ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

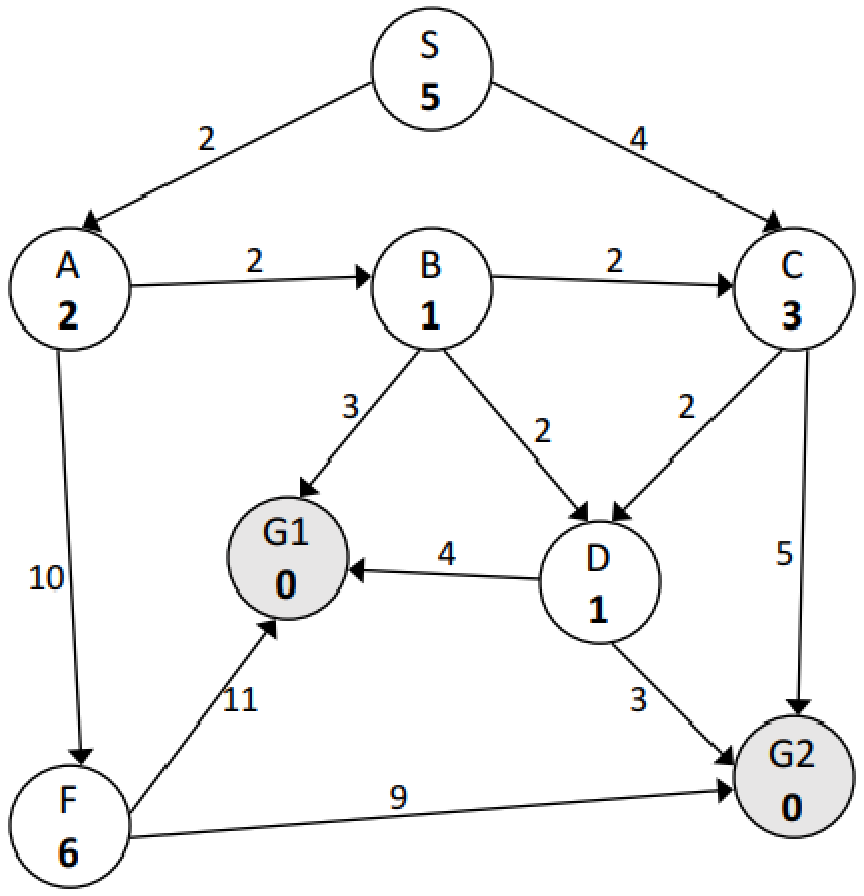
Project 1

ARTIFICIAL INTELLIGENCE

Anna-Aikaterini Kavvada | NKUA | 30/10/2018

Table of Contents

Problem 1



Below follows the solution of each search problem asked for the above graph. For every problem the goal node we reach is presented and then follows the instances of the fringe from the beginning of the search to the point we reach our goal state.

a) **Breadth First Search**

Goal Node: G2

Fringe: S ➔A, C ➔ C, B, F ➔ B, F, D, G2 ➔ F, D, G2, G1 ➔ D, G2, G1 ➔ G2, G1 ➔ G1

First, we push the node S, which is not a goal state in the fringe (a FIFO queue). Then, we pop S and call for its successors A and C. A is being popped out of the fringe and we check whether it is a goal state. As it follows, it is not a goal state. We continue by calling its successors and pushing them in the fringe (C, B, F), and we pop the next node which is C. We check if C is a goal state and call for its successors which we push in the fringe (B, F, D, G2). This process continues till we reach this instance of our fringe: G2, G1. The next node, G2, when popped out of the queue and is checked for being a goal node, the answer will be yes. Thus, we've reached the end of our search.

(Note: we keep all the nodes that have been popped from the fringe in a list, so that we do not double check them in the fringe. For example, when we have this fringe instance: B, F, D, G2 where B is popped, checked for being a goal and calling its successors, to this point one of its successors is C which has already passed from the fringe, so there is no need to push it back. Apart from that successor D already exists in the fringe, so from all of B's successors just G1 is being pushed in the FIFO queue)

b) **Depth First Search**

Goal Node: G2

Fringe: S ➔ A, C ➔ A, D, G2 ➔ A, D

Due to the stack we use as a fringe for the DFS, we pop from the stack the last node we pushed in it and check it for being a goal state or not. So, the algorithm reaches G2 node and finds that it is a goal state.

