Fundamentals of Artificial Intelligence

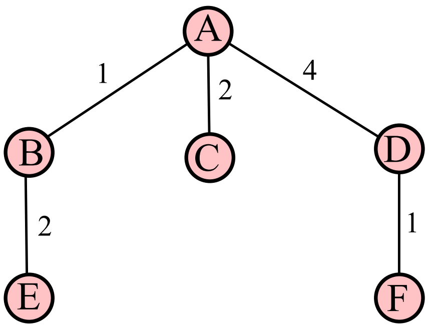
Exercise 2a: Uninformed Search – Solutions Winter Semester 2022/23

Sebastian Mair

1 Presented Problems

Problem 2.1: Graph Search

Consider the following graph. We start from A and our goal is F. Path costs are shown on the arcs. If there is no clear preference for the next node to explore, we explore the node which is first alphabetically.



Perform Breadth-First, Depth-First and Uniform-Cost search using graph search. For each step, write down the node which is currently expanded, the frontier set, and the explored set.

Solution: Note that when writing down the search steps, we omit the initialization step (where the initial state is set as current node, this node becomes the only element of the frontier set, and the explored set is initialized as empty set), but directly start with the search loop.

Solution: Breadth-First: (Recall that Breadth-First uses a FIFO queue for the frontier set.)

node	A	B	C	D
frontier	BCD	CDE	DE	E
explored	A	AB	ABC	ABCD

Solution: Depth-First: (Here we use the non-recursive implementation. Recall that Depth-First uses a LIFO queue for the frontier set—if the depth is equal, we give preference to the children which is first alphabetically.)

node with depth	A(0)	B(1)	E(2)	C(1)	D(1)
frontier with depth	B(1), C(1), D(1)	C(1), D(1), E(2)	C(1), D(1)	D(1)	
explored	A	AB	ABE	ABEC	ABECD

Solution: Uniform-Cost:

node with cost	A(0)	B(1)	C(2)	E(3)	D(4)	F(5)
frontier with cost	B(1), C(2), D(4)	C(2), E(3), D(4)	E(3), D(4)	D(4)	F(5)	
explored	A	AB	ABC	ABCE	ABCED	ABCED

Problem 2.2: Application of Search Algorithms: Transport

I have bought 120 bricks at the hardware store, which I need to transport to my house. I have the following actions a:

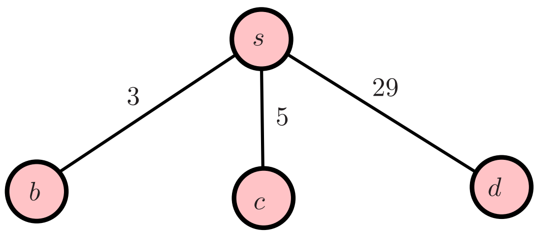
1. take the bus (b) – I can carry up to 30 bricks, costing 3 €
2. take my car (c) – it can carry up to 100 bricks, costing 5 €
3. take a delivery (d) – it delivers all the bricks, costing 29 €

Assume that I explore actions in the order: bus, car, delivery; and we start at the initial node s. For clarity, we name nodes after the actions taken to get to them (e.g. after taking the bus twice, the node is labeled with bb); and the order of actions reaching a state is not important (e.g. the states bc and cb are considered to be the same).

Solution: The goal test for a node n is $30 \times \text{num}(b, n) + 100 \times \text{num}(c, n) + 120 \times \text{num}(d, n) \geq 120$, where $\text{num}(a, n)$ returns the number of actions of type a performed to reach the node n.

Problem 2.2.1: Assuming I want to make as few trips as possible, which search method should I use and what is the resulting solution?

Solution: Breadth-First, since it expands nodes only until the depth of the shallowest goal node. The solution is d with cost of 29.

