**全局搜索算法的实验分析**

**实验一：程序语言和I/O对程序运行时间的影响**

实验目的：了解C++和Python语言程序的运行效率及I/O在程序运行时间中的占比

实验步骤：分别运行2次C++和Python版本的宽度优先搜索解11和12皇后问题的样例程序，一次在显示器上输出解，一次注释掉输出解的语句，比较它们找到全部解的运行时间。全部采用树搜索。

实验结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时间（秒） | C++程序 | | Python程序 | |
| 输出解 | 不输出解 | 输出解 | 不输出解 |
| 11 | 1 | 0 | 4.83 | 4.59 |
| 12 | 2 | 0 | 27.38 | 26.03 |

实验结果分析：

C++是程序在Linux平台，g++9.4.0运行。python是在Ubuntu18.04,python3.8运行（一开始在Windows平台运行，因为ubuntu18.04自带7.5.0版的g++报错）。12皇后问题比11皇后问题计算量大，且远超线性（12/11）。程序输出结果会占用时间，C++远快于Python。

**实验二：深度优先和宽度优先的时间效率比较**

实验目的：比较深度优先和宽度优先算法在求解N皇后问题上的时间效率

实验步骤：分别运行C++和Python版本的宽度优先和深度优先搜索算法求解N皇后问题，注释掉输出解的语句，比较它们找到全部解的运行时间。全部采用树搜索

实验结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时间（秒） | 宽度优先 | | 深度优先 | |
| C++ | Python | C++ | Python |
| 8皇后 | 0 | 0 | 0 | 0 |
| 9皇后 | 0 | 0 | 0 | 0 |
| 10皇后 | 0 | 1 | 0 | 1 |
| 11皇后 | 0 | 5 | 0 | 5 |
| 12皇后 | 0 | 27 | 0 | 26 |
| 13皇后 | 2 | 158 | 2 | 148 |
| 14皇后 | 10 | >300 | 11 | >300 |
| 15皇后 | 65 | >300 | 74 | >300 |
| …… |  |  |  |  |

实验结果分析：宽度优先与深度优先几乎同速，随着皇后数量增加，时间近指数增加。

**实验三：深度优先和宽度优先的空间效率比较**

实验目的：比较深度优先和宽度优先算法在求解N皇后问题上的空间效率

实验步骤：分别运行C++和Python版本的宽度优先和深度优先搜索算法求解N皇后问题，注释掉输出解的语句，比较它们找到全部解时开节点集同时存储的最大节点数。全部采用树搜索

实验结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 空间效率  （节点数） | 宽度优先 | | 深度优先 | |
| C++ | Python | C++ | Python |
| 8皇后 | 573 | 573 | 9 | 9 |
| 9皇后 | 2295 | 2295 | 10 | 10 |
| 10皇后 | 9643 | 9643 | 11 | 11 |
| 11皇后 | 44235 | 44235 | 12 | 12 |
| 12皇后 | 223174 | 223174 | 13 | 13 |
| 13皇后 | 1161451 | 1161451 | 14 | 14 |
| 14皇后 | 6573621 | / | 15 | / |
| 15皇后 | 33933409 | / | 16 | / |
| …… |  |  |  |  |

实验结果分析：随着皇后数量增加，深度优先空间线性增加，宽度优先空间近指数。

**实验四：比较不同算法求解最短路径问题**

实验目的：比较分析一致代价、贪心、A\*算法求解最短路径问题的差异

实验步骤：在根据要求完善代码的基础上，分别运行一致代价、贪心、A\*算法求解最短路径问题的程序（C++或者Python，语言不限），填写并分析实验结果。

实验结果：

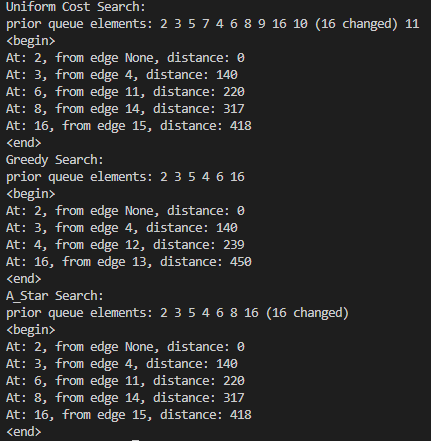
|  |  |  |  |
| --- | --- | --- | --- |
|  | 进入优先队列顺序 | 输出的解路径 | 解路径的花费 |
| 一致代价 | 2-3-5-7-4-6-8-9-16-10-11 | 4-11-14-15 | 418 |
| 贪心搜索 | 2-3-5-4-6-16 | 4-12-13 | 450 |
| A\* | 2-3-5-4-6-8-16 | 4-11-14-15 | 418 |

实验结果分析：

就从算法原理来看，一致代价是启发函数h恒等于0的A\*算法，也就是dijkstra算法。可以证明，A\*算法在启发函数h(n)一致小于节点n到目标的真实距离的条件下，就是完备的，因此作为h恒为0的特殊情况，一致代价算法是完备的。A\*算法是条件完备的，贪心算法不是完备的（但也有可能找到最优）。

就结果来看，存储量大小：一致代价>A\*>贪心；解的正确性：一致代价=A\*>贪心。

1. 截图【一致代价搜索：进优先队列顺序、输出的解路径、解路径的花费】
2. 截图【贪心搜索：进优先队列顺序、输出的解路径、解路径的花费】
3. 截图【A\*搜索：进优先队列顺序、输出的解路径、解路径的花费】



图：一致代价搜索、贪心搜索、A\*搜索输出结果