《算法分析与设计》

实 验 报 告

学 号

姓 名

年 级 2020

专 业 软件工程

院 系 计算机与人工智能学院

二0二二年三月

目 录

实验一 求两个圆相交部分的面积······································1

实验二 算法效率分析与比较·········································11

实验三 穷举法设计与实验···········································11

实验四 分治法设计与实验···········································11

实验五 动态规划算法设计与实验·····································11

实验六 贪心算法设计与实验·········································11

实验七 回溯法设计与实验···········································11

实验八 分支限界法设计与实验·······································11

# 实验8.3机器人搬货物

1. 实验目的

|  |
| --- |
| 1. 理解分支限界法的求解过程。 2. 分析分支限界法的时间复杂度，比较分支限界法算法与其他算法的 时间效率差异。 3. 学会如何利用分支限界法求解具体问题，了解动分支限界法的应用 范围及在实际应用中的局限性。 |

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。  （5） 编写相应的程序。  （6） 调试步骤（5）的程序，验证样例输入时程序执行过程中堆的变化过程是否与步骤（3）的分析结果一致。   1. 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？ |

3.实验环境

|  |
| --- |
| 3.1 硬件环境   1. 计算机：拯救者R7000P 2020H 2. CPU: AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz 3. RAM：16GB   3.2 软件环境   1. 操作系统：Windows11家庭中文版 2. 开发工具：Visual Studio Code |

4.实验步骤及结果

|  |
| --- |
| 4.1 实验预习  4.1.1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。  目标函数：货物最少的移动次数  约束条件：机器人和箱子不能在墙上，机器人不能在箱子上  限界函数：货物移动次数保持最小  4.1.2给出采用分支限界法求解样例输入时的解空间树。  4.1.3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。  广度搜索一个堆所产生的子节点，判断是否符合限界条件，即机器人与箱子是否在墙上，或者机器人是否在箱子上，符合即从该子节点继续生成并继续剪枝，并使该节点压入堆中。  堆节点定义如下：  struct rec  {      int bx, by;      int px, py;      int step;  };  箱子的横纵坐标，机器人的横纵坐标，当前步数  堆元素变化过程如下（部分）：    4.1.4画出采用分支限界法求解样例输入时的搜索空间树。  4.1.5编写相应的程序。  #include <iostream>  #include <queue>  using namespace std;  struct rec  {      int bx, by;      int px, py;      int step;  };  queue<rec> q;  char maze[12][12];  int pstart\_x, pstart\_y, bstart\_x, bstart\_y, target\_x, target\_y;  int visited[12][12][12][12];  *//移动方向   上 下  左 右*  int dx[4] = {0, 0, -1, 1};  int dy[4] = {-1, 1, 0, 0};  int n, m;  int bfs() { *//广搜*      rec temp;      temp.px = pstart\_x, temp.py = pstart\_y; *//机器人*      temp.bx = bstart\_x, temp.by = bstart\_y; *//箱子*      temp.step = 0;      q.push(temp);      while (q.size())      {          rec now = q.front();          cout << " now.step :" << now.step << endl;          if (now.bx == target\_x && now.by == target\_y)              return now.step;          q.pop();          if (visited[now.px][now.py][now.bx][now.by] == 1)              continue;          visited[now.px][now.py][now.bx][now.by] = 1;          for (int i = 0; i < 4; i++)          {              rec next;              next.px = now.px + dx[i];              next.py = now.py + dy[i];              if (next.py >= 0 && next.py < n && next.px >= 0 && next.px < m && maze[next.px][next.py] != '#')              {                  if (next.px == now.bx && next.py == now.by)                  {                      int xx = now.bx + dx[i];                      int yy = now.by + dy[i];                      if (xx >= 0 && xx < m && yy < n && yy >= 0 && maze[xx][yy] != '#')                      {                          next.bx = xx, next.by = yy;                          next.px = now.bx, next.py = now.by;                          next.step = now.step + 1;                          q.push(next);                      }                  }                  else                  {                      next.step = now.step + 1;                      next.bx = now.bx, next.by = now.by;                      q.push(next);                  }              }          }      }      return -1;  }  int main()  {      cin >> n >> m;      for (int i = 0; i < n; i++)          for (int j = 0; j < m; j++)          {             cin >> maze[i][j];             if (maze[i][j] == 'B')                 bstart\_x = i, bstart\_y = j; *//箱子*             else if (maze[i][j] == 'T')                 target\_x = i, target\_y = j; *//目标位置*             else if (maze[i][j] == 'S')                 pstart\_x = i, pstart\_y = j; *//机器人*          }      int ans = bfs();      cout << ans << endl;      system("pause");      return 0;  }  4.2上机实验  4.2.1算法测试  7 11  ###########  #T##......#  #.#.#..####  #....B....#  #.######..#  #.....S...#  ###########  4.2.2测试结果及其分析  调试过程堆变化如截图所示（部分）：      箱子移动6步的堆变化如上图所示，其中只有箱子位置变化时的堆变化，（3）中为机器人位置与箱子位置变化时的堆变化，两者分析过程一致。  4.2.3影响算法时间效率的主要因素  影响算法时间复杂度主要为bfs广度优先搜索函数，还有就是数据规模，以及回溯法中的约束条件，在本题中即为是否在墙上或者是否已到达目的地。数据规模即仓库大小，会直接影响广度搜索的规模。由于本算法采用回溯法实现，所以首先即采用广度搜索，接着按约束条件剪枝即可，所以影响时间效率的因素如上三种。 |

5.实验总结

|  |
| --- |
| 本章学习了分支限界法的分析求解过程，与其他算法的运行时间效率比较以及如何采用分支限界法解决实际应用问题。能够正确理解分支限界法的特点以及在实际应用中如何正确设计并使用分支限界法。 |