c) **Iterative Deepening Search**

Goal Node: G2

Fringe: S ➔ A, C ➔ B, F, D, G2 ➔ G1

IDS checks all the nodes that finds in a particular depth of the graph. So, in the first iteration it comes up with S. When S is popped from the fringe found not to be a goal state it calls for its successors A and B, which are being pushed in the fringe. Then A and B are popped checked and call for their successors and push them in the fringe (in case a successor found has already been checked we ignore it) and the fringe instance now is: B, F, D, G2. After that when these nodes are popped we find that G2 is our goal state. G1 has been pushed though in the fringe since it is B's successor which was checked and called for its successors earlier than G2.

d) **Greedy Best First Search**

Goal Node: G1

Fringe: S ➔ A, C ➔ C, B, F ➔ C, F, D, G1 ➔ C, F, G1, G2 ➔ C, F, G2

The algorithm chooses from the fringe the node with the smaller cost. So, the nodes popped from the fringe are in order A, B, D, G1. Since G1 is a goal state, when is popped from the fringe and checked the search is over.

e) **A***

Goal Node: G1

Fringe: S (0) ➔ A (2), C (4) ➔ B (4), C (4), F (12) ➔ C (4), D (6), G1 (7), F (12)

➔ D (6), G1 (7), G2 (11), F (12) ➔ G1 (7), G2 (9), F (12) ➔ G2 (9), F (12)

The A* algorithm chooses the best node which has the smaller cost from the fringe. This time the cost is the cost of the path we have followed to reach a certain node. Thus we end up with the above instances and eventually come up with the solution G1.

Problem 2

If the branching factor is b = 1, we have:

a) **Breadth First Search:** O (b^d) = O (1)
b) **Depth First Search:** O (b^d) = O (1)
c) **Iterative Deepening Search:** O(d)

Problem 3

1. **False**
   Depth-first search sometimes might stand lucky and expand fewer nodes than A* search with an admissible heuristic. E.g., it is logically possible that sometimes, by good luck, depth-first search may march directly to the goal with no back-tracking.
2. **True**
   Let h*(N) be the cost of the optimal path from N to the goal state node of the 8-tile problem. The heuristic function h(N) is admissible if:
   $$0 \leq h(N) \leq h^*(N)$$
   Since an admissible heuristic function is always optimistic if n is a goal node then we have h(n) = 0.
3. **True**
   If there exists a goal it occurs at finite depth *d* and will be found in $O(b^d)$ steps. Complete means "will find a goal when one exists". This statement does not

imply also optimal, which means “will find a lowest-cost goal when one exists.” Thus, the step costs are irrelevant to the algorithm being complete.

4. **True**
   The UCS algorithm is like Breadth First algorithm with the addition of taking into consideration the path cost of every node. The UCS algorithm expands the node that finds in the fringe, which has the smallest path cost. Thus, we can say that the UCS is a special case of the BFS algorithm.
5. **False**
   Best First algorithm is a special case of the Breadth First algorithm. In Best First we follow the steps of Breadth First algorithm, but instead of searching blindly all the nodes that come up in the fringe we always choose with the help of a heuristic function the optimal node (e.g. the node with the smaller path cost).
6. **False**
   The two algorithms are similar. The difference is located to the fact that A* uses a heuristic function to find the optimal solution by choosing the lowest cost node from the frontier. The above makes the A* a special case of the uniform cost algorithm, since it uses the aforementioned heuristic to raise its optimality.

Problem 4

We find that the misplaced-tiles heuristic, $h_{1,}$ is exact for the problem where a tile can move from square A to square B. This is a relaxation of the condition that a tile can move from square A to B if only square B is blank. If h_3 is Gaschnig’s heuristic, then it cannot be less than the h_1, the heuristic for the misplaced tiles. Due to the fact that it is also admissible, since it is exact for the relaxation of the original problem, Gaschnig’s heuristic, $h_{3,}$ is more accurate than h_1 (not just as accurate).

Example: Let’s say we have a goal state in which we permute two adjacent tiles. Thus, we have a state where both h_1 (misplaced-tiles heuristic) and h_2 (Manhattan distance) return 2. On the other hand, h_3 returns 3.

Problem 5

1. Hill Climbing Algorithm with h = 1 (always moving to the next adjacent optimal option)
2. Breadth First Search Algorithm (the only difference is that all the nodes of a layer are created simultaneously)
3. Hill Climbing Algorithm, where we always choose the first optimal move.
4. Mutation. If the population is 1, then the two individuals chosen for cross-fertilization will be the same individual. As a result, the offspring will have the same gene code thus, being the same individual as his parent/s. The algorithm ends up making a random search in the search space.