CS 5100: Foundations of Artificial Intelligence (Fall 2023) Chris Amato
Student: Bharat Chawla Due September 29, 2023

Solving Problems by Searching

1 Solving Problems by Searching

1. Which of the following are true and which are false? Explain your answers.

   (a) A* search with an admissible heuristic is always more efficient than breadth-first search (BFS).

   True, A* is optimally efficient for any given heuristic function because no other algorithm is guaranteed to expand fewer nodes than A*. Whereas, BFS just searches the nodes at the current depth level to reach a goal state which guarantees to find a shortest path to the goal irrespective of it's optimality. Hence, any algorithm that does not expand all the nodes with f(n) < C* runs the risk of missing the optimal solution.

   (b) $h(n) = 0$ is an admissible heuristic for the 8-puzzle.

   True, when h(n) = 0 is an admissible heuristic because it meets the requirements of an admissible heuristic function which never overestimates the cost to reach the goal from the current state i.e. true cost can't be negative.

   (c) A* is of no use in robotics because percepts, states, and actions are continuous.

   False, it is not directly applicable to continuous spaces, but can be modified to adapt to it which can be done by the process of discretization, that is, dividing up the possible values into a fixed set of intervals.

   (d) Assume that a rook can move on a chessboard any number of squares in a straight line, vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the problem of moving the rook from square A to square B in the smallest number of moves.

   True. The reasons why Manhattan distance is admissible because it is always consistent. The estimated cost to reach the goal from a state is always less than or equal to true cost. In this instance, true cost is the minimum moves required rook to move from A to B and since it can only travel vertically or horizontally, never jumps over other pieces, it must travel the Manhattan distance. Secondly, it is optimistic because it never overestimates the true cost.

2. Prove each of the following statements, or give a counterexample:

(a) Breadth-first search is a special case of uniform-cost search.

True. This will be the case if all steps costs are equal i.e. if they all are 1 or equal to another constant.
Time complexity for BFS is b^{d+1} and for UCS is $b^{1 + [c^* / e]}$ where C* denotes cost of the optimal solution.
However, when all steps costs are equal, it will just become b^{d+1}.

(b) Uniform-cost search is a special case of A* search.

True. This will be the case when h(s) = 0.

f(s) = h(s) + g(s)
f(s) = 0 + g(s)
f(s) = g(s)

2 Search Problem 1

Consider the two graphs shown in Figures ?? and ??. In what order does BFS, DFS and UCS (assume you use the graph search version) expand the nodes for each graph (assume that nodes are added to the stack/queue in alphabetical order)? The agent starts at node s and must reach node g. (Note: All iterations might not be necessary.)

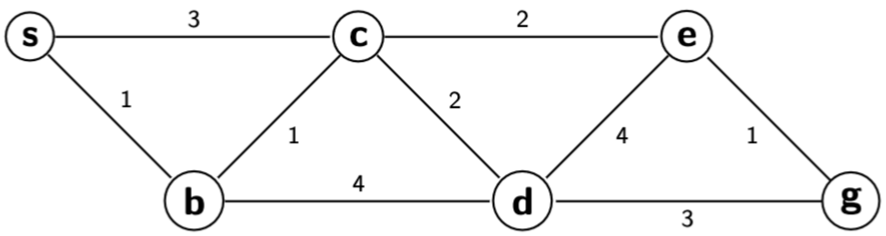


Figure 1: Graph A

Table 1: Answer:

	BFS		DFS		UCS	
Iter.	Queue <-	CurrNode	Stack ->	CurrNode	Queue <-	CurrNode
0.	s		s		s(0)	
1.	b, c	s	b, c	s	b(1), c(3)	s
2.	c, d	b	b, d, e	c	c(2), d(5)	b
3.	d, e	c	b, d, g	e	d(4), e(4)	c
4.	e, g	d	b, d	g	e(4), g(7)	d
5.	g	e			g(5)	e
6.		g				g
7.						
8.						

