Dancing Links

舞蹈链

问题1：

集合s有n个成员，现在有m个子集，每个子集包含一些成员，每个成员都属于集合s。在m个子集中选择一些子集组成子集的集合t，使t中包含的成员可以覆盖集合s，即s中所有成员都属于t中的某个子集。

重复覆盖：集合s中的任意成员x至少属于t中的一个子集，允许同时属于两个及以上的子集。例如集合，在子集、、中选择即可重复覆盖s。

精确覆盖：集合s中的任意成员x属于且只属于t中的一个子集，不能出现x不属于t中的任何子集，或者x同时属于t中两个及以上的子集。例如集合，在子集、、中选择即可精确覆盖s。

给定集合s和m个子集，求出其重复覆盖和精确覆盖。

重复覆盖解法：

遍历集合s中每个成员x，若其尚未被包含在t中，则在m个集合中寻找一个包含x的子集加入t中，重复该步骤即可获得重复覆盖。

精确覆盖解法：

求解精确覆盖的算法称为X算法，将集合s中n个成员看作列，将m个子集看作行，组成一个的矩阵d。若子集（其中）包含某个成员（其中），则；若不包含则。对于集合有个成员，还有个子集、、、、、的情况，如图所示：



在这个矩阵d上进行回溯法（我个人认为回溯法和深度优先递归搜索的本质是一样的）即可得到精确覆盖，过程如下：

1. 从0开始遍历集合s中每个成员，对于成员0，遍历所有子集，找到第一个满足的子集，选择该子集作为精确覆盖中的一个子集，包含的其他成员，其他子集不能再包含该子集中出现的成员，因此删掉其他包含的子集和，将也删掉；



1. 从1开始遍历集合s中剩下的成员，对于成员1，遍历剩余子集，找到第一个满足的子集，选择该子集作为精确覆盖中的一个子集，包含的其他成员，其他子集不能再包含该子集中出现的成员，因此删掉其他包含的子集，将也删掉，这时矩阵d已经被删空，说明已经找到了精确覆盖，即为和；



1. 现在考虑失败情况，若一开始依次选择从1开始遍历集合s中剩下的成员，对于成员1，遍历剩余子集，找到第一个满足的子集，选择该子集作为精确覆盖中的一个子集，包含的其他成员，其他子集不能再包含该子集中出现的成员，因此删掉其他包含的子集，将也删掉，这时矩阵d已经被删空，说明已经找到了精确覆盖，即为和；

回溯法的递归结束条件是矩阵d为空（当矩阵d为空时递归结束），每次递归时选择矩阵中的一列（其中），遍历矩阵d中的所有行，找到一行（其中）满足，选择该行。由于精确覆盖的要求，其他包含该行中成员的行，都不能再选择，因此将所有包含该行成员的其他行删掉，然后将行删掉。重复这个操作试图将矩阵d删空，在选取包含成员的行时，可能有多个选择，若选择其中一行而无法将d删空，则在递归函数中返回这一层，重新选择下一行。

A\*搜索算法结合了广度优先搜索、Dijkstra算法和启发式搜索，能够有效提高搜索效率。与DFS和BFS这种误差别搜索不同，启发式搜索会设置一个评价函数来计算每个节点的搜索代价（或到目标的距离），优先搜索那些离目标最近的点，从而提高搜索效率。

A\*算法的评价函数为，其中x是一个节点，表示x点到end的估计距离，表示从beg到x点的距离，表示从x点到end的估算距离。在A\*算法的等待队列中，总是优先选取最小的点进行搜索。

在下面这个的二维方格s中，其中黑色的格子代表不能通过。从移动到，设置open队列、close队列和g分数表。x点和end点的估算距离。过程如下：



g分数表：表示点到beg点所需要的最小距离

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 将beg加入open队列并染绿，初始化，其中，；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 |  |  |  |  |
| 1 | 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 从open队列中取出f值最小的点（唯一的），比较，将它加入close队列并染红，它周围的、不属于open队列，将这两点加入open队列并染绿，令、的父节点为，计算；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 |  |  |  |
| 1 | 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 从open队列中取出f值最小的点（多个f值相等的点中可以随机选择一个，考虑到目标的估算距离h值，在f值相等的基础上应该优先选择h值最小的，本解答在选取时有疏漏），比较，将它加入close队列并染红，它周围的不属于open队列并染绿，将该点加入open队列，令的父节点为，计算；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 | 3 |  |  |
| 1 | 1 |  | 3 |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 从open队列中取出f值最小的点，比较，将它加入close队列并染红，它周围的、不属于open队列，将该两点加入open队列并染绿，令、的父节点为，计算；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 | 3 |  |  |
| 1 | 1 |  | 3 |  |  |  |
| 2 | 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 从open队列中取出f值最小的点，比较，将它加入close队列并染红，它周围的不属于open队列，将该点加入open队列并染绿，令的父节点为，计算；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 | 3 | 4 |  |
| 1 | 1 |  | 3 | 4 |  |  |
| 2 | 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 从open队列中取出f值最小的点，比较，将它加入close队列并染红，它周围的、不属于open队列，将该两点加入open队列并染绿，令、的父节点为，计算；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 | 3 | 4 |  |
| 1 | 1 |  | 3 | 4 |  |  |
| 2 | 2 |  | 4 |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

