

LL->Next = tmp;

LL = LL->Next;

scanf("%d",&Data);

}

return L;

}

List Common(List L1, List L2){

List tmp1,tmp2, L, LL, tmp;//tmp1,tmp2分别在L1,L2中循环，L为返回的表头，LL为循环变量，tmp为交集的临时节点

L = (List)malloc(sizeof(List));

L->Data = -1;

L->Next = NULL;

LL = L;

tmp1 = L1->Next;//跳过表头

tmp2 = L2->Next;

while(tmp1 && tmp2){//只要一个链表空则查找结束

if(tmp1->Data==tmp2->Data){//若相等则插入交集链表

tmp = (List)malloc(sizeof(List));

tmp->Data = tmp1->Data;

tmp->Next = NULL;

LL->Next = tmp;

LL = LL->Next;

tmp1 = tmp1->Next;

tmp2 = tmp2->Next;

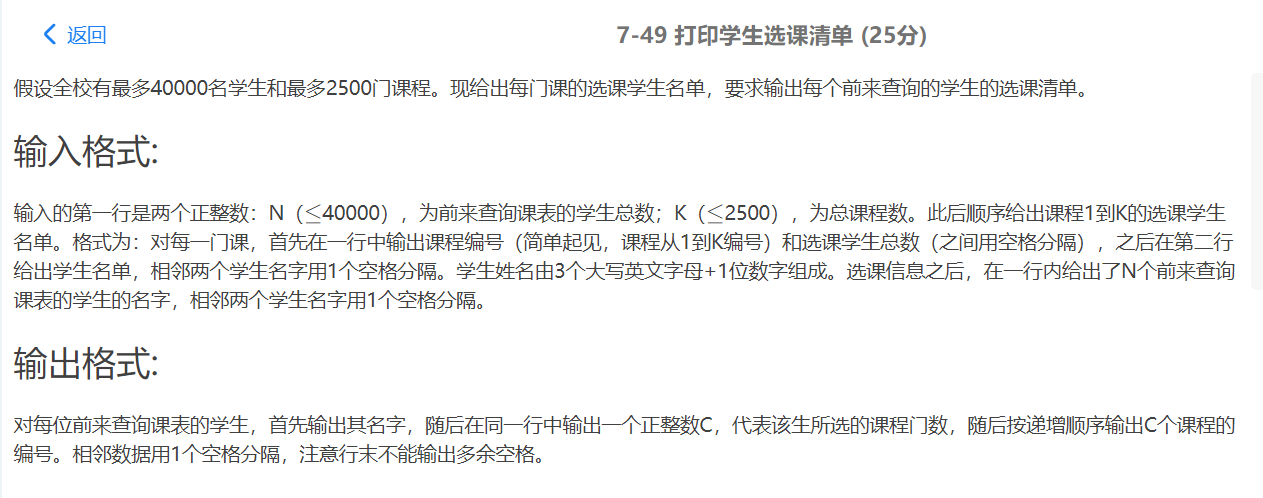
}else if(tmp1->Data < tmp2->Data) tmp1 = tmp1->Next;//非降序，故不等时数据较小的链表节点后移即可

else tmp2 = tmp2->Next;

}

return L;

}



#include <stdio.h>

#include <malloc.h>

typedef struct TNode {

int CourseNo;

struct TNode \*Left,\*Right;

}\*Tree;

typedef struct Node {

int num;

struct TNode \*Next;

} Node;

typedef char Element[5];

Tree BuildBiSearchTree(Tree T,int CourseNo);

void Trav(Tree T);

void Out(Node \*head);

int main(int argc,char \*\*argv) {

int i,j,k,t;

int n,m;

int CourseNo,Num;

Element Name;

Node \*List[26][26][26][10];

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

for(j=0; j<26; j++) {

for(k=0; k<26; k++) {

for(t=0; t<10; t++) {

List[i][j][k][t]=NULL;

}

}

}

}

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

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

scanf("%d %d",&CourseNo,&Num);

