程序设计实验室暑期选拔赛 题解

1.冬冬走迷宫

题目描述

众所周知，冬冬是一个~~土豪以及游戏狂热玩家~~。最近他趁着暑假，买了许多游戏来消遣。可是当他把所有游戏都通关了以后，感到无敌是多么寂寞，必须靠经典才能满足他。于是他又玩起了走迷宫的小游戏。

走迷宫可以简化为一个矩阵，里面含n*m个格子。有些格子是可以进入并且通过的，有些格子内含障碍物，是不能进入通过它的。我们的任务就是在给定一个起点S和一个终点T后，选择一个不经过障碍物以及走出矩阵边界的方案，把自己控制的角色从S走到T。

然而冬冬身经百战，不知道比其他人高到哪里去了，因此他总会选择最短的路径来走完这个迷宫。可是他不能确定自己的路径是不是最短的，因此他希望你能帮助他，给出每个迷宫的最短路径长度(即从S到T经过（包含S和T）的格子数）。 作为新晋ACMer，你能帮助冬冬完成这个任务吗?

Input

多组数据输入，保证格子总数不超过100w。

第一行是两个正整数n，m，代表矩阵的高度和宽度。其中0≤n,m≤100.

接下来n行m列，给出迷宫每个格子的情况。’S’代表起点，’T’代表终点，’#’代表障碍物，’.’代表可通行区域。

Output

对于每组数据，输出一行从S到T的最短路径的长度。若无从S到T的路径，则输出“Impossible”。

Sample Input

```
5 6
S.....
#.###.
##.T..
#.###.
#.....
2 9
S.#......
###T####.
```

Sample Input

```
10
Impossible
```

题解

一道很基础的bfs题，除了迷宫的地图数组外，加入标记数组，步数数组，bfs时让每个格子向四周扩展，把没有标记的格子以及他的步数加入队列中并标记他们，还需在步数数组上记录到该点的步数，直至队列中没有元素为止（或者bfs

到T点就跳出循环）。最后检查下终点T有无被标记过。没有则是Impossible，有则输出T点的步数。

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<queue>
#define clr_1(x) memset(x,-1,sizeof(x))
#define clr(x) memset(x,0,sizeof(x))
using namespace std;
struct node
{
    int x,y,step;
}u,v;
queue<node> que;
char s[200];
int map[200][200];
int stepm[200][200];
int dirx[4]={0,0,-1,1},diry[4]={1,-1,0,0};
int n,m,t,k,sx,sy,tx,ty;
void init()
{
    clr(map);
    for(int i=1;i<=n;i++)
    {
        scanf("%s",s);
        for(int j=0;j<strlen(s);j++)
        {
            if(s[j]=='S')
            {
                sx=i;
                sy=j+1;
                map[i][j+1]=1;
            }
            if(s[j]=='T')
            {
                tx=i;
                ty=j+1;
                map[i][j+1]=1;
            }
            if(s[j]=='.')
            {
                map[i][j+1]=1;
            }
        }
    }
    while(!que.empty())
        que.pop();
    return ;
}
void bfs()
{
    while(!que.empty())
    {
        u=que.front();
        que.pop();
        for(int i=0;i<4;i++)
        {
            v=u;
            v.x+=dirx[i];
            v.y+=diry[i];
            v.step++;
```

```
            if(map[v.x][v.y]==1)
            {
                map[v.x][v.y]=0;
                stepm[v.x][v.y]=v.step;
                que.push(v);
            }
        }

    }
    return ;
}
int main()
{
    freopen("1.in","r",stdin);
    freopen("1.out","w",stdout);
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        init();
        u.x=sx;
        u.y=sy;
        u.step=1;
        map[sx][sy]=0;
        que.push(u);
        bfs();
        if(map[tx][ty]==1)
        {
            printf("Impossible\n");
        }
        else
        {
            printf("%d\n",stepm[tx][ty]);
        }
    }
    return 0;
}
```

2.冬冬沉迷走迷宫

题目描述

众所周知，冬冬是一个~~土豪以及游戏狂热玩家~~。最近他趁着暑假，买了许多游戏来消遣。可是当他把所有游戏都通关了以后，感到无敌是多么寂寞，必须靠经典才能满足 他。于是他又玩起了走迷宫的小游戏。

走迷宫可以简化为一个矩阵，里面含n*m个格子。有些格子是可以进入并且通过的，有些格子内含障碍物，是不能进入通过它的。我们的任务就是在给定一个起点S和一个终点T后，选择一个不经过障碍物以及走出矩阵边界的方案，把自己控制的角色从S走到T。

然而冬冬身经百战，不知道比其他人高到哪里去了，因此他总会选择最短的路径来走完这个迷宫。可是他不能确定自己的路径是不是最短的，并且他想知道最完美的方案一共有几种。因此他希望你能帮助他，给出每个迷宫的最短路径长度(即从S到T经过（包含S和T）的格子数）以及该长度下一共有几种不同的走法。 作为新晋ACMer，你能帮助冬冬，完成这个任务吗?

Input

多组数据输入，保证不超过2组数据。

第一行是两个正整数n，m，代表矩阵的高度和宽度。其中2≤n,m≤9。

接下来n行m列，给出迷宫每个格子的情况。’S’代表起点，’T’代表终点，’#’代表障碍物，’.’代表可通行区域，其中S一定在左上角，也就是(0,0)的位置；T一定在右下角，也就是(n-1,m-1)的位置。

Output

对于每组数据，输出一行从S到T的最短路径的长度，以及该长度下的方案数，保证答案在long long范围内。若无从S到T的路径，则输出“Impossible”

Sample Input

