CECS 451: Artificial Intelligence
Project 2: Uninformed Search
Dr. Roman Tankelevich

Varderes Barsegyan
016163470
Due: 9/16/16

Project 2: Uninformed Search

In this project we explored the elementary concept of uninformed search. In uniformed searches, the agent has no prior knowledge of the state space of the problem and thus must find the goal and the set of actions that lead to that goal by examining its surroundings. In particular, BFS and DFS are well-known techniques for this paradigm. I have demonstrated them with two examples: a letter tree and the 8-puzzle board game.

The source code for everything below can be found here. Furthermore, I worked on this project on my own. Actually, my roommate, who is a kinesiology major, provided some moral support.

Task 1: DFS Trace

The DFS trace is given below. We see that in this algorithm the state tree is delved into "depth first". This can either be implemented recursively or using a Stack data structure. In my example, I perform DFS recursively. However, in the 8-puzzle I use a Stack.

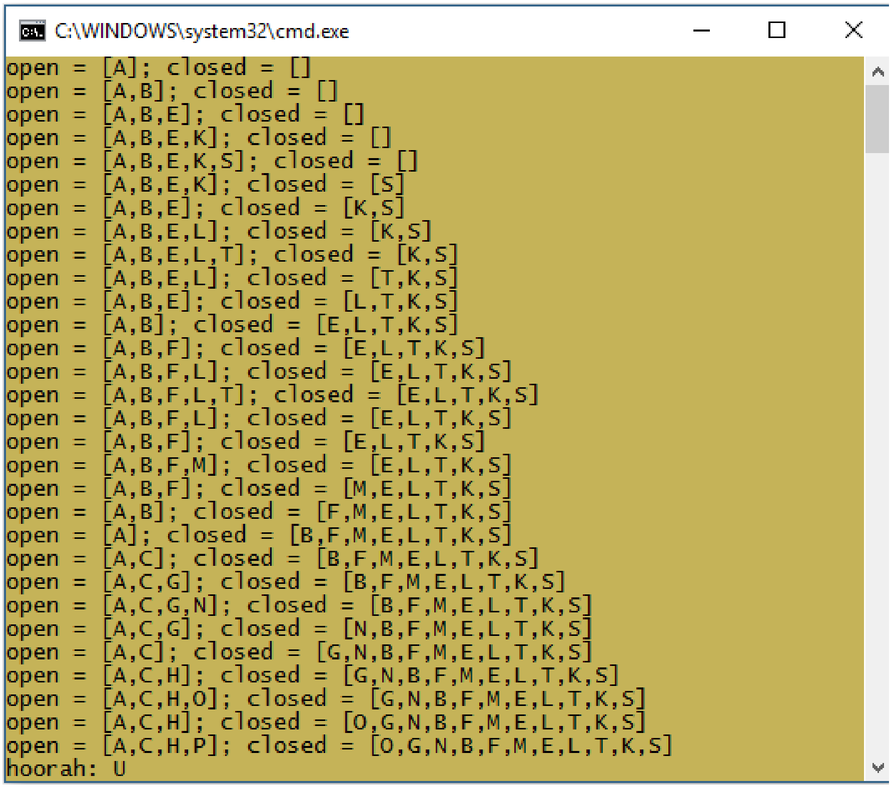
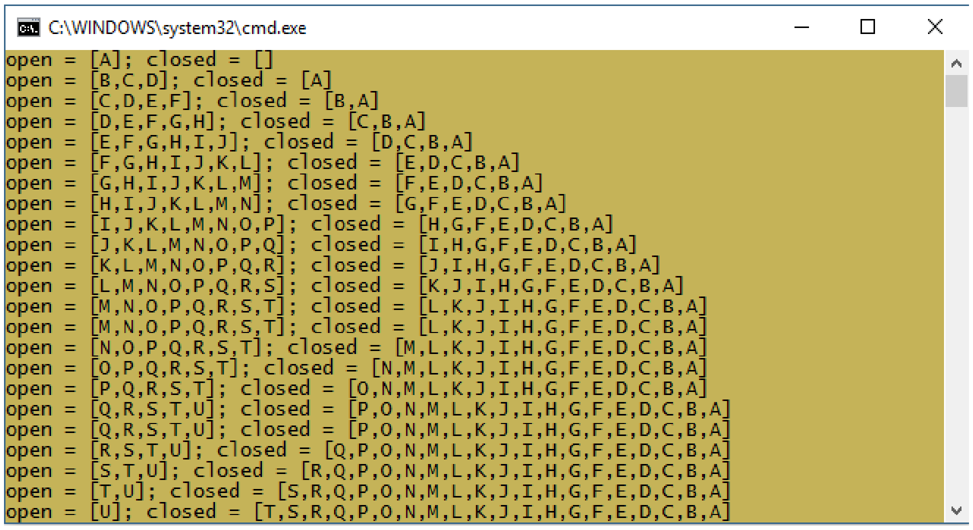

```
open = [A]; closed = []
open = [A,B]; closed = []
open = [A,B,E]; closed = []
open = [A,B,E,K]; closed = []
open = [A,B,E,K,S]; closed = []
open = [A,B,E,K]; closed = [S]
open = [A,B,E]; closed = [K,S]
open = [A,B,E,L]; closed = [K,S]
open = [A,B,E,L,T]; closed = [K,S]
open = [A,B,E,L]; closed = [T,K,S]
open = [A,B,E]; closed = [L,T,K,S]
open = [A,B]; closed = [E,L,T,K,S]
open = [A,B,F]; closed = [E,L,T,K,S]
open = [A,B,F,L]; closed = [E,L,T,K,S]
open = [A,B,F,L,T]; closed = [E,L,T,K,S]
open = [A,B,F,L]; closed = [E,L,T,K,S]
open = [A,B,F]; closed = [E,L,T,K,S]
open = [A,B,F,M]; closed = [E,L,T,K,S]
open = [A,B,F]; closed = [M,E,L,T,K,S]
open = [A,B]; closed = [F,M,E,L,T,K,S]
open = [A]; closed = [B,F,M,E,L,T,K,S]
open = [A,C]; closed = [B,F,M,E,L,T,K,S]
open = [A,C,G]; closed = [B,F,M,E,L,T,K,S]
open = [A,C,G,N]; closed = [B,F,M,E,L,T,K,S]
open = [A,C,G]; closed = [N,B,F,M,E,L,T,K,S]
open = [A,C]; closed = [G,N,B,F,M,E,L,T,K,S]
open = [A,C,H]; closed = [G,N,B,F,M,E,L,T,K,S]
open = [A,C,H,O]; closed = [G,N,B,F,M,E,L,T,K,S]
open = [A,C,H]; closed = [O,G,N,B,F,M,E,L,T,K,S]
open = [A,C,H,P]; closed = [O,G,N,B,F,M,E,L,T,K,S]
hoorah: U
```