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

scanf("%s",Name);

if(List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']==NULL)

{

List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']=malloc(sizeof(struct Node));

List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']->num=0;

List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']->Next=NULL;

}

List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']->num++;

List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']->Next

=BuildBiSearchTree(List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']->Next,CourseNo);

}

}

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

scanf("%s",Name);

printf("%s",Name);

if(List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']!=NULL)

Out(List[Name[0]-'A'][Name[1]-'A'][Name[2]-'A'][Name[3]-'0']);

else

printf(" 0");

printf("\n");

}

}

Tree BuildBiSearchTree(Tree T,int CourseNo) {

if(!T) {

T=malloc(sizeof(struct TNode));

T->CourseNo=CourseNo;

T->Left=T->Right=NULL;

} else if(CourseNo<T->CourseNo)

T->Left=BuildBiSearchTree(T->Left,CourseNo);

else

T->Right=BuildBiSearchTree(T->Right,CourseNo);

return T;

}

void Trav(Tree T) {

if(T) {

Trav(T->Left);

printf(" %d",T->CourseNo);

Trav(T->Right);

}

}

void Out(Node \*head) {

printf(" %d",head->num);

if(head->Next) {

Tree T=head->Next;

Trav(T);

}

}

**6、哈夫曼树（二叉树，功能不是很完善）**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

// 哈夫曼树

typedef struct {

char info;

int weight;

int parent, lchild, rchild;

} HTNode, \*HuffmanTree;

typedef char\* \*HuffmanCode;

// 全局变量

HuffmanTree HT; // 哈夫曼树

HuffmanCode HC; // 哈夫曼树的结点

int n = 8; // 默认有八个字符及权重

// 选取最小权重点函数

void Select(HuffmanTree HT, int t, int \*s1, int \*s2) {

int i, temp1, temp2;

temp1 = temp2 = 1000;

// 在哈夫曼树HT中选择父亲不为0且权值最小的两结点，索引分别记为s1和s2

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

if (HT[i].parent == 0 && (HT[i].weight<temp1 || HT[i].weight<temp2)) {

if (temp1<temp2) {

temp2 = HT[i].weight;

\*s1 = i;

} else {

temp1 = HT[i].weight;

\*s2 = i;

}

}

}

// s1为较小的序号，s2取较大的序号

if (\*s1 > \*s2) {

i = \*s1;

\*s1 = \*s2;

\*s2 = i;

}

}

// 哈夫曼树编码（n为字符串个数）

HuffmanTree HuffmanCoding(int \*w, int n, char \*info) {

// p为当前所指结点

HuffmanTree HT, p;

char \*cd;

// s1/s2记录权重最小的两点，f为父结点，c为子结点

int m, s1, s2, i, start, f, c;

if (n <= 1) {

return 0;

}

m = 2\*n-1;

HT = (HuffmanTree)malloc((m+1)\*sizeof(HTNode));

p = HT + 1;

// 初始化各点权重

for (i=1; i<= n; ++i, ++p, ++w) {

// 不用第0号单元

p->weight = \*w;

p->info = info[i-1];

p->parent = 0;

p->lchild = 0;

p->rchild = 0;

}

for (; i<= m; ++i, ++p) {

p->weight = 0;

p->parent = 0;

p->lchild = 0;

p->rchild = 0;

}

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

// 调用select函数

Select(HT, i-1, &s1, &s2);

// 根据select返回的两个点，生成新节点

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

}

// 逆向求哈夫曼编码

HC = (HuffmanCode)malloc((n+1)\*sizeof(char \*));

code = (char \*)malloc(n\*sizeof(char));

code[n-1] = '\0';

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

start = n - 1;

for (c = i, f = HT[i].parent; f != 0; c = f, f = HT[f].parent) {

// 从子叶到根逆向求编码

if (HT[f].lchild == c) {

code[--start] = '0';

} else {

code[--start] = '1';

}

}

HC[i] = (char \*)malloc((n-start)\*sizeof(char));

strcpy(HC[i], &cd[start]);

