15-381: Artificial Intelligence
Assignment 1: Search Algorithms

Handed out: Thursday, September 6th, 2001
Due: Tuesday, September 18th, 2001 (before class)

Introduction
This is your first programming assignment (aren't you excited?). In this assignment, you will implement (and, hopefully, gain an appreciation for) unguided and heuristic search. More specifically, you will implement a forward search engine that supports multiple search strategies and use this implementation to find solutions for a simple puzzle.

Also included as the second part of the homework is a short written assignment. Print out and turn in the last two pages of the homework with your answers written in.

The Puzzle
You are given a row containing 2*N* tiles arranged in 2*N* + 1 spaces. There are *N* black tiles (B), *N* white tiles (W), and a single empty space. The tiles are initially in an arbitrary ordering. Your goal is to arrange the tiles such that all white tiles are positioned to the left of the black ones, and one black tile is in the rightmost position. The goal position of the empty space is not specified.

Tiles can be moved to the empty space when the empty space is at most *N* cells away. Hence there are at most 2*N* legal moves from each state. The cost of each move is the distance between the tile and the empty space to which it moves (1 to *N*). "Circular" moves (i.e. moves wrapping from one side of the row to the other) are not permitted.

To illustrate the tile domain, consider the trivial case where *N* = 2.

Initial state may look like:

W	B	-	B	W

There are 4 successors to the initial state:

W	-	B	B	W
W	B	B	-	W
W	B	W	B	-
-	B	W	B	W

There are 4 goal states:

W	W	B	-	B
W	W	-	B	B
W	-	W	B	B
-	W	W	B	B

The Assignment
Your implementation should read in an arbitrary starting state for tile problems containing up to 21 positions (10 black, 10 white, and 1 space). Each input file contains a single line specifying

the initial state using `B` to represent a black tile, `W` to represent white, and `x` to represent the space. For example, one possible starting state for a 7-position problem is the string:

```
WBBWxWB
```

You should use one operator, `move x`, to specify which tile to move into the current empty position. `x` is the position of the card (starting from left to right). Note that you only need specify which card to move since it will always move into the single empty position. In the absence of any other heuristic function that determines which tile to move, all legal moves should be considered in increasing order (i.e. from 0 to 2*N* + 1; move 1 occurs before move 3).

Your program should print out the initial state followed by each move operation selected and its successor state (i.e. both the operator path and the state path). At the end, it should print the cost (or the total number of positions that the blank had to move) and the number of nodes expanded (not just the ones to the solution path). For example, given an initial state `BWxBW`, your program might print the following lines:

```
WBxBW
move 5 WBWBx
move 4 WBWxB
move 2 WxWBB
5
20
```

The number of nodes expanded in this case, 20, is made up (yours should do the right thing).

Search Strategies

Your program must support three different search strategies: breadth-first search (BFS), depth first search (DFS), and A* (A-star). Each of these search strategies should be selectable at run-time by using a command-line keyword (see Logistics section for more details).

Note that you should be implementing this using a single function, not three different functions (one for each search strategy). Hint: use a next-state queue containing states that have been generated but not-yet visited as your primary data structure. In such a data structure, BFS implies a FIFO discipline, DFS implies a LIFO or "stack" discipline, and A* implies a priority queue sorted according to the heuristic function. Also note that you will need a means of avoiding loops by checking whether or not a state has already been visited.

A* combines the cumulative cost so far `g(n)` with a guess about the cost of getting to the solution from the current state `h(n)`, using the evaluation function `f(n)= g(n)+h(n)`. Define `g(n)` and `h(n)` such that `g(successor(n)) ≥ g(n)` and `h(n)` is an admissible heuristic (you can't use h = 0).

Logistics

The program should be written in C or C++. You must turn in documented source code, as well as an executable compiled under Linux. Your executable should accept command-line arguments specifying the puzzle name and the search type:

`./search <inputfile> <bfs|dfs|astar>`

The output for the program should be placed in a file called `<puzzle>_<searchtype>.txt`. For example, if Best First Search is run on a puzzle called `puz1`, then the output of the search would be placed in `puz1_bfs.txt`.

You can find sample test files in:

/afs/andrew.cmu.edu/course/15/381/homework/Project1/samples/

The files you should submit are:

- Readme.txt: including
  - a list of all your source files
  - documentation of any problems you had as well as any help you attained (describing who and the nature of the help). Remember: it is much better to seek help, learn the material, and turn in the assignment complete and on time than to fail to complete the assignment.
  - a description of the heuristic you used for the A* search. You must show the formula you used, but may describe it in plain English. Explain why the heuristic is admissible.
- Your source code
- An executable called `search`, compiled and tested under Linux.

These files should be submitted in the course directory at:

/afs/andrew.cmu.edu/course/15/381/handin/<your_id>/p1/

These directories will be created a few days before the due date and will be closed when class starts on Tuesday, September 18th, 2001. If you are submitting your project late (try to avoid this for the first assignment!), let us know and we will keep the folder open. E-mail Desney (desney@cs.cmu.edu) when you do submit it.