```
5 6
S.....
#.###.
##....
#.###.
#....T
2 9
S.#......
########T
```

Sample Output

```
10 1
Impossible
```

题解

当初准备写个插头dp来着的，后来发现bfs也能做。把上题的所有数组保留加入方案数数组。起初S的方案数为1，其余不变。当bfs扩展到某个格子时，若该格子未被标记，记录从之前格子扩展到该格子的步数，该格子方案数为扩展到他的格子的方案数+1，加入队列。若已被标记，且记录的步数和现在扩展到该格子时的步数相同，则在原先方案数上加上这次扩展该格子获得的方案数。若记录的步数比现在扩展到该格子时的步数少，则不处理。这样做一遍bfs之后，跟上面的做法相同，检查下终点T有无被标记过。没有则是Impossible，有则输出T点的步数和方案数。

插头dp的话左上角要建立一个独立插头，右下角要截止一个独立插头。其他格子在dp时有且仅存在一个独立插头，转移的时候和独立插头的转移一样，就是不能创建独立插头以及截止独立插头。最后dp结束检查最后一个格子步数最小的状态的方案数，若没有则是Impossible，有即为答案。

标程用插头dp写的：

```
#include<cstdio>
#include<iostream>
#include<cstring>
#define clr(x) memset(x,0,sizeof(x))
#define clr_1(x) memset(x,-1,sizeof(x))
#define LL long long
#define HASH 10007
#define STATE 1000010
using namespace std;
struct hashmap//hash表 存 状 态
{
    int size;
```

```
    int next[STATE],head[HASH];
    LL state[STATE],len[STATE];
    LL fas[STATE];
    void init()//清 空
    {
        size=0;
        clr_1(head);
    }
    void add(LL st,LL length,LL solu)//状 态 加 入
    {
        int k=(st*10000+length)%HASH;
        for(int i=head[k];i!=-1;i=next[i])
            if(st==state[i] && length==len[i])
            {
                fas[i]+=solu;
                return;
            }
        next[++size]=head[k];
        fas[size]=solu;
        state[size]=st;
        len[size]=length;
        head[k]=size;
        return ;
    }
}dp[2];
int maped[40][40];
int code[40];
char s[100];
void init(int n,int m)//初 始 化 值 ， 读 入maped
{
    clr(maped);
    for(int i=1;i<=n;i++)
    {
        scanf("%s",s);
        for(int j=0;j<strlen(s);j++)
        {
            if(s[j]=='.' || s[j]=='S' || s[j]=='T')
            {
                maped[i][j+1]=1;
            }
        }
    }
    return ;
}
void decode(LL st,int *code,int m)//解 码 ， 括 号 序 列
{
    for(int i=m;i>=0;i--)
    {
        code[i]=st&3;
        st>>=2;
    }
    return ;
}
LL encode(int *code,int m)//编 码 ， 括 号 序 列
{
    LL st=0;
    for(int i=0;i<=m;i++)
    {
        st<<=2;
        st|=code[i];
    }
    return st;
```

