**全局搜索算法的实验分析**

**(2.5 分) 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。

(2 分) 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 | 1s | 1s | 6.39s | 5.82s |
| 12 | 2s | 2s | 39.03s | 30.89s |

(0.5 分) 实验结果分析：

总体而言，c++程序效率远高于python，因为c++是编译语言，直接生成机器码执性，而python需要逐行解释，更慢。

皇后数提高使得c++运行时间翻倍，python运行时间增长越五倍，python在高时间复杂度的程序上不占优。

同时I/O操作对于c++几乎无影响，因为c++的std::cout默认启用缓冲区，减少系统调用次数；python 的I/O则影响效率，因为print涉及解释器层面的类型转换和系统调用。

**(2.5 分) 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

(2 分) 实验结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 时间（秒） | 宽度优先 | | 深度优先 | |
| C++ | Python | C++ | Python |
| 8皇后 | 0 | 0.05s | 0 | 0.06s |
| 9皇后 | 0 | 0.22s | 0 | 0.24s |
| 10皇后 | 0 | 1.02s | 0 | 1.07s |
| 11皇后 | 1s | 5.12s | 1s | 5.24s |
| 12皇后 | 2s | 29.82s | 2s | 27.97s |
| 13皇后 | 8s | 169.20s | 15s | 158.54 |
| 14皇后 | 51s | >300s | 96s | >300s |
| 15皇后 | >300s  爆内存无输出 | >300s | >300s | >300s |
| …… |  |  |  |  |

(0.5 分) 实验结果分析：

对于n皇后问题这种需要找到所有解的问题，BFS和DFS在剪枝的情况下排除的信息量一致，所以时间效率在于具体的访问节点数。DFS中，一旦发现矛盾就会立即剪枝并回溯到上一层，而BFS中一旦发现矛盾，就不会再展开节点，所以总体分析而言，这二者处理的节点数完全相同，理论上时间应该近似。

在python中二者时间差异的确不大，但在c++中，可以看出DFS在n较大时慢于BFS

**(1.5 分) 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：运行C++ 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

(1 分) 实验结果：

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

(0.5 分) 实验结果分析：

**(3.5 分) 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。

(3 分) 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 | 2-3-5-7-4-6-8-9-16-10-16-11 | 2-3-6-8-16 | 418 |
| 贪心搜索 | 2-3-5-4-6-16 | 2-3-4-16 | 450 |
| A\* | 2-3-5-4-6-8-16-16 | 2-3-6-8-16 | 418 |

(0.5 分)实验结果分析：

都以cost+启发式函数的角度看，一直代价搜索的启发值函数恒为0，既可接纳又一致，与A\*算发的直线距离启发式函数一样，所以一致代价、A\*都是完备，最优的。但是由于A\*又启发式函数引导，其队列展开的总节点数小于一致代价，搜索效率高。

而贪心搜索不考虑cost，仅考虑启发式函数，可能会因启发式函数不准确而丢失实际信息，导致输出次优解。