Grading

Your assignment will be graded by running your code on new initial states and examining the traces you produce. Grades will be determined by:

- Conformance to the execution and output formats specified above
- Correctness of your search implementation
- Whether or not your program finds a valid (and when required, optimal) path

Further Information

Please post to the class bboard: `cmu.cs.class.cs381` or contact Desney (desney@cs.cmu.edu) or Hakan (lorens@cs.cmu.edu) if you have further questions.

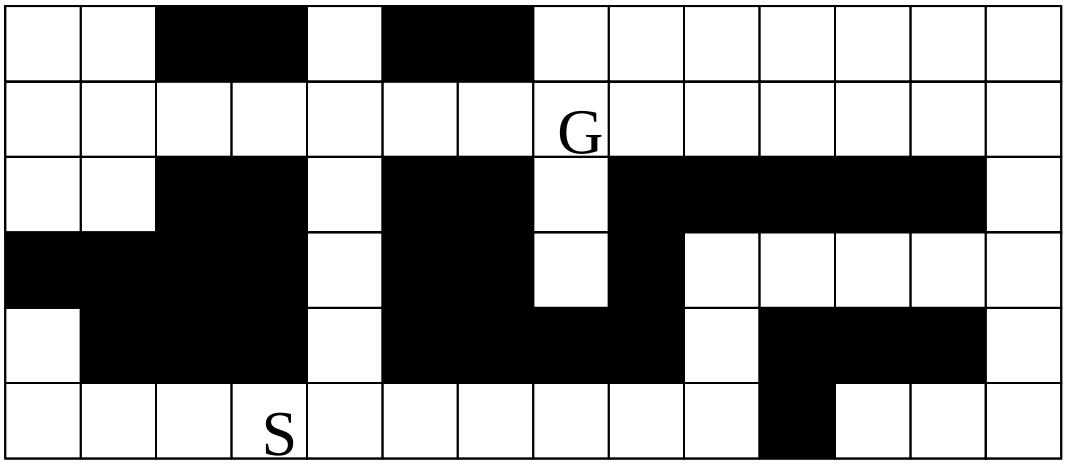
Written Assignment 1

Problem 1
In your programming assignment, we limited the size of the puzzles to 21 (10 of each color and a blank space). Briefly discuss how each of the search techniques would scale to larger puzzles and how performance would be affected. Think of a puzzle that is a million and one pieces large if that helps you.

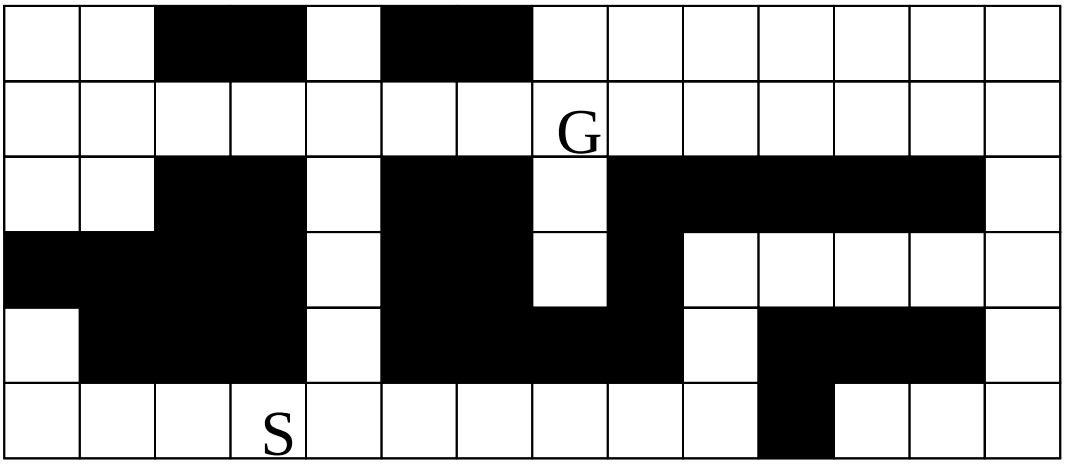
Problem 2
Imagine a scenario with a robot trying to navigate in the following maze from the start position (S) to the goal position (G). At each step, the robot can move in one of the four compass directions. The robot contemplates alternatives in the following order:

Move South
Move East
Move North
Move West

a. Mark the set of states that are expanded during the search, in the order they are expanded, by putting a "1" in the first state, a "2" in the second, and so on (Put a "1" in the cell marked S). Assume that the search is a Depth First Search (DFS). Use the version of DFS that avoids loops by never re-expanding a state that is on the current path.



b. Using the same notation, mark the set of states that Breadth First Search (BFS) would expand, in the order in which they are expanded.



c. Again using the same notation, mark the set of states that A-star search would expand, in the order that they are expanded. Assume we use the following heuristic:

H(x) = (Horizontal number of cells between x and G) +
0.999 * (Vertical number of cells between x and G)