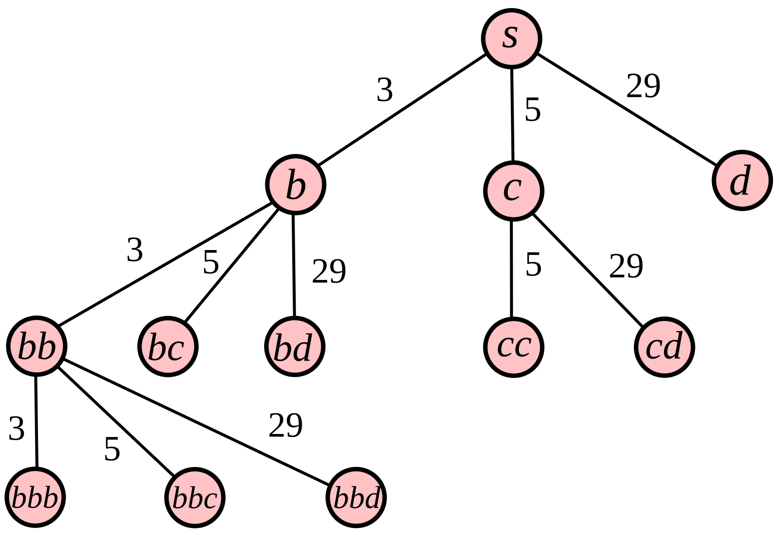


node with cost	$s(0)$
frontier with cost	$b(3)$, $c(5)$

Solution: Note that if we apply the order of actions given in Problem 2.2.4 in the Breadth-First search, the frontier set would be empty (since the algorithm returns the solution before adding the other child nodes b and c to the frontier).

Problem 2.2.2: Assuming I want to minimize the cost, which search method should I use and what is the resulting solution?

Solution: Uniform-Cost, since it is optimal. The solution is bc with cost of 8.



node with cost	s(0)	b(3)	c(5)
frontier with cost	b(3), c(5), d(29)	c(5), bb(6), bc(8), d(29), bd(32)	bb(6), bc(8), cc(10), d(29), bd(32), cd(34)

bb(6)	bc (8)
bc(8), bbb(9), cc(10), bbc(11), d(29), bd(32), cd(34), bbd(35)	bbb(9), cc(10), bbc(11), d(29), bd(32), cd(34), bbd(35)

Problem 2.2.3: Perform depth-first search using the recursive implementation.

Solution: The solution is *bbbb* with cost of 12. The explored nodes are, in order, *s*, *b*, *bb*, *bbb*, *bbbb*.

Problem 2.2.4: Perform depth-first search using the recursive implementation again, but now assume I explore actions in the order: delivery, car, bus.

Solution: The solution is *d* with cost of 29. The only explored nodes are *s*, *d*.

2 Additional Problems

Problem 2.3: Application of Search Algorithm: Train Journey

From Problem 2.6 (of Exercise 2b), perform Breadth-First, Depth-First, and Uniform-Cost search using graph search for both time and price cost. For each step, write down the node which is currently expanded, the frontier set, and the explored set.

Solution: Breadth-First: The search is independent of the cost measure. Note that for each node in the frontier, we track the parent that lead to this node (in brackets); we use the information about the parent to construct the resulting path, by backtracking from the goal node (always considering the expanded nodes). BFS returns the solution A-E-Y-D-B with cost in time of 440 minutes and in price of £184.

node with parent	A	E(A)	G(A)	C(E)
frontier with parent	E(A), G(A)	G(A), C(E), Y(E)	C(E), Y(E)	Y(E), P(C)
explored	A	AE	AEG	AEGC

Y(E)	P(C)	D(Y)
P(C), D(Y), M(Y)	D(Y), M(Y), W(P)	M(Y), W(P)
AEGCY	AEGCYP	AEGCYPD

Solution: Depth-First: The search is independent of the cost measure. After exploring 6 nodes, it returns the solution A-E-C-P-M-S-B with cost in time of 420 minutes and in price of £125.
We do not track the parent for each node here, as our graph does not have dead ends: therefore, DFS will never backtrack, so that the resulting path simply consists of all expanded nodes and the goal node.

node with depth	A(0)	E(1)	C(2)	P(3)	M(4)	S(5)
frontier with depth	E(1), G(1)	G(1), C(2), Y(2)	G(1), Y(2), P(3)	G(1), Y(2), M(4), W(4)	G(1), Y(2), W(4), S(5)	G(1), Y(2), W(4)
explored	A	AE	AEC	AECP	AECPM	AECPMS

Solution: Uniform-Cost (for time cost): The solution is A-E-C-P-W-B with cost in time of 400 minutes and in price of £131 (which is optimal in time). Like for BFS, we track the parent for each node to construct the solution path at the