Figure 1: Depth first seach trace created in C#

I also recreated the BFS trace in the assignment description by implementing it myself. We can clearly see the difference in the two traversal techniques. In particular, the space complexity of

BFS is much larger: $O(b^d)$, where b is the number of branches from each node, and d is the depth of the tree. Whereas in DFS, the space complexity is $O(bm)$. m is the maximum depth of the tree. However, they have the same time complexity: $O(b^d)$.



```
open = [A]; closed = []
open = [B,C,D]; closed = [A]
open = [C,D,E,F]; closed = [B,A]
open = [D,E,F,G,H]; closed = [C,B,A]
open = [E,F,G,H,I,J]; closed = [D,C,B,A]
open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]
open = [G,H,I,J,K,L,M]; closed = [F,E,D,C,B,A]
open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]
open = [I,J,K,L,M,N,O,P]; closed = [H,G,F,E,D,C,B,A]
open = [J,K,L,M,N,O,P,Q]; closed = [I,H,G,F,E,D,C,B,A]
open = [K,L,M,N,O,P,Q,R]; closed = [J,I,H,G,F,E,D,C,B,A]
open = [L,M,N,O,P,Q,R,S]; closed = [K,J,I,H,G,F,E,D,C,B,A]
open = [M,N,O,P,Q,R,S,T]; closed = [L,K,J,I,H,G,F,E,D,C,B,A]
open = [M,N,O,P,Q,R,S,T]; closed = [L,K,J,I,H,G,F,E,D,C,B,A]
open = [N,O,P,Q,R,S,T]; closed = [M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [O,P,Q,R,S,T]; closed = [N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [P,Q,R,S,T]; closed = [O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [Q,R,S,T,U]; closed = [P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [Q,R,S,T,U]; closed = [P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [R,S,T,U]; closed = [Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [S,T,U]; closed = [R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [T,U]; closed = [S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
open = [U]; closed = [T,S,R,Q,P,O,N,M,L,K,J,I,H,G,F,E,D,C,B,A]
```

