#Name: Md Raihanul Islam Bhuiyan  
#ID:20101239  
  
##########################################  
#Task1  
###########################################  
  
data=open('input1.txt')  
data=data.read()  
data=data.split('\n')  
row= len(data)  
  
for i in range(0,len(data)):  
 data[i]=data[i].split(' ')  
  
column=len(data[0])  
  
visited=[]  
class region:  
  
 def right\_check(self,i,j):  
 if i < row and j + 1 < column and data[i][j + 1] == 'Y':  
 b = {(i, j): [i, j + 1]}  
 return b  
 def bottom\_check(self,i,j):  
 if i + 1 < row and j < column and data[i + 1][j] == 'Y':  
 b = {(i, j): [i + 1, j]}  
 return b  
 def left\_corner(self,i,j):  
 if i + 1 < row and j - 1 < column and data[i + 1][j - 1] == 'Y':  
 b = {(i, j): [i + 1, j - 1]}  
  
 return b  
 def right\_corner(self,i,j):  
 if i + 1 < row and j + 1 < column and data[i + 1][j + 1] == 'Y':  
 b = {(i, j): [i + 1, j + 1]}  
 return b  
  
 def up\_right\_corner(self,i,j):  
 if i - 1 >=0 and j + 1 < column and data[i - 1][j + 1] == 'Y':  
 b = {(i, j): [i - 1, j + 1]}  
 return b  
 def up\_left\_corner(self,i,j):  
 if i - 1 >=0 and j -1>=0 and data[i - 1][j - 1] == 'Y':  
 b = {(i, j): [i - 1, j - 1]}  
 return b  
  
 def graph(self):  
 i=0  
  
 dict={}  
 while i<row:  
 j=0  
 while j< column:  
 if data[i][j] == 'Y' and (i,j) not in visited:  
 self.check(i,j,dict) #to check all the sides recursively in this method  
  
 j = j + 1  
 i = i + 1  
  
 v= [] #visited nodes  
 count=[] #list of the numbers of affected people in different areas  
 for k in dict.keys():  
 if k not in v:  
 affected=0  
 c=self.dfs(dict,v,k,affected)  
 count.append(c)  
  
 print(max(count))  
  
 def dfs(self,dict,v,k,affected):  
 if k not in v:  
 v.append(k)  
 affected+=1  
 #exploring children  
 try:  
 for i in dict[k]:  
 if i not in v:  
 v.append(i)  
 affected+=1  
  
 affected=self.dfs(dict,v,i,affected) #going into the depth recursively  
  
 except:  
 pass  
 return affected #number of affected people in the area  
  
  
 def check(self,i,j,dict):  
 #making dictionary  
 a = self.right\_check(i, j)  
 if a != None:  
 if (i,j) in dict.keys():  
 dict[i,j].append((i,j+1))  
 else:  
 dict[i,j]=[(i,j+1)]  
  
 b = self.bottom\_check(i, j)  
 if b!= None:  
 if (i,j) in dict.keys():  
 dict[i,j].append((i+1,j))  
 else:  
 dict[i,j]=[(i+1,j)]  
  
 c = self.left\_corner(i, j)  
 if c != None:  
 if (i,j) in dict.keys():  
 dict[i,j].append((i+1,j-1))  
 else:  
 dict[i,j]=[(i+1,j-1)]  
  
 d = self.right\_corner(i, j)  
 if d != None:  
 if (i,j) in dict.keys():  
 dict[i,j].append((i+1,j+1))  
 else:  
 dict[i,j]=[(i+1,j+1)]  
 e = self.up\_right\_corner(i, j)  
 if e != None:  
 if (i, j) in dict.keys():  
 dict[i, j].append((i - 1, j + 1))  
 else:  
 dict[i, j] = [(i - 1, j + 1)]  
  
 f = self.up\_left\_corner(i, j)  
 if f != None:  
 if (i, j) in dict.keys():  
 dict[i, j].append((i - 1, j - 1))  
 else:  
 dict[i, j] = [(i - 1, j - 1)]  
  
 visited.append((i,j))  
  
 #updating i and j  
 if a!=None and (i,j+1) not in visited:  
  
 j=j+1  
  
 self.check(i,j,dict)  
 elif b!=None and (i+1,j) not in visited:  
 i=i+1  
 self.check(i, j,dict)  
 elif c!=None and (i+1,j-1) not in visited:  
 i=i+1  
 j=j-1  
 self.check(i, j,dict)  
 elif d!=None and (i+1,j+1) not in visited:  
 i=i+1  
 j=j+1  
 self.check(i, j,dict)  
 elif e!=None and (i-1,j+1) not in visited:  
 i=i-1  
 j=j+1  
 #print('f')  
 self.check(i, j,dict)  
 elif f!=None and (i-1,j-1) not in visited:  
 i=i-1  
 j=j-1  
  
 self.check(i, j,dict)  
 elif a==None and b==None and c==None and d==None and e==None and f==None and data[i][j]=='Y':  
 dict[i,j]=None  
 else:  
 pass  
  
  
a=region()  
a.graph()  
  
  
###########################################  
#Task2  
###########################################  
import collections  
  
data=open('input2.txt')  
data=data.read()  
data=data.split('\n')  
row= int(data[0])  
column=int(data[1])  
data=data[2:]  
  
for i in range(0,len(data)):  
 data[i]=data[i].split(' ')  
  
  
  
visited=[]  
class region:  
  
  
 def right\_check(self,i,j,roots):  
 if i < row and j + 1 < column and data[i][j + 1] == 'H':  
 data[i][j+1]='A'  
 roots.append((i,j+1))  
 def bottom\_check(self,i,j,roots):  
 if i + 1 < row and j < column and data[i + 1][j] == 'H':  
 data[i+1][j]='A'  
 roots.append((i+1, j))  
 def up\_check(self,i,j,roots):  
 if i - 1 >=0 and j < column and data[i - 1][j] == 'H':  
 data[i-1][j]='A'  
 roots.append((i-1, j))  
 def left\_check(self,i,j,roots):  
 if i < row and j - 1 >=0 and data[i][j-1] == 'H':  
 data[i][j-1]='A'  
 roots.append((i, j - 1))  
 def search(self):  
 roots=[]  
 visited=[]  
 human=0  
 i=0  
 while i < row:  
 j = 0  
 while j < column:  
 if data[i][j]=='A':  
 roots.append((i,j))  
 elif data[i][j]=='H':  
 human+=1  
  
 j+=1  
 i+=1  
 time=-1  
 #print(human)  
  
 while len(roots)!=0:  
 #print(roots)  
 old = []  
 for o in roots:  
 # print(o)  
 old.append(o)  
 new = []  
 for alien in roots:  
  
 self.right\_check(alien[0],alien[1],new)  
 self.bottom\_check(alien[0],alien[1],new)  
 self.up\_check(alien[0],alien[1],new)  
 self.left\_check(alien[0],alien[1],new)  
 #print(new)  
 human=human-len(new)  
 for i in new:  
 roots.append(i)  
  
  
 for d in old:  
 roots.remove(d)  
 time+=1  
  
 if time==-1:  
 time+=1  
  
  
 print('Time: '+str(time)+ ' minutes')  
 print(str(human)+' survived')  
  
a=region()  
a.search()