```
}
void shift(int *code,int m)//左移操作
{
    for(int i=m;i>0;i--)
        code[i]=code[i-1];
    code[0]=0;
    return ;
}
void dpblank(int i,int j,int cnt,int n,int m)//空格子状态转移
{
    int top,left,up;
    if(i==1 &&j==1)//i=1 且 j=1时加入只有下独立插头 和 右独立插头的状态
    {
        decode(dp[cnt].state[1],code,m);
        if(maped[i][j+1])
        {
            code[j-1]=0;
            code[j]=3;
            dp[cnt^1].add(encode(code,m),dp[cnt].len[1]+1,dp[cnt].fas[1]);
        }
        if(maped[i+1][j])
        {
            code[j-1]=3;
            code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[1]+1,dp[cnt].fas[1]);
        }
        return ;
    }
    if(i==n &&j==m)//i=n 且 j=m时截止单个独立插头的状态
    {
        for(int it=1;it<=dp[cnt].size;it++)
        {
            decode(dp[cnt].state[it],code,m);
            code[j-1]=code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        return ;
    }
    for(int it=1;it<=dp[cnt].size;it++)
    {
        decode(dp[cnt].state[it],code,m);
        left=code[j-1];
        up=code[j];
        if(left && up)//上插头和左插头都存在
        {
            if(left==3 || up==3)//存在一个独立插头
            {
                if(up==2|| left==2)//若另一个插头为右括号插头，则找到对应的左括号插头
变为独立插头
                {
                    top=0;
                    for(int k=j-2;k>=0;k--)
                    {
                        if(code[k]==2)
                            top++;
                        if(code[k]==1)
                            if(top>0)
                                top--;
                            else
                                {
```

```
                        code[k]=3;
                        break;
                    }
                }
            }
            else if(up==1 ||left==1)//若另一个插头为左括号插头，则找到对应的右括号插头变为独立插头
            {
                top=0;
                for(int k=j+1;k<=m;k++)
                {
                    if(code[k]==1)
                        top++;
                    if(code[k]==2)
                        if(top>0)
                            top--;
                        else
                            {
                                code[k]=3;
                                break;
                            }
                }
            }
            code[j]=code[j-1]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        else if(left!=up)//两个括号插头
        {
            if(left==1)//左右括号插头，不允许形成回路
                continue;
            else//右左括号插头直接去掉
            {
                code[j]=code[j-1]=0;
                if(j==m) shift(code,m);
                dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
             }
        }
        else
        {
            if(left==2)//都是左括号插头，找到对应的第一个右括号插头变为左括号插头
            {
                top=0;
                for(int k=j-2;k>=0;k--)
                {
                    if(code[k]==2)
                        top++;
                    if(code[k]==1)
                        if(top>0)
                            top--;
                        else
                            {
                                code[k]=2;
                                break;
                            }
                }
            }
            else//都是右括号插头，找到对应的第一个左括号插头变为右括号插头
            {
                top=0;
                for(int k=j+1;k<=m;k++)
```

```
                {
                    if(code[k]==1)
                        top++;
                    if(code[k]==2)
                        if(top>0)
                            top--;
                        else
                            {
                                code[k]=1;
                                break;
                            }
                }
            }
            code[j]=code[j-1]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
    }
    else if(left || up)//仅有一个插头，则延伸插头
    {
        if(left) top=left;
        else top=up;
        if(maped[i][j+1])//右延伸插头
        {
            code[j-1]=0;
            code[j]=top;
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        if(maped[i+1][j])//下延伸插头
        {
            code[j-1]=top;
            code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        else//没有插头
        {
            if(maped[i+1][j] && maped[i][j+1])//下插头和左插头
            {
                code[j-1]=1;
                code[j]=2;
                dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
            }
            //可经过可不经过点则可以保持原样，没有插头
            code[j-1]=code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it],dp[cnt].fas[it]);
        }
    }
    return ;
}void dpblock(int i,int j,int cnt,int n,int m)//障碍格子状态转移
{
    for(int it=1;it<=dp[cnt].size;it++)
    {
        decode(dp[cnt].state[it],code,m);
        if(j==m) shift(code,m);
        dp[cnt^1].add(encode(code,m),dp[cnt].len[it],dp[cnt].fas[it]);
    }
    return ;
}
void solve(int n,int m)
```

