《算法分析与设计》

实 验 报 告

**学 号：2022112566**

**姓 名：倪阳**

**年 级：2022级**

**专 业：计算机科学与技术**

**院 系：计算机与人工智能学院**

**二 0 二四年六月**

**1. 实验目的**

（1） 实验比较数据输入规模对算法执行时间的影响。

（2） 实验比较不同算法对问题求解时间的影响。

（3） 掌握算法时间效率的分析方法。

**2. 实验任务**

（1） 给出采用分支限界法求解该问题时的目标函数，约束条件以及限界

函数。

（2） 给出采用分支限界法求解样例输入时的解空间树。

（3） 分析采用分支限界法求解样例输入时的求解过程，堆结点的定义，堆

结点的值以及堆中元素的变化过程。

（4） 画出采用分支限界法求解样例输入时的搜索空间树。

**3. 实验环境**

**3.1 硬件环境**

（1） 计算机：HUAWEI MateBook 14s

（2） CPU: 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz

（3） RAM：16GB

**3.2 软件环境**

（1） 操作系统：Windows 11 专业版

（2） 开发工具：Visual Studio 2022

**4. 实验步骤及结果**

**4.1 实验预习**

**4.1.1 该问题的自然语言描述法**

创建一个函数来计算长度最长的回路的长度。

遍历每个三角形区域，使用深度优先搜索（DFS）算法找到每个区域的最长回路长度。

在DFS中，记录已访问的三角形区域，遍历相邻的未访问区域，并递归地计算最长回路长度。

返回长度最长的回路的长度。

**4.1.2 给出采用分支限界法求解该问题时的目标函数，约束条件以及限界**

**函数。**

约束条件:

auto ok=[&](int nx, int ny)

{

if(nx<0 || ny<0 || nx>=2\*n || ny>=2\*m || vis[nx][ny] || s[nx][ny]!=' ')

return 0;

return 1;

};

限界函数:

if(ok(nx,ny))

{

if(abs(dx[i])+abs(dy[i])<2)

dfs(dfs,cnt+1,nx,ny);

else

{

if(dx[i]\*dy[i]==1)

{

if(s[x][ny]==s[nx][y] && s[x][ny]=='/')

continue;

dfs(dfs,cnt+1,nx,ny);

}

else

{

if(s[x][ny]==s[nx][y] && s[x][ny]=='\\')

continue;

dfs(dfs,cnt+1,nx,ny);

}

}

}

**4.1.3画出采用分支限界法求解样例输入时的搜索空间树。**

cnt=1, x=0, y=0

cnt=1, x=0, y=3

cnt=2, x=0, y=4

cnt=1, x=0, y=6

cnt=2, x=1, y=5

cnt=3, x=2, y=5

cnt=4, x=3, y=6

cnt=5, x=4, y=6

cnt=6, x=5, y=5

cnt=7, x=6, y=4

cnt=8, x=7, y=3

cnt=1, x=1, y=1

cnt=2, x=2, y=1

cnt=3, x=2, y=2

cnt=4, x=1, y=2

cnt=1, x=1, y=7

cnt=2, x=2, y=7

cnt=1, x=3, y=0

cnt=2, x=4, y=0

cnt=1, x=3, y=3

cnt=2, x=4, y=2

cnt=3, x=5, y=1

cnt=4, x=6, y=1

cnt=5, x=6, y=2

cnt=6, x=5, y=3

cnt=7, x=4, y=4

cnt=8, x=3, y=4

cnt=1, x=5, y=7

cnt=2, x=6, y=6

cnt=3, x=7, y=5

cnt=1, x=7, y=0

cnt=1, x=7, y=7

**4.1.4 程序代码**

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<int,int> pii;

#define endl '\n'

#ifndef ONLINE\_JUDGE

#include "debug.h" //https://github.com/Heltion/debug.h/blob/main/debug.h

#else

#define debug(...) 42

#define debug\_assert(...) 42

#endif

int dx[]={-1,-1,0,1,1,1,0,-1};

int dy[]={0,-1,-1,-1,0,1,1,1};

int main()