}

free(code);

return HT;

}

// 编码文本

void TextToCode() {

char str[50];

int i, j, p;

if (HT == NULL) {

printf("请先进行初始化！\n");

return ;

}

printf("请输入编码的文本（英文字母）：\n");

getchar();

gets(str);

printf("编码结果如下：\n");

p = strlen(str);

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

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

if (str[i-1] == HT[j].info) {

printf("%s", HC[j]);

break;

}

}

}

printf("\n");

}

// 译码

void CodeToText() {

int i = 1, j, key;

int i1, i2;

char str[100];

int a = 1;

int m = 2 \* n - 1;

if (HT == NULL) {

printf("请先进行初始化！\n");

return ;

}

printf("\n各字母对应的哈夫曼编码如下：\n");

/\* 以一定格式输出字母对应的编码 \*/

for (i1 = i; i1 <= 2; i1++) {

printf("┌———————┐ ┌———————┐ ┌———————┐ ┌———————┐\n");

for (i2 = 1; i2 <= 4; i2++) {

printf(" %c：%-11s ", HT[a].info, HC[a]);

a++;

}

printf("\n");

printf("└———————┘ └———————┘ └———————┘ └———————┘\n");

}

printf("\n请输入哈夫曼编码：\n");

scanf("%s", str);

j = strlen(str);

key = m;

printf("哈夫曼编码译码如下：\n");

while (i <= j) {

// 若左孩子不为0

while (HT[key].lchild != 0) {

if (str[i-1] == '0') {

key = HT[key].lchild;

i++; continue;

}

if (str[i-1] == '1') {

key = HT[key].rchild;

i++; continue;

}

}

printf("%c", HT[key].info);

key = m;

}

printf("\n");

}

// 菜单界面

void menu() {

printf("┌—————————————————————————┐\n");

printf("| 1 构造哈夫曼树 | \n");

printf("└—————————————————————————┘\n");

printf("┌—————————————————————————┐\n");

printf("| 2 哈夫曼编码 | \n");

printf("└—————————————————————————┘\n");

printf("┌—————————————————————————┐\n");

printf("| 3 哈夫曼译码 | \n");

printf("└—————————————————————————┘\n");

printf("┌—————————————————————————┐\n");

printf("| 4 退出 | \n");

printf("└—————————————————————————┘\n");

}

// 哈夫曼树的Demo主函数

int main() {

int flag, mark = 1;

// 默认的八个字母和权重（可修改）

int wei[8] = {1, 3, 5, 7, 8, 9, 10, 13};

char info[8] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};

for (;mark;) {

menu();

scanf("%d", &flag);

switch(flag) {

case 1:

HT = HuffmanCoding(wei, n, info);

break;

case 2:

TextToCode();

break;

case 3:

CodeToText();

break;

case 4:

j = 0;

break;

default: printf("输入错误，重新输入！\n");

}

}

return 0;

}

## **一.vector**

****1.**** 应用  
用于普通数组会超内存的情况  
以邻接矩阵的方式存储图  
****2.**** 定义：typename可以是int,double,char,结构体  
****3.**** 元素访问  
下标访问  
迭代器访问：vector< typename >::iterator it;  
****4.**** 函数

#include<vector>

using namespace std;

vector<typename>name;*//定义*

vector<vector<int> >name;*//如果typename也是一个vector,>>之间要有空格*

vector<typename>name[maxn];*//二维vector< typename >name[maxn],有一维长度固定为maxn，无法改变*int main(){

push\_back();*//vector后加一个元素*

pop\_back();*//删去vector的尾元素*

size();*//获得vector中元素个数*

clear();*//清空*

insert(it,x);*//向vector任意迭代器it处插入一个元素x*

erase(it);*//删去单个元素*

erase(first,last);*//删去区间[first,last)内的所有元素*

return 0;}

## **二.set**

****1.**** 应用：去重，自动排序  
****2.**** 定义：typename类型同vector;  
****3.**** 元素访问：只能用迭代器set< typename >::iterator it;  
****4.**** 函数：

