华南理工大学

《课程名称》课程实验报告

实验题目：The Tree Search Algorithm

姓名： 谭演锋 学号： 202130100456

班级： 计算机全英联合班 组别： 无

合作者： 无

指导教师： 颜小洋

|  |
| --- |
| **实验概述** |
| 【实验目的及要求】  实验目的：  Solve the 8-Puzzles Problem by Tree Serach Algorithms——using Kruskal’s algorithm and Prim’s algorithm. Compare the differences between two algorithms.  实验要求：  Given a random sequence for the numbers from 1 to 8 like below and use the tree search algorithm to find if there is the goal node or not after moving them.  Use different strategies to perform this algorithm, such as: the Breadth-first search (BFS), the Depth-first search(DFS) and the Best-first search(BFS) Search strategies etc.  And compare the result difference solved by them.  The template should be used for all kinds of data type, such as: integer, real, double, etc. in the program.  Programs should be made by Object-Oriented Programming (OOP) method.  The results should be compared with ones of other algorithms, such as: Straight selection sort, insert sort, etc., and draw the graph to find their differences.  Write down the report in which there should be the execution results of the program.  【实验环境】  操作系统：Windows XP |
| **。** |
| 【实验过程】   1. 实验步骤： 2. Definition of class and function：   **Class:** MySearch  **Data member:**  shifts: A data that type is vector<vector<int>> which stores the possible moving for each possible position of 0  visit: A data that type is unordered\_set<string> which stores the visited state so that no it will not repeat.  init\_state: The initial state of the 8-puzzle problems  goal\_state: The goal state of the 9-puzzle problems  step: Recording the running step for the solving algorithm.  run\_time: Recording the running time for the  lowerBound: Using for Branch and Bound Searching algorithm to cut off the finding steps.  **Member functions:**  searchHelp: Used to select a Tree Search Algorithm to solve the problem.  getSearchTime: Return run\_time.  getSearchStep：Return step  setInitState: set initial state.  setGoalState: set goal state.  bfs: Run the breadth first search.  dfs: Run the depth first search.  best\_first: Run the best first search.  Astar: Run the A\* Search.  cmp: The evaluation function of the Best First Search and return true of false for the comparison of two state. The evolution value is the right position in the current state compared to the goal state. Each right position, the value of the current state adds 1.  move: Move the 0 to the other position according to the current position.   1. Combine the MySearch with the UI by Qt   **Class:** MainWindow  **Data member:**  MainWindow \*ui : create UI widget  searchType: select one algorithm for searching  init\_state = "203184765" : initial state  goal\_state = "123804765": goal state  MySearch\* s: the class of the searching  Table: label list of the view of 8-puzzle  buttonList : button list  **Member functions:**  on\_pushButton\_clicked: push button then initialize the initial state  on\_pushButton\_2\_clicked: push button then run the selected algorithm. If no algorithm is selected give a warning “"please select searching methods”. If no result, give a message with error “it doesn't have a solution”. If has a result, give a information” solution has been found”  settable: Set the 8-puzzle table according to the initial state.  (3)Run program in the main function:  Create an object of the class Mysearch. Then call the member function searchHelp to run the different Tree Search Algorithms. In the searchHelp function will print the running time of running the algorithm and each algorithm will   1. 实验数据：   The UI is followed    Initial State: 203184765  Goal State: 123804765   1. 实验主要过程：   There are 4 main procedures of the experiment:  Breath First Search:  Input: A string of initial state  Output: The step to find out the goal state of fail to find the goal state  Initial queue  Push the initial state into the queue  Insert the initial state to the visited set  While the queue is not empty do  Begin  Current state:= the first element in queue  Step++  If Current state = goal state then  Print the step  End the function  For each possible move of the Current state do  Begin  Move to the next state  If the next state is not in the set of visited then  Add the next state to the queue  Add the next state to the visited state  End  End    Depth First Search:  Input: A string of current state, the depth of current state in the searching tree  Output: The step to find out the goal state of fail to find the goal state  Initial queue  Step++  Add the current state to the visited state  If depth> lowerBound then  End the function  If Current state = goal state then  Print the step  End the function  For each possible move of the Current state do  Begin  Move to the next state  If the next state is not in the set of visited then  Recursive call the dfs(next state,depth+1)  End  Best First Search  Input: A string of initial state  Output: The step to find out the goal state of fail to find the goal state  Initial vector  Push back the initial state into the vector  Insert the initial state to the visited set  While the vector is not empty do  Begin  Current state:= the last element in vector  Step++  If Current state = goal state then  Print the step  End the function  For each possible move of the Current state do  Begin  