```
{
    int cnt=0;
    LL ans,minfas;
    dp[cnt].init();
    dp[cnt].add(0,0,1);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            dp[cnt^1].init();
            if(maped[i][j]==1)
                dpblank(i,j,cnt,n,m);
            else
                dpblock(i,j,cnt,n,m);
            cnt^=1;
/*            for(int it=1;it<=dp[cnt].size;it++)
            {
                decode(dp[cnt].state[it],code,m);
                for(int k=0;k<=m;k++)
                    printf("%d:%d ",k,code[k]);
                printf("fas:%lld\n",dp[cnt].fas[it]);
            }
            printf("\n"); */
        }
    }
    if(dp[cnt].size==0)
        printf("Impossible\n");
    else
    {
        ans=0x3f3f3f3f;
           for(int i=1;i<=dp[cnt].size;i++)
            {
                if(dp[cnt].len[i]<ans)
                {
                    ans=dp[cnt].len[i];
                    minfas=dp[cnt].fas[i];
                }
            }
            printf("%lld %lld\n",ans,minfas);
    }
    return ;
}
int main()
{
    int n,m,kase=0;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        init(n,m);
        solve(n,m);
    }
    return 0;
}
```

3.数论只会GCD

林学姐从初中就开始参加OI(信息学竞赛),并且当时花了好久写出来了.可爱的林学姐最喜欢GCD问题了,如果你做出来这道题,一定能得到林学姐的芳心.

定义函数:对于正整数,H(x)表示x的素数因子的种类.

例如:2=2, H(2)=1, 2的素数因子只有一种; 6=*23, H(6)=2, 6的素数因子有2,3两种; 9=*33, H(9)=1, 9的素数因子只有一种; 12=*223, H(9)=2, 12的素数因子有两种. 每次查询一个范围[L,R],求MAX(GCD(H(i),H(j))),(L<=i<j<<=R).

GCD(a,b)是指a,b两个数最大公约数

Input

多组数据,第一个数为查询次数T,接下来T行每行有一个L,R.

T<=1000000

1<L,R<=1000000

Output

一行,MAX(GCD(H(i),H(j)))

Sample Input

```
2
2 3
2 5
```

Sample Output

```
1

1
```

题解

看起来100w挺大的，然而每个数包含的不同的质数因子个数算下最多才7个(2*3*5*7*11*13*17*19=9,699,690)，写一个线性筛O(n)的复杂度算下每个数不同质数因子个数，或者埃氏筛法求解不超过O(7*n)的时间复杂度。题目时间放宽写埃氏筛法也能过。把100w的数的进行预处理，并且统计下含质因子数量不同的数的个数的前缀和。之后后再进行读入，用常数的时间算出MAX(GCD(H(i),H(j))),(L<=i<j<<=R)即可，一堆if判断下就行。

```cpp
#include <bits/stdc++.h>
using namespace std;

int F[1000010];
bool flag[1000010];
int S[7][10000010];
void init()
{
    for(int i = 2; i <= 1000000; i++) if(!flag[i]){
        F[i]++;
        for(int j = i + i; j <= 1000000; j += i) {
            flag[j] = true;
            F[j]++;
        }
    }
    for(int i = 2; i <= 1000000; i++) {
        for(int j = 0; j < 7; j++) {
            S[j][i] = S[j][i - 1] + (F[i] == j + 1);
        }
    }
}
int main()
{
    init();
    int T, l, r;
    scanf("%d", &T);
    while(T--){
        scanf("%d%d", &l, &r);
        assert(l != r);
        int cnt[8];
        memset(cnt, 0, sizeof(cnt));
        for(int i = 0; i < 7; i++) {
            cnt[i + 1] = S[i][r] - S[i][l - 1];
        }
        int ret = 1;
        if(cnt[2] + cnt[4] + cnt[6]>=2) ret = 2;
        if(cnt[3] + cnt[6] >= 2) ret = 3;
        if(cnt[4] >= 2) ret = 4;
        if(cnt[5] >= 2) ret = 5;
        if(cnt[6] >= 2) ret = 6;
        if(cnt[7] >= 2) ret = 7;
        printf("%d\n", ret);
    }
    return 0;
}
```

4.宣区校内快递点

题目描述

每次都从校外取快递对同学们来说是一件很累的事情，因此快递点的老板们决定在校内设立几个快递点，但要遵守一个规则就是：使得每个宿舍楼都可以通过在道路上最多d公里到达一个快递点。宿舍楼的位置在本题里是假设的，与真实情况无关。

假设全校一共有n个宿舍，编号从1到n，初始时有n - 1条道路相连。所有道路长1公里。最初可以从一个宿舍到使用这些道路连接的任何其他宿舍。本校一共要设立k个快递点，每个快递点位置就是在n个宿舍中的一个。特别是宿舍整个网络结构符合上述快递老板们遵守的规则。另请注意，一个宿舍可以有多个快递点。

然而，冬冬同学感觉没必要建立n-1条路，为了尽可能多地关闭道路来尽量减少道路维护成本，各位大佬快来帮助冬冬找到最多可以关闭的道路。但我们始终都要遵守一个规则，就是使得每个宿舍楼都可以通过在道路上最多d公里到达一个快递点。

Input

第一行包含三个整数n，ķ，和d（2≤ n≤3·10^5，1≤ ķ ≤3*10^5，0≤ d ≤ n- 1）分别表示宿舍的数量，设立快递点的数量，上述规则中距离限制在d公里内。

第二行包含ķ个整数，p1， p2，…， pķ（1≤ pi≤ n） 表示快递点位于宿舍的编号。

在我以下的第n - 1行包含两个整数u和v （1≤u，v ≤n，u≠ v）表示宿舍u和v间建立一条道路，分别为道路1到道路n-1。从1开始编号（1，2，3...) 只有通过道路才可以从一个宿舍到任何其他宿舍。另外，从任何一个宿舍都可以到达d公里内的一个快递点。

Output

输出包括一行，打印一个整数，表示可以被关闭道路的最大数量。

注：此题使用多组输入。

Sample Input

