**CS570 Artificial Intelligence**

**Spring 2012**

**Project 1a**

**Sanqing Yuan**

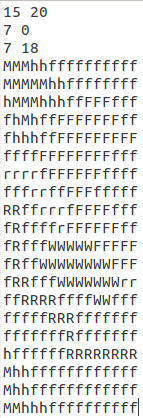
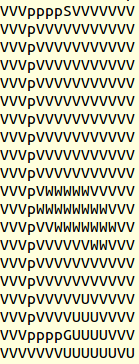
1. **Algorithm**

For this project, the algorithm of breadth first search was used. To do this, using C++, I created a class of queue by single linked list, a multi-branches tree (at most 4 branches for each node) using double linked list. The steps of the algorithm are described below:

* read the data from map.txt file, including the width and height of map, the start point , the goal and the whole map, which is stored in a 2d character array. Another 2d character array was created to store the information regarding if the cells are visited and the cells on the path. This program is good for any rectangle map with max size of 100x100.
* an objective of the class queue and an objective of class tree were created. The queue was used to store the frontier list with the policy of FIFO, and the tree was for tracing from goal back to the start point.
* For the searching, the left, down, right and up side of current cell (first in the queue) were checked to see if it is the goal. If the answer is yes, then game over, we find the goal and we can use a member function from the tree to find the path from start to the goal. If the answer is no, other conditions have to be checked, including if it was visited, if it is outside of map and if it is water. If at least one questions’ answer is yes, this cell will be skipped and not push into the queue and add to the tree. Otherwise, this cell is push into the queue and added into the tree. After one cycle (left, down, right and up, always the same order), we checked if the queue is empty(search fails), if not, we pop the queue and start another cycle until we find the goal. I used recursion here which cause glibc problem, but before this happened, we can obtain all information we need. During the recursion, one of the 2d arrays store the changes of each cells, for example from unvisited to visited, or unvisited to path.
* Using one of the 2d array and the member function from tree class, I can draw the map after the search, which shows unvisited cells (U), visited cells (V), water (W) , path (p), start point(S) and the goal (G). Also it calculated the number of visited cells and the steps of the path.

1. **Results**

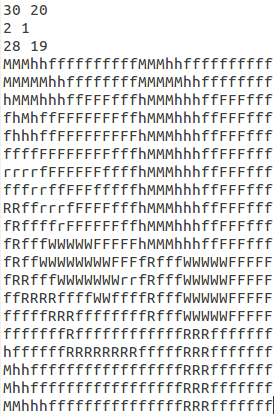
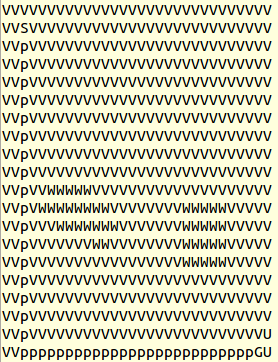
Breadth first search is able to find the shortest path, but it explores a large amount of cells. Take the map provided by Dr. Soule as example:

**Figure 1** Original map **Figure 2** Breadth first search [unvisited cells (U), visited

cells (V), water (W) , path (p), start point(S) and the goal (G)]. It

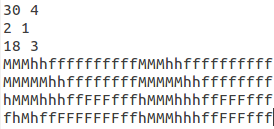
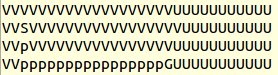
has 236 visited cells and 24 steps of path.

**Figure 3** Original map **Figure** **4** Breadth first search [unvisited cells (U), visited

cells (V), water (W) , path (p), start point(S) and the goal (G)]. It

has 511 visited cells and 42 steps of path.

**Figure 5** Original map **Figure** **6** Breadth first search [unvisited cells (U), visited

cells (V), water (W) , path (p), start point(S) and the goal (G)]. It

has 58 visited cells and 16 steps of path.

1. **Discussion**

If the goal is in the map, breadth first search can always find the goal, so it is complete. Without considering the cost, breadth first search always finds the shortest path, which means it is optimal. However, the time and space complexity are not that good. In this case, I have to create a tree to store the information of cells that are visited. As the node expanding, the tree gets bigger and bigger. As shown in figure 1-6, breadth first search explores most of the area of the map if the start point and goal are far from each other. It also takes a lot of time to visit all those positions.