# Homework #2: DFS, DFID

Submit Assignment



**Due** Apr 9 by 11:59pm **Points** 100 **Submitting** a file upload
**Available** after Apr 2 at 12a



HW #2

This homework is to be com ou can discuss the assignment with other students in thinking about how to solve t all programming and other portions of the assignment must be entirely your own work. You may not show your code to other students. Undergraduates may work in teams with up to one other student, while graduate students must work individually. If you are working with another student, you need to designate this by forming a group in Canvas.



**Task:**

In this assignment you will implement two search algorithms for the Sliding-Tile Puzzle code you wrote in Homework #1.



**Algorithms:**



1) Depth-First Iterative Deepening



Implement the DFID algorithm. This algorithm should include a depth-limited depth-first search which is called repeatedly with larger depth bounds until the goal is found. The depth bounds will increase by one each iteration. The algorithm does not need to return the path found, but should report the total number of nodes expanded. (For debugging purposes you may want to print the number of nodes expanded in each depth-limited iteration.)



For efficiency purposes, you should keep track of the parent of each state and avoid recursing back to the parent. (This should happen in the DFID algorithm, not in the successor/operator generation.) Be sure to generate operators and modify a single copy of the current state with Apply/Undo move.

2) Breadth-First Search

Implement a simple breadth-first search. This algorithm should assume (albeit incorrectly) that none of the input domains contain any cycles. But, with each state you should store information to avoid generating the parent at the next level. The search should terminate when the goal is found, but does not need to return the path that was found. After the goal is found, it should report the total number of nodes expanded.

**Testing:**

Test each of these algorithms by performing a short random walk (10 moves) and then use each algorithm to find a solution to the resulting state. (For comparison purposes you should make sure that you test the same states with both algorithms.)

Measure the memory and time required by your program with both
length of the random walk increases.

**What to turn in**:

Your submission should include the two algorithms, and a sample
the random walk states. Along with the code also submit a text file
program and a description of
students should include a for
results.

**Sample Code**:

Your algorithm should look s

```
class MySearchAlgorithm
{
public:
    MySearchAlgorithm();
    // GetPath returns if the goal was found
    bool GetPath(environment &e, state &start, state &goal);
    // Returns the total nodes expanded by the last GetPath call
    uint64_t GetNodesExpanded();
private:
     // ...
};
```

**Homework 2: BFS, DFID**

| Criteria | | | |
|---|---|---|---|
| DFID Implementation<br>*This algorithm should include a depth-limited depth-first search which is called repeatedly with larger depth bounds until the goal is found. The depth bounds will increase by one each iteration. The algorithm does not need t d, but should report the total number of nod uld keep track of the parent of each state an to the parent. (This should happen in the DF e successor/operator generatio operators and modify a single copy of th pply/Undo move.* | 40<br>Ur<br>- F | | |
| BFS Implementation<br>*This algorithm should assume (albeit incorrectly) that none of the input domains contain any cycles. But, with each state you should store information to avoid generating the parent at the next level. The search should terminate when the goal is found, but does not need to return the path that was found. After the goal is found, it should report the total number of nodes expanded.* | 40<br>Ur<br>F | | |
| Testing<br>*Test each of these algorithms by performing a short random walk (10 moves) and then use each algorithm to find a solution to the resulting state. For comparison purposes you should make sure that you test the same states with both algorithms. Measure the memory and time required by your program with both the DFID and BFS searches as the length of the random walk increases.* | 15.0 pts<br>Undergraduate - Full Marks | 10.0 pts<br>Graduate - Full Marks | 0.0 pts<br>No Marks | 15.0 pts |
| README File<br>*Along with the code also submit a text file containing a sample run of your testing program and a description of the largest size problem that could be solved with each algorithm.* | 5.0 pts<br>Graduate & Undergraduate - Full Marks | | 0.0 pts<br>No Marks | 5.0 pts |
| GRAD ONLY - Write up<br>*Graduate students should include a formal writeup that formally describes the algorithms, the domain, and the final results.* | 20.0 pts<br>Graduate - Full Marks | | 0.0 pts<br>No Marks | 20.0 pts |
| | | | Total Points: 120.0 | |