Move to the next state  If the next state is not in the set of visited then  Add the next state to the queue  Add the next state to the visited state  End  Sort the vector according to the evaluation function “cmp”  End    Branch and Bound Search  Input: A string of current state, the depth of current state in the searching tree  Output: The step to find out the goal state of fail to find the goal state  Initial queue  Step++  Add the current state to the visited state  If depth> lowerBound then  End the function  If Current state = goal state then  Print the step  lowerBound = min(lowerBound,depth);  End the function  For each possible move of the Current state do  Begin  Move to the next state  If the next state is not in the set of visited then  Recursive call the dfs(next state,depth+1)  End   1. 实验结果：           Another initial state: |
| **小结** |
| By creating the UI interface of the 8-puzzles question, I learned how to make the UI with Qt. More importantly, by improving my code basing on the previous experiment, I have the deeper learning about the 8-puzzles and the tree search algorithms. Not all the 8 puzzles questions have a solution. However, most of the time the Best First Search And Breadth First Search can find the solution if there is a solution. Instead, DFS and Branch and Bound Search can not find a result in some cases with solution.    Here's a comparison of four common algorithms used to solve the 8-puzzle: Depth-First Search (DFS), Breadth-First Search (BFS), Best-First Search (BFS), and Branch and Bound.  1. Breadth-First Search (BFS):  BFS explores all the neighboring nodes of the current node before moving to the next level of the search tree. It uses a queue data structure to keep track of the nodes to be explored. BFS guarantees to find the optimal solution, i.e., the shortest path to the goal state, if the edge costs are uniform. It has higher memory requirements compared to DFS since it needs to store all the generated nodes until the goal state is found. BFS may not be efficient in terms of time complexity, especially if the branching factor of the search tree is high. In the example of the experiment, it takes lots of time and step to find the result since its goal state is deep.  2. Depth-First Search (DFS):  DFS explores a path as far as possible before backtracking. It traverses the depth of the search tree first. DFS does not use any heuristics or cost functions to guide the search. It may find a solution quickly if it's located closer to the root, but it's not guaranteed to find the optimal solution. DFS has low memory requirements since it only needs to store the path from the root to the current node. However, DFS can get stuck in loops or take a long time to find a solution if the search space is large and the goal state is deep. In the example of the experiment, it takes lots of time and step to find the result since its goal state is deep.  .  3. Best-First Search (BFS):  Best-First Search uses a heuristic function to guide the search by evaluating the potential of each node to lead to the goal. It expands the node with the lowest heuristic value first, prioritizing nodes that are closest to the goal state. The choice of heuristic function greatly affects the efficiency and optimality of Best-First Search. Best-First Search is not guaranteed to find the optimal solution unless an admissible and consistent heuristic is used. It can be memory-intensive, especially if the search space is large and the heuristic values need to be stored for each node. In the experiment, the heuristic function is “cam”. The best first search is like the greedy method. Therefore, sometimes it can find the solution with the fastest speed. For example, in this experiment it only need 5 step.  4. Branch and Bound:  Branch and Bound is an algorithmic technique that combines elements of DFS and BFS. It explores the search space by branching out from the current node and bounds the search based on certain criteria. Branch and Bound uses a cost function to estimate the cost of reaching the goal state and prunes branches that exceed the current cost bound. It guarantees to find the optimal solution if the cost function is consistent, and the pruning strategy is correctly implemented. The memory requirements of Branch and Bound depend on the size of the search tree and the number of nodes generated during the search. In the example, though the Branch and Bound method has the same step to the DFS, since its searching way is similar, it has the lower time cost. That is because the Branch and Bound end many unnecessary searching by the lowerbound.  In summary, DFS is simple but may not find the optimal solution. BFS guarantees optimality but can be memory intensive. Best-First Search uses heuristics to guide the search but requires a good heuristic function for optimality. Branch and Bound combines elements of DFS and BFS and guarantees optimality if implemented correctly. The choice of algorithm depends on the specific requirements of the problem, the search space characteristics, and the available computational resources. 8-puzzles is a good way to examine and compare different tree search algorithm with different initial states. |
| **指导教师评语及成绩** |
| 评语：  成绩：           指导教师签名：                                                 批阅日期： |