{

ios::sync\_with\_stdio(0);

cin.tie(0),cout.tie(0);

int n,m;

cin>>n>>m;

vector<string> s(2\*n);

int i,j,k;

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

{

string str;

cin>>str;

s[2\*i].resize(2\*m);

s[2\*i+1].resize(2\*m);

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

{

if(str[j]=='\\')

{

int x=i\*2, y=j\*2;

s[x][y]=s[x+1][y+1]='\\';

s[x][y+1]=s[x+1][y]=' ';

}

else

{

int x=i\*2, y=j\*2;

s[x][y+1]=s[x+1][y]='/';

s[x][y]=s[x+1][y+1]=' ';

}

}

}

int res=0;

vector<vector<int>> vis(2\*n+2, vector<int>(2\*m+2));

auto dfs=[&](auto dfs, int cnt, int x, int y)->void

{

if(s[x][y]!= ' ')

return;

debug(vector{cnt,x,y});

vis[x][y]=1;

res=max(res,cnt);

for(int i=0;i<8;i++)

{

int nx=x+dx[i];

int ny=y+dy[i];

auto ok=[&](int nx, int ny)

{

if(nx<0 || ny<0 || nx>=2\*n || ny>=2\*m || vis[nx][ny] || s[nx][ny]!=' ')

return 0;

return 1;

};

if(ok(nx,ny))

{

if(abs(dx[i])+abs(dy[i])<2)

dfs(dfs,cnt+1,nx,ny);

else

{

if(dx[i]\*dy[i]==1)

{

if(s[x][ny]==s[nx][y] && s[x][ny]=='/')

continue;

dfs(dfs,cnt+1,nx,ny);

}

else

{

if(s[x][ny]==s[nx][y] && s[x][ny]=='\\')

continue;

dfs(dfs,cnt+1,nx,ny);

}

}

}

}

};

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

for(j=0;j<2\*m;j++)

if(!vis[i][j])

{

debug(pii{i,j});

dfs(dfs,1,i,j);

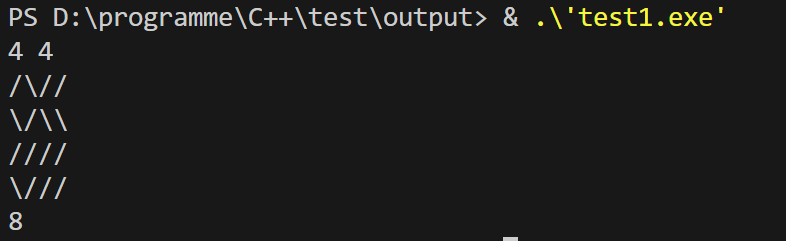
}

cout<<res<<endl;

}

**4.2上机实验**

**4.2.1 运行结果**



**4.2.2** **通过程序的求解过程，分析影响算法时间效率的主要因素有哪些？**

影响算法时间效率的主要因素有以下几个：

1. 算法复杂度：算法的时间复杂度描述了算法运行时间与输入规模之间的关系。通常来说，算法的时间复杂度越低，算法的运行时间越短。

2. 数据规模：输入数据的规模越大，算法的运行时间通常也会越长。因此，对于大规模数据，算法的时间效率尤为重要。

3. 算法实现细节：算法的具体实现细节也会影响算法的时间效率，比如循环次数、递归深度、数据结构的选择等。

**5实验总结**

首先读取了迷宫的大小和迷宫的布局，然后使用DFS来搜索迷宫中的路径，计算最长回路的长度。

在DFS函数中，通过递归的方式遍历迷宫中的每个三角形区域，并根据斜杠的方向来判断是否可以继续前进。在遍历过程中，使用一个二维数组来记录已经访问过的区域，以避免重复访问。

最后，输出最长回路的长度。

总的来说，这段代码实现了一个基于DFS算法的迷宫路径搜索，并成功计算出了迷宫中长度最长的回路的长度。