#include<set>*//头文件*

using namespace std;

set<typename>name;*//定义*int main(){

insert(x);*//将x插入set容器中，并自动排序，去重*

find(value);*//返回对应值为value的迭代器*

size();*//获得vector中元素个数*

*//删去单个元素*

name.erase(it);*//*

name.erase(value);*//*

name.erase(first,last);*//删去区间[first,last)内的所有元素*

clear();*//清空*

insert(it,x);*//向vector任意迭代器it处插入一个元素x*

return 0;}

## **三.string**

****1.**** 应用  
代替一般字符数组char str[];  
****2.**** 访问：  
通过下标访问  
注：如果要读入或者输出，只能用cin,cout  
迭代器访问：string::iterator it;  
string和vector一样，支持直接对迭代器进行加减某个数  
如：str.begin()+3;  
****3.**** 函数：

#include<string>

using namespace std;

string str;int main(){

str1+=str2;*//直接将str1,str2连起来，拼到str1上*

str1==str2;*//类似的操作还有：!=,<,<=,>,>=*

length(str);*//=size(str);*

insert(pos,str);*//在pos位置上插入str*

insert(it,it1,it2)*//it为字符串要插入的位置，it1和it2分别是待插入字符串的首尾*

str.erase(it);

str.erase(first,last);

str.erase(pos,len);*//pos是开始删除的位置，len是要删除的字符串的长度*

str.clear();

substr(pos,len);*//返回从pos开始，长度为len的字符串*

string::npos*//一个常数，值为-1，实际也可做unsigned\_int的最大值*

str.find(str2);*//str2是str子串，返回子串第一次出现的位置，否则返回string::npos*

str.find(pos,str2);*//从pos开始找str2,其他同上*

str.replace(pos,len,str2);*//将从pos位置开始长为len的子串替换为str2*

return 0;}

## **四.map**

1.应用：可将任何基本类型映射到任何基本类型  
2.访问：  
通过下标访问:mp[‘c’]=a,注意引号  
通过迭代器访问：map<typename1,typename2>::iterator it;  
可用it->first访问键，it->second访问值  
另外，map会以键的自大到小排序  
3.函数

#include<map>

using namespace std;

map<typename1,typename2>mp;*//name1是键的类型,name2是值的类型*int main(){

find(key);*//返回key映射的迭代器*

mp.erase(it);*//同上*

mp.erase(key);*//key为要删除的映射的键*

mp.erase(first,last);*//同上*

size();

clear();

return 0;}

## **五.queue**

1.思想：先进先出  
2.访问：只能用front()访问队首，back()访问队尾  
3.函数：

#include<queue>

using namespace std;

queue<typename>q;*//name1是键的类型,name2是值的类型*int main(){

q.push();*//入队*

q.front();*//获取队首*

q.back();*//获取队尾*

q.pop();*//出队*

q.empty();*//判断是否为空*

q.size();

return 0;}

## **六.priority\_queue**

1.访问：只能用top()访问队首元素  
2.函数

#include<queue>

using namespace std;

priority\_queue<typename>q;*//name1是键的类型,name2是值的类型*int main(){

q.push();*//入队*

q.top();*//获取队首*

q.pop();*//出队*

q.empty();*//判断是否为空*

q.size();

return 0;}

3.优先级设置  
(1)基本数据类型：

priority\_queue<int,vector<int>,less<int>> q;

priority\_queue<typename>q;*//二者等价*

less< int >表示数字越大优先级越大，great< int >刚好相反；  
char行根据字典序来排序  
(2)结构体：

struct node{

string a;

int b;

friend bool operator<(node x,node y)*//重载小于号，即对小于符号重新定义*

{

return x.b<y.b;

* }};

注：1.此函数与sort中cmp的效果刚好相反  
2.若结构体内数据较为庞大，可使用引用来提高效率，即加上“const”“&”,如：

friend bool operator<(const node &x,const node &y)

{

return x.b<y.b;

}