```
6 2 4

1 6

1 2

2 3

3 4

4 5

5 6

6 3 2

1 5 6

1 2

1 3

1 4

1 5

5 6
```

Sample Output

```
1

2
```

题解

这题有点问题。本意是让大家把所有快递点加入到队列中，做一遍bfs。并且对bfs中访问过的点做个标记，然后在bfs中删去边的终点为标记点的边，剩下的就是删去边数最大时的边的数量了。

```
//code by fhs
#include <bits/stdc++.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <set>
#include <queue>

using namespace std;

typedef long long int64;
const int kMaxN= 3e5+10;
const int kInf = (1<<30);

int n, k, d;
bool police[kMaxN];
vector<int> T[kMaxN];
vector<pair<int,int>> edges;
int from[kMaxN];

void bfs() {
    queue<int> Q;
    for (int i = 1; i <= n; ++i) {
        if (police[i]) {
            from[i] = i;
            Q.push(i);
        }
    }

    while (!Q.empty()) {
        int x = Q.front();
        Q.pop();

        for (int i = 0;i < T[x].size();++ i) {
            int y = T[x][i];
            if (from[y] == 0) {
                from[y] = from[x];
                Q.push(y);
            }
        }
    }
}

void solve() {
    memset(police,0,sizeof(police));
    for(int i = 0;i <= kMaxN;++ i)
        T[i].clear();
    edges.clear();
    memset(from,0,sizeof(from));
```

```
    scanf("%d %d %d", &n, &k, &d);
    for (int i = 1; i <= k; ++i) {
        int x;
        scanf("%d", &x);
        police[x] = true;
    }

    for (int i = 1; i < n; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        T[x].push_back(y);
        T[y].push_back(x);
        edges.push_back({x, y});
    }
    bfs();
    vector<int> sol;
    for (int i = 0; i < edges.size(); ++i) {
        auto& edge = edges[i];
        if (from[edge.first] != from[edge.second]) {
            sol.push_back(i+1);
        }
    }
    cout << sol.size() << "\n";
}

int main() {
    int tests = 10;

    for (;tests; --tests) {
        solve();
        // reset();
    }
}
```

5.林喵喵算数

题目描述

给你两个八进制数，你需要在八进制计数法的情况下计算a-b。 如果结果为负数，你应该使用负八进制数表示。

Input

输入的第一个数字为T,表式测试的样例的组数(1 <= T <= 1000)。 每组数据包括两个八进制数a,b.

Output

输出计算结果，同样八进制表示。如果结果为负数，你应该使用负八进制数表示。

Sample Input

```
2

176 7

4 14
```

Sample Output

```
167

-10
```

题解

```cpp
//code by xad
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
typedef long long ll;
ll c[1111];
ll work1(ll t)
{
    ll ans=0,temp=1;
    ll num=t;
    while(num) {
        ans+=num%10*temp;
        temp*=8;
        num/=10;
    }
    return ans;
}

int work2(ll t)
{
    ll num=t;
    int k=0;
    while(num) {
        c[++k]=num%8;
        num/=8;
    }
    return k;
}

int main()
{
    int k,T,i,j,flag;
    ll a,b,num;
    cin>>T;
    while(T--) {
        cin>>a>>b;
        if(a>=b) flag=1;
        else flag=-1;
        num=work1(a)-work1(b);
        if(num==0) {
            cout<<"0"<<endl;
            continue;
        }
        k=work2(num*flag);
        if(flag==-1) cout<<"-";
        for(i=k;i>=1;i--) cout<<c[i];
        cout<<endl;
    }
    return 0;
}
```

6.签到题

题目描述

在计算机网络考试中, 黑帅男神看到一个将IP网络分类的题, 精通C++的他瞬间写出了几行代码成功解决了这个问题

请解析IP地址和对应的掩码，进行分类识别。要求按照A/B/C/D/E类地址归类.

现在假设将所有的IP地址划分为 A,B,C,D,E五类

A类地址1.0.0.0~126.255.255.255;

B类地址128.0.0.0~191.255.255.255;

C类地址192.0.0.0~223.255.255.255;

D类地址224.0.0.0~239.255.255.255；

E类地址240.0.0.0~255.255.255.255

请对输入的IP地址进行分类

Input

多组,第一行是一个T(T<=100000) 接下来T行,每行一个ip地址(不保证正确)

Output

在上面5类中则输出对应类型 不属于上述类型或IP错误则输出”nope”(不含引号)

Sample Input