Path for BFS: s->b->d->g
Path for DFS: s->c->e>g
Path for UCS: s->b->c->e->g

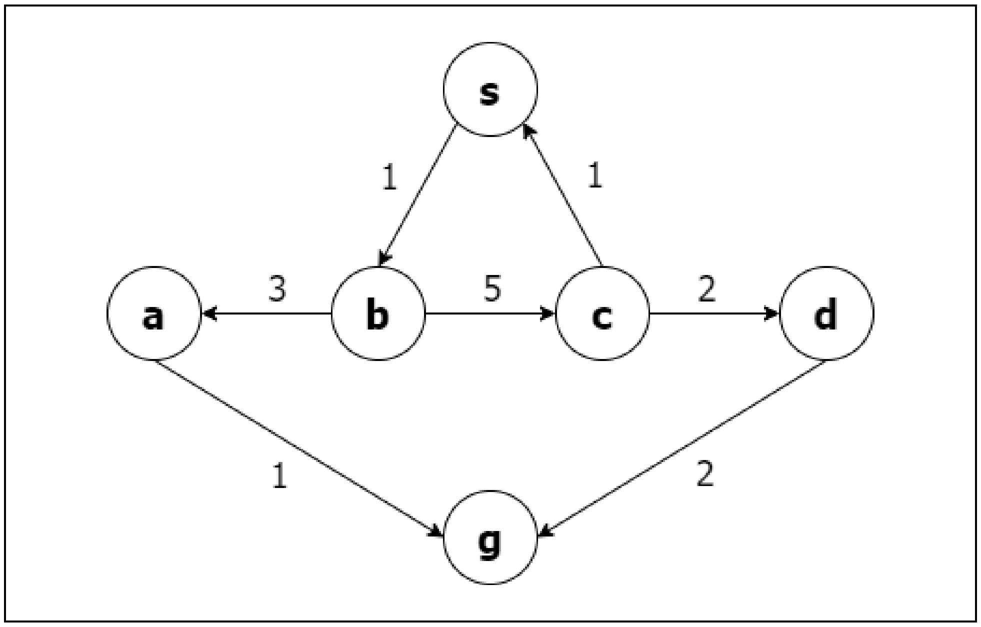


Figure 2: Graph B

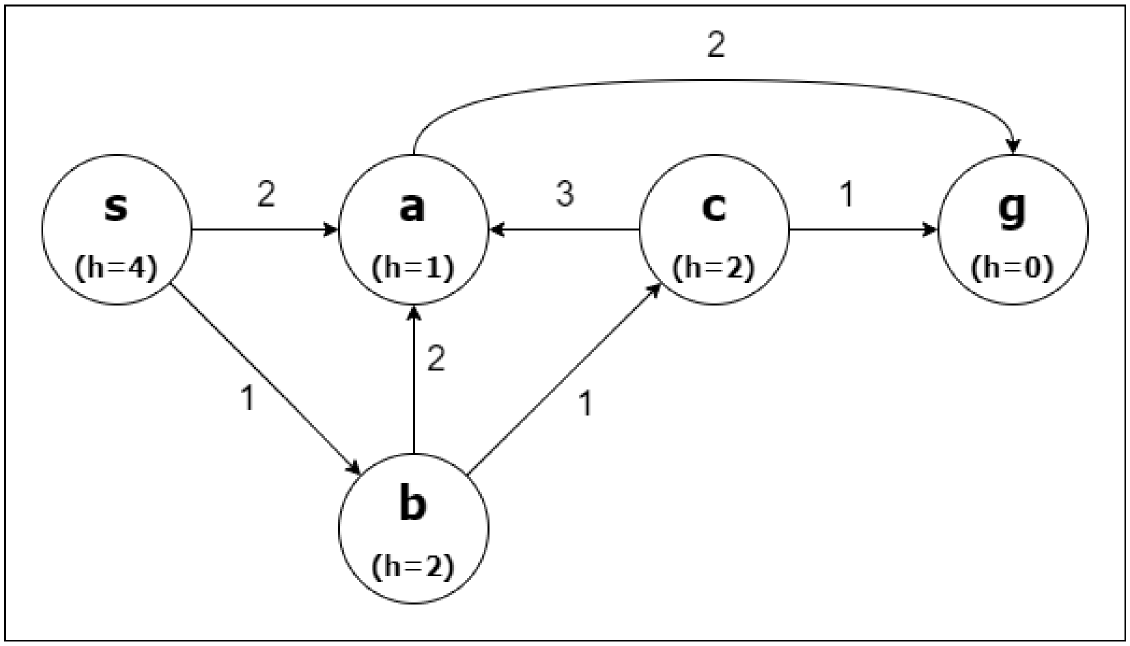
Table 2: Answer:

	BFS		DFS		UCS	
Iter.	Queue <-	CurrNode	Stack ->	CurrNode	Queue <-	CurrNode
0.	s		s		s(0)	
1.	b	s	b	s	b(1)	s
2.	a, c	b	c, a	b	a(4), c(6)	b
3.	c, g	a	d, a	c	g(5), c(6)	a
4.	g, d, s	c	g, a	d	c(6)	g
5.	d, s	g	a	g		
6.						
7.						
8.						

Path for BFS: s->b->a->g
Path for DFS: s->b->a->g
Path for UCS: s->b->a->g

3 Search Problem 2

Consider the graph shown in the below figure. In what order does A* expand the nodes for the graph (assume that nodes are added to the stack/queue in alphabetical order)? The agent starts at node s and must reach node g. (Note: All iterations might not be necessary.)



A*		
Iter.	Queue	CurrNode
1.	s(4)	
2.	a(7), b(7)	s
3.	b(7), g(9)	a
4.	g(9), a(10), c(10)	b
5.	a(10), c(10)	g
6.		
7.		

Path: s->a->g