1. 从open队列中取出f值最小的点加入close队列并染红，它周围的不属于open队列，已经属于open队列，将加入open队列并染绿，令的父节点为，计算，计算，新的路径并不比老的路径更短，不必更新点的父节点；
2. 每一步从open队列中取出f值最小的（目前open队列中所有点的f值都相同，可以随机选择任意点，不过考虑到目标的估算距离h值，在f值相等的基础上应该优先选择h值最小的）点x，比较x与end，(a)若open队列为空，算法结束没有找到end点，(b)若则算法结束找到end点，(c)继续寻找，将x点加入close队列并染红，对于x点周围的其他节点y有以下可能：(a)若y点属于close队列为红色则直接跳过该点，(b)若y点不属于close队列，不属于open队列，将y点加入open队列并染绿，父节点设置为x，计算、、值，(c)若y点不属于close队列，属于open队列，说明y点已经被访问过，重新计算以x点为父节点的y的、、值，若更小则更新y点的信息和父节点为x点，若并不比原本的路径更短则保持现在的y点不变；



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 |  | 3 | 4 | 5 | 6 |
| 2 | 2 |  | 4 |  |  | 7 |
| 3 | 3 |  |  |  | 9 | 8 |
| 4 |  |  |  |  | 10 | 9 |
| 5 |  |  |  |  |  | 10 |

1. 跳过中间重复的步骤，最终情况如上图所示，open队列中包含end点，算法结束，beg点到end点的距离为10，路径通过节点的父节点指针回溯回去；

对于的二维方格s，A\*搜索的过程并不是向四周均匀发散开的，而是沿着f值和h值最小的方向移动，最坏情况下时间复杂度为。

问题2（八数码问题）：

对于的矩阵，x点可以与上下左右的相邻点交换位置，除此之外不能随意改变位置，将该矩阵变成下面状态：。

求最少变换次数以及变化经过，即从起点状态beg到终点状态end的路径。

解法：

与之前问题不同，本问题将每种矩阵状态看作一个节点，是一种时间上的状态搜索。

从移动到，设置open队列、close队列和g分数表。x点和end点的估算距离，即对于同一个位置，若则h值加1，否则加0，其中。当x点与end点相同时，x点中每个位置的值都和end点相同。过程如下：

1. 将beg点加入open队列，g分数表中；
2. 从open队列中取出f值最小的（唯一的）点，将它加入close队列，该点的x与上下左右交换位置后的状态有，这3个状态可以看作该点的相邻点（和问题1中二维方格的上下左右4个相邻格子类似），这3个状态不属于open队列和close队列，将这3个点加入open队列并设置父节点都为并计算g值；
3. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有2个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该点加入open队列，设置父节点为并计算g值；
4. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有2个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该点加入open队列，设置父节点为并计算g值；
5. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有4个点，其中属于close队列，跳过该点，不属于open队列和close队列，将这3点加入open队列，设置父节点为并计算g值；
6. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将这2点加入open队列，设置父节点为并计算g值；
7. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将这2点加入open队列，设置父节点为并计算g值；
8. 从open队列中取出f值最小的点（open队列中存在多个的点，但是该点h值最小因此优先选择），将它加入close队列，该点的x与上下左右交换位置后的状态有2个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该点加入open队列，设置父节点为并计算g值；
9. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将这2点加入open队列，设置父节点为并计算g值；
10. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有2个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该点加入open队列，设置父节点为并计算g值；
11. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该2点加入open队列，设置父节点为并计算g值；
12. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该2点加入open队列，设置父节点为并计算g值；
13. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该2点加入open队列，设置父节点为并计算g值；
14. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有3个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该2点加入open队列，设置父节点为并计算g值；
15. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有4个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该3点加入open队列，设置父节点为并计算g值；
16. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有2个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该点加入open队列，设置父节点为并计算g值；
17. 从open队列中取出f值最小的点，将它加入close队列，该点的x与上下左右交换位置后的状态有4个点，其中属于close队列，跳过该点，不属于open队列和close队列，将该3点加入open队列，设置父节点为并计算g值；
18. 每一步从open队列中取出f值最小的（f值相同时优先选h值最小的）点x，比较x点与end点，(a)若open队列为空，则没有找到end，算法结束，(b)若则找到end点，算法结束，(c)继续寻找，将x点加入close队列，x点相邻的其他节点y有以下可能：(a)若y点属于close队列则直接跳过该点，(b)若y点不属于close队列，不属于open队列，将y点加入open队列，父节点设置为x，计算、、值，(c)若y点不属于close队列，属于open队列，说明y点已经被访问过，重新计算以x点为父节点的y的、、值，若更小则更新y点的信息和父节点为x点，若并不比原本的路径更短则保持现在的y点不变；

对于八进制数码问题，A\*搜索在最坏情况下时间复杂度为。