Figure 2: Trace for breadth-first search, recreated from the assignment example

Task 2: DFS and BFS Implementation

Instead of writing up the pseudocodes, I went ahead and implemented DFS and BFS in C#.

```csharp
public void DepthFirstSearch(Node<T> start, Node<T> goal)
{

    OpenedNodes.Add(start);

    Console.Write($"open = {Print(OpenedNodes)}; ");
    Console.WriteLine($"closed = {Print(ClosedNodes)}");

    foreach (Node<T> successor in start.Successors)
    {
        if (successor != goal)
        {
            DepthFirstSearch(successor, goal);
        }
        else
        {
            Console.WriteLine("hoorah: " + successor.Value);
        }
    }

    OpenedNodes.RemoveAt(OpenedNodes.Count - 1);

    if (!ClosedNodes.Contains(start))
    {
        ClosedNodes.Insert(0, start);
    }

    Console.Write($"open = {Print(OpenedNodes)}; ");
    Console.WriteLine($"closed = {Print(ClosedNodes)}");

}
```

Varderes Barsegyan
016163470
Due: 9/16/16

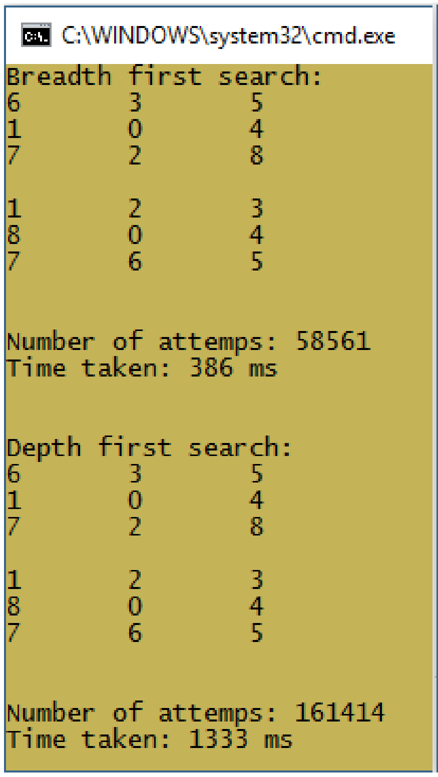
```
public void BreadthFirstSearch(Node<T> start, Node<T> goal)
{
    Queue<Node<T>> queue = new Queue<Node<T>>();
    queue.Enqueue(start);

    OpenedNodes.Add(start);
    Console.Write($"open = {Print(OpenedNodes)}; ");
    Console.Write($"closed = {Print(ClosedNodes)}\n");

    while(queue.Peek() != goal)
    {

        Node<T> current = queue.Dequeue();
        OpenedNodes.Remove(current);

        if (!ClosedNodes.Contains(current))
        {
            ClosedNodes.Insert(0, current);
        }

        foreach (var successor in current.Successors)
        {

            queue.Enqueue(successor);

            if (!OpenedNodes.Contains(successor))
            {
                OpenedNodes.Add(successor);
            }

        }

        Console.Write($"open = {Print(OpenedNodes)}; ");
        Console.WriteLine($"closed = {Print(ClosedNodes)}");

    }
}
```

