

**并行原理实践报告**

|  |  |
| --- | --- |
|  |  |
| 姓 名： | 李超峰 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | CS1605 |
| 学 号： | U201614633 |
| 指导教师： | 陆枫 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2019 年 7 月28 日

目录

[实验任务二 3](#_Toc15337349)

[2.1任务要求 3](#_Toc15337350)

[2.2实验内容 3](#_Toc15337351)

[2.3实验结果 7](#_Toc15337352)

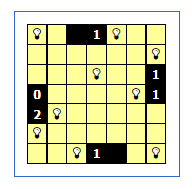
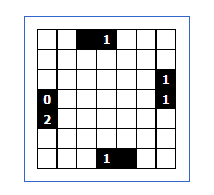
# 实验任务二

## 2.1任务要求

Akari问题有时又被称为Light up或者Beleuchtung，源于日本逻辑解密游戏系列Nikoli，同属于Nikoli谜题的除Akari之外还有Sudoku(数独)和Kakuro(数谜)等。

游戏规则很简单。点灯游戏的棋盘是一张方形格网，其中的格子可能是黑色也可能是白色。游戏目标是在格网中放置灯泡，使之能照亮所有的白色方格。如果一个方格所在的同一行或同一列有一个灯泡，并且方格和灯泡之间没有黑色格子阻挡，那么这个方格将被灯泡照亮。同时，放置的每个灯泡不能被另一个灯泡照亮。  
某些黑色格子中标有数字。这些数字表示在该格子四周相邻的格子中共有多少个灯泡。

Akari问题的示意如图1所示。



Akari的初始状态与解

## 2.2实验内容

2.2.1使用回溯法解决Akari问题

回溯法是一种迭代算法。在回溯法中，首先将问题的解决分为若干步，其次通过枚举每一步的选择构造解空间树。在此基础上通过深度优先搜索遍历此解空间树，若当前节点的局部解不能构造出全局解，则向上回溯，否则向下扩展。重复此步骤直到找到全局解。

回溯法的关键点在于：

问题可分步并且可枚举每一步的选择。

可以迅速的判断出当前局部解可否构造出全局解。

若问题可分为N步，每一步有M种选择，易知其时间复杂度为O(m^n)O(m​n​​)，虽然和穷举法有着相同的时间复杂度，但因为是逐步搜索并不断摒弃局部解，因此在实际应用中比穷举法效率高的多。

要求首先对问题进行划分，并根据每一步构造解空间，进而设计合理的数据结构与程序结构对问题进行求解，并在此基础上对算法以及程序逻辑进行优化和改进，最后进行问题的分析、讨论和展望。关键步骤如下。

对问题进行划分  
根据黑色方格中数字的大小，按从大到小的顺序进行排序，并将“在一个有数字的黑色方格周围放置‘车’”定义为一步。

选择枚举  
枚举每一步的选择。  
若黑色方格中的数字为4，则其相邻的周围格子的“灯泡”的放置方式为1种，即上下左右均放置一个“灯泡”。  
若黑色方格中的数字为3，则其相邻的周围格子的“灯泡”的放置方式为4种，即在左右下、下右上、右上左、上左下的格子中放置“灯泡”。  
若黑色方格中的数字为2，则其相邻的周围格子的“灯泡”的放置方式有6种，即在上左、上下、上右、左下、左右、下右的格子中放置“灯泡”。  
若黑色方格中的数字为1，则其相邻的周围格子的“灯泡”的放置方式有4种，即在上、下、左、右的格子中放置“灯泡”。  
若黑色方格中的数字为0，则其相邻的周围格子的“灯泡”的放置方式有1种，即所有周围格子均不放置车。

构造解空间  
根据上述讨论构造解空间，初始状态为解空间树的根节点，从编号最大的黑色格子开始尝试填入“灯泡”，填入后判断是否为一个可行解，若为可行解，则解空间向下进行分枝，否则向上进行回溯。解空间树的结构大致如图2所示。

程序设计与实现  
根据回溯法的伪代码和上述算法的设计思路，建立合适的数据模型与程序结构，编写程序求解问题，同时记录程序的运行时间。

讨论与改进  
分析算法的时间负责度与空间复杂度，根据某一用例和计算机的计算能力，估计程序运行时间，并将该时间与实际运行时间进行比较。  
同时讨论与优化程序结构和数据结构，以求达到更快的程序执行速度和更少的内存占用量。

2.2.2使用并行法解决Akari问题

串行算法并行化的过程中的核心问题是问题分解和解除数据相关的问题。所谓的问题分解问题即将串行算法采取分治的理念分成可并行计算的子问题，解除数据相关的问题就是采取一定的冗余策略或锁策略使各子问题的私有数据之间不相互影响。

回溯法是易于并行化的典型算法之一，在Akari问题的回溯算法中，可以清楚的认识到回溯的具体路线的选择对产生解的正确性没有影响，因此该回溯法中每个节点的计算过程以及节点的分裂过程均是可以并行计算的。同时由算法的性质决定了每个解空间节点各自数据不会相互影响，没有任何数据相关。

使用pthread进行多线程编程  
POSIX thread 简称为pthread，是一个支持POSIX的标准线程，该标准定义内部API用于创建和操纵线程。POSIX是指可移植性操作系统接口（Portable Operating System Interface），被大多数操作系统支持（如Unix，Linux，Windows），其编程方式简单易用并且可移植。  
在Linux下使用pthread多线程需要在使用gcc/g++的时候加上lpthread编译选项，以便使链接器能动态链接到pthread库。

使用fork()进行多进程编程  
在Linux环境下，一个现有进程可以调用fork函数创建一个新进程。由fork创建的新进程被称为子进程（child process）。fork函数被调用一次但返回两次。两次返回的唯一区别是子进程中返回0值而父进程中返回子进程ID。  
子进程是父进程的副本，它将获得父进程数据空间、堆、栈等资源的副本。注意，子进程持有的是上述存储空间的“副本”，这意味着父子进程间不共享这些存储空间。  
使用fork()系统调用可在Linux实现并行编程，但相对于pthread来说创建新的进程开销要大于新的线程

2.2.3使用改进的并行回溯法求解Akari问题

并行计算中的粒度与性能

多线程应用的并行任务工作量大小（粒度）会对其并行性能产生很大影响。在分解一项应用使之适用于多线程处理时，编程时通常采用的方法是从逻辑上将问题分割成尽量多的并行任务，或者在并行任务内根据共享数据与执行顺序决定进行哪些必要的通信。由于分割任务、将任务分配给线程以及在任务之间进行数据通信（共享）涉及到一定的成本，编程者通常需要聚合或整合分割的任务，用于避免随之产生的开销，尽量实现应用高效运行。通过聚合分割的任务可确定并行任务的最佳粒度。粒度通常与工作负载在线程之间的均衡程度有关。尽管平衡大量小型任务的工作负载更容易，但这样做却可能导致通信和同步等方面的并行开销过高。此时，编程者可以通过将小型任务整合成一项任务，增加每项任务的粒度（工作量）来减少并行开销。并行回溯法求解Akari问题时的并行粒度划分方法。并行粒度的划分实质上是一个将工作负载（任务）与工作单元（线程或进程）对应的过程，在本实验过程中，对于Akari问题的粒度划分方法有以下几种：每个解空间的节点分裂均产生新任务

1.当且仅当解空间树节点第一次分裂的时候产生新任务

2.当且仅当解空间树节点在数字“3”的时候分裂时产生新任务

## 2.3实验结果

运行结果如下所示