```
2

222.195.8.207

127.0.0.1
```

Sample Output

```
C

nope
```

题解

签到题

```cpp
//code by xad
#include<cstdio>
#include<iostream>
using namespace std;

int main(){
    int T;
    scanf("%d",&T);
    for(int i=0;i<T;i++){
        int a,b,c,d;
        scanf("%d.%d.%d.%d",&a,&b,&c,&d);
        if(a>=1&&a<=126&&b>=0&&b<=255&&c>=0&&c<=255&&d>=0&&d<=255){
            printf("A\n");
        }
        else if(a>=128&&a<=191&&b>=0&&b<=255&&c>=0&&c<=255&&d>=0&&d<=255){
            printf("B\n");
        }
        else if(a>=192&&a<=223&&b>=0&&b<=255&&c>=0&&c<=255&&d>=0&&d<=255){
            printf("C\n");
        }
        else if(a>=224&&a<=239&&b>=0&&b<=255&&c>=0&&c<=255&&d>=0&&d<=255){
            printf("D\n");
        }
        else if(a>=240&&a<=255&&b>=0&&b<=255&&c>=0&&c<=255&&d>=0&&d<=255){
            printf("E\n");
        }
        else{
            printf("nope\n");
        }

    }

    return 0;
}
```

7.冬冬不喜欢迷宫

题目描述

众所周知，冬冬是一个~~土豪以及游戏狂热玩家~~。最近他趁着暑假，买了许多游戏来消遣。可是当他把所有游戏都通关了以后，感到无敌是多么寂寞，必须靠经典才能满足他。于是他又玩起了走迷宫的小游戏。

走迷宫可以简化为一个矩阵，里面含n*m个格子。有些格子是可以进入并且通过的，有些格子内含障碍物，是不能进入通过它的。我们的任务就是在给定一个起点S和一个终点T后，选择一个不经过障碍物以及走出矩阵边界的方案，把自己控制的角色从S走到T。

然而冬冬身经百战，不知道比其他人高到哪里去了，因此他总想选择最短的路径来走完这个迷宫。可是冬冬的策略总是不能使他达到最优，帅宝宝也告诉他他的路径是第二短的。冬冬很生气。他一直找不出来他的策略哪里有漏洞。

为了安慰冬冬，你能告诉他第二短的路径有多长，以及有多少条这样的路径吗?

Input

多组数据输入，保证不超过2组数据。

第一行是两个正整数n，m，代表矩阵的高度和宽度。其中2≤n,m≤9。

接下来n行m列，给出迷宫每个格子的情况。’S’代表起点，’T’代表终点，’#’代表障碍物，’.’代表可通行区域，其中S一定在左上角，也就是(0,0)的位置；T一定在右下角，也就是(n-1,m-1)的位置。

Output

对于每组数据，输出一行从S到T的第二短路径的长度，以及该长度下的方案数，保证答案在long long范围内。若无从S到T的长度第二短的路径，则输出“Impossible”。

Sample Input

```
5 6
S.....
#.###.
##....
#.##..
#....T
2 9
S.#......
########T
```

Sample Output

```
12 3
Impossible
```

题解

防新生ak题，简单粗暴插头dp，跟第二题的插头dp想法一样，然后最后输出的是步数第二大的状态 以及该状态下的方案数。

```
#include<cstdio>
#include<iostream>
#include<cstring>
#define clr(x) memset(x,0,sizeof(x))
#define clr_1(x) memset(x,-1,sizeof(x))
#define LL long long
#define HASH 10007
#define STATE 1000010
using namespace std;
struct hashmap//hash表 存 状 态
{
    int size;
    int next[STATE],head[HASH];
    LL state[STATE],len[STATE];
    LL fas[STATE];
    void init()//清 空
    {
        size=0;
        clr_1(head);
    }
    void add(LL st,LL length,LL solu)//状 态 加 入
    {
        int k=(st*10000+length)%HASH;
        for(int i=head[k];i!=-1;i=next[i])
            if(st==state[i] && length==len[i])
```