Task 3: 8-Puzzle

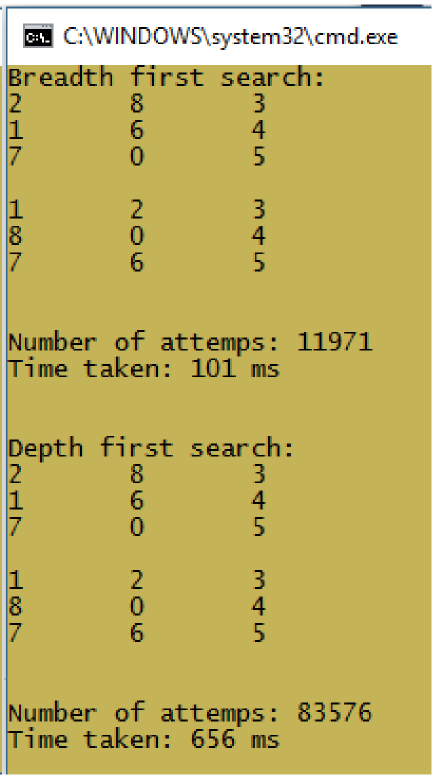
The results from three runs are given below. I ran a random example in the first one and the project example in the second. The third is an unsolvable puzzle where the maximum number of iterations are reached. This is because the initial set up of an 8-puzzle could render it unsolvable. The maximum number of traversals such that the board has a unique set up every time is $\frac{9!}{2} = 181440$. After this point any iteration create duplicate, and if a solution has not been reached then the puzzle cannot be solved. In fact, the solvability of an 8-puzzle can be determined before running any search algorithms. This is done by counting the inversions. However, I just stuck to expending the configurations.

We see that in both cases, BFS took both less time as well as less attempts. This makes sense, as a DFS might causes a trajectory of configurations that are getting further from the goal, yet BFS provides more paths to work with. However, it is not always the case that BFS performs better. Once in a while DFS outperforms BFS, but it is quite rare.

CECS 451: Artificial Intelligence | Varderes Barsegyan
Project 2: Uninformed Search | 016163470
Dr. Roman Tankelevich | Due: 9/16/16

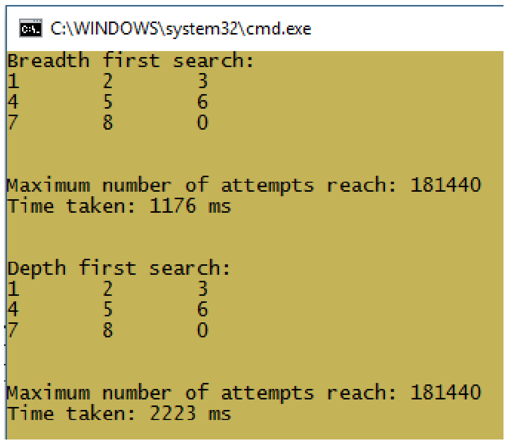


```
C:\WINDOWS\system32\cmd.exe
Breadth first search:
6         3         5
1         0         4
7         2         8

1         2         3
8         0         4
7         6         5


Number of attemps: 58561
Time taken: 386 ms


Depth first search:
6         3         5
1         0         4
7         2         8

1         2         3
8         0         4
7         6         5


Number of attemps: 161414
Time taken: 1333 ms
```



```
C:\WINDOWS\system32\cmd.exe
Breadth first search:
2         8         3
1         6         4
7         0         5

1         2         3
8         0         4
7         6         5


Number of attemps: 11971
Time taken: 101 ms


Depth first search:
2         8         3
1         6         4
7         0         5

1         2         3
8         0         4
7         6         5


Number of attemps: 83576
Time taken: 656 ms
```

Figure 3: A run with a random initial setup and a run using the project example, respectively



```
C:\WINDOWS\system32\cmd.exe
Breadth first search:
1       2       3
4       5       6
7       8       0


Maximum number of attempts reach: 181440
Time taken: 1176 ms


Depth first search:
1       2       3
4       5       6
7       8       0


Maximum number of attempts reach: 181440
Time taken: 2223 ms
```

Figure 4: A run with an unsolvable initial grid

In addition to using a stack and queue for DFS and BFS, respectively, I used a hash set to store every unique iterated configuration of the board. When a new grid is created, I first check if it is in the hash set. If it is, I discard it. If not, I add it to the stack or queue. This ensures that no duplicates will be checked.