```
            {
                fas[i]+=solu;
                return;
            }
        next[++size]=head[k];
        fas[size]=solu;
        state[size]=st;
        len[size]=length;
        head[k]=size;
        return ;
    }
}dp[2];
int maped[40][40];
int code[40];
char s[100];
void init(int n,int m)//初始化值，读入maped
{
    clr(maped);
    for(int i=1;i<=n;i++)
    {
        scanf("%s",s);
        for(int j=0;j<strlen(s);j++)
        {
            if(s[j]=='.' || s[j]=='S' || s[j]=='T')
            {
                maped[i][j+1]=1;
            }
        }
    }
    return ;
}
void decode(LL st,int *code,int m)//解码，括号序列
{
    for(int i=m;i>=0;i--)
    {
        code[i]=st&3;
        st>>=2;
    }
    return ;
}
LL encode(int *code,int m)//编码，括号序列
{
    LL st=0;
    for(int i=0;i<=m;i++)
    {
        st<<=2;
        st|=code[i];
    }
    return st;
}
void shift(int *code,int m)//左移操作
{
    for(int i=m;i>0;i--)
        code[i]=code[i-1];
    code[0]=0;
    return ;
}
void dpblank(int i,int j,int cnt,int n,int m)//空格子状态转移
{
    int top,left,up;
    if(i==1 &&j==1)//i=1 且 j=1时加入只有下独立插头 和 右独立插头的状态
    {
```

```
        decode(dp[cnt].state[1],code,m);
        if(maped[i][j+1])
        {
            code[j-1]=0;
            code[j]=3;
            dp[cnt^1].add(encode(code,m),dp[cnt].len[1]+1,dp[cnt].fas[1]);
        }
        if(maped[i+1][j])
        {
            code[j-1]=3;
            code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[1]+1,dp[cnt].fas[1]);
        }
        return ;
    }
    if(i==n &&j==m)//i=n 且 j=m时截止单个独立插头的状态
    {
        for(int it=1;it<=dp[cnt].size;it++)
        {
            decode(dp[cnt].state[it],code,m);
            code[j-1]=code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        return ;
    }
    for(int it=1;it<=dp[cnt].size;it++)
    {
        decode(dp[cnt].state[it],code,m);
        left=code[j-1];
        up=code[j];
        if(left && up)//上插头和左插头都存在
        {
            if(left==3 || up==3)//存在一个独立插头
            {
                if(up==2|| left==2)//若另一个插头为右括号插头，则找到对应的左括号插头
变为独立插头
                {
                    top=0;
                    for(int k=j-2;k>=0;k--)
                    {
                        if(code[k]==2)
                            top++;
                        if(code[k]==1)
                            if(top>0)
                                top--;
                            else
                                {
                                    code[k]=3;
                                    break;
                                }
                    }
                }
                else if(up==1 ||left==1)//若另一个插头为左括号插头，则找到对应的右括号
插头变为独立插头
                {
                    top=0;
                    for(int k=j+1;k<=m;k++)
                    {
                        if(code[k]==1)
                            top++;
```

```
                    if(code[k]==2)
                        if(top>0)
                            top--;
                        else
                            {
                                code[k]=3;
                                break;
                            }
                    }
                }
                code[j]=code[j-1]=0;
                if(j==m) shift(code,m);
                dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
            }
            else if(left!=up)//两个括号插头
            {
                if(left==1)//左右括号插头，不允许形成回路
                    continue;
                else//右左括号插头直接去掉
                {
                    code[j]=code[j-1]=0;
                    if(j==m) shift(code,m);
                    dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
                }
            }
            else
            {
                if(left==2)//都是左括号插头，找到对应的第一个右括号插头变为左括号插
头
                {
                    top=0;
                    for(int k=j-2;k>=0;k--)
                    {
                        if(code[k]==2)
                            top++;
                        if(code[k]==1)
                            if(top>0)
                                top--;
                            else
                                {
                                    code[k]=2;
                                    break;
                                }
                    }
                }
                else//都是右括号插头，找到对应的第一个左括号插头变为右括号插头
                {
                    top=0;
                    for(int k=j+1;k<=m;k++)
                    {
                        if(code[k]==1)
                            top++;
                        if(code[k]==2)
                            if(top>0)
                                top--;
                            else
                                {
                                    code[k]=1;
                                    break;
                                }
                    }
                }
```

```
            code[j]=code[j-1]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
    }
    else if(left || up)//仅有一个插头，则延伸插头
    {
        if(left) top=left;
        else top=up;
        if(maped[i][j+1])//右延伸插头
        {
            code[j-1]=0;
            code[j]=top;
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        if(maped[i+1][j])//下延伸插头
        {
            code[j-1]=top;
            code[j]=0;
            if(j==m) shift(code,m);
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
    }
    else//没有插头
    {
        if(maped[i+1][j] && maped[i][j+1])//下插头和左插头
        {
            code[j-1]=1;
            code[j]=2;
            dp[cnt^1].add(encode(code,m),dp[cnt].len[it]+1,dp[cnt].fas[it]);
        }
        //可经过可不经过点则可以保持原样，没有插头
        code[j-1]=code[j]=0;
        if(j==m) shift(code,m);
        dp[cnt^1].add(encode(code,m),dp[cnt].len[it],dp[cnt].fas[it]);
    }
    return ;
}void dpblock(int i,int j,int cnt,int n,int m)//障碍格子状态转移
{
    for(int it=1;it<=dp[cnt].size;it++)
    {
        decode(dp[cnt].state[it],code,m);
        if(j==m) shift(code,m);
        dp[cnt^1].add(encode(code,m),dp[cnt].len[it],dp[cnt].fas[it]);
    }
    return ;
}
void solve(int n,int m)
{
    int cnt=0;
    LL ans,minfas,secans,secminfas;
    dp[cnt].init();
    dp[cnt].add(0,0,1);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            dp[cnt^1].init();
            if(maped[i][j]==1)
                dpblank(i,j,cnt,n,m);
            else
```

```
                dpblock(i,j,cnt,n,m);
            cnt^=1;
/*            for(int it=1;it<=dp[cnt].size;it++)
            {
                decode(dp[cnt].state[it],code,m);
                for(int k=0;k<=m;k++)
                    printf("%d:%d ",k,code[k]);
                printf("fas:%lld\n",dp[cnt].fas[it]);
            }
            printf("\n"); */
        }
    }
    if(dp[cnt].size<=1)
        printf("Impossible\n");
    else
    {
        ans=secans=0x3f3f3f3f;
           for(int i=1;i<=dp[cnt].size;i++)
           {
               if(dp[cnt].len[i]<ans)
               {
                   ans=dp[cnt].len[i];
                   minfas=dp[cnt].fas[i];
               }
           }
           for(int i=1;i<=dp[cnt].size;i++)
           {
               if(dp[cnt].len[i]<secans && dp[cnt].len[i]!=ans)
               {
                   secans=dp[cnt].len[i];
                   secminfas=dp[cnt].fas[i];
               }
           }
           printf("%lld %lld\n",secans,secminfas);
    }
    return ;
}
int main()
{
    int n,m,kase=0;
/*    freopen("11.in","r",stdin);
    freopen("11.out","w",stdout);*/
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        init(n,m);
        solve(n,m);
    }
    return 0;
}
```

8.MCC的考验

题目描述

MCC男神听说新一期的选拔赛要开始了，给各位小伙伴们带来了一道送分题，如果你做不出来，MCC会很伤心的。

给定一个大小为n的非空整数数组,现在定义一种操作，每一步操作会将该数组中的n-1个数同时加1，问最少进行多少步操作后，可以使得数组里的每一个数字都相等。

例如，一个n为3的数组[1,2,3]，最少进行3步操作后，可以变为[4,4,4]

过程如下：[1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]

Input

第一行为一个整数n，代表数组个数。1<=n<=1,000,000

第二行为每个数组的数字a[i],1<=a[i]<=1000

Output

输出需要的最少操作数。

Sample Input

```
3
2 3 4
```

Sample Output

```
3
```

题解

让n-1个数组的数字同时+1，相当于让一个数组的数字-1。计算下每个数组数字与数字最小的数组的数字的差，然后把他们求和后即为操作数。

```cpp
//code by mcc
#include<iostream>
#include<cmath>
#include<vector>
#include<utility>
#include<algorithm>
#include<string>
#include<queue>
#include<set>
#include<map>
#include<stack>
using namespace std ;
int nums[1000010];
int main()
{
/*    freopen("1.in","r",stdin);
    freopen("1.out","w",stdout); */
    int len ;
    cin>>len ;
    for (int i = 0; i < len;i++ ){
       cin>>nums[i] ;
    }
    if (len <= 1){
        cout<<0<<endl ;
        return 0 ;
    }
    int Min = nums[0] ;
    int sum = 0 ;
    for (int i = 0; i < len;i++ ){
        sum += nums[i] ;
        if (nums[i] < Min)
           Min = nums[i] ;
    }
    cout<<sum-Min*len<<endl ;
    return 0;
}
```

比赛链接:http://xcacm.hfut.edu.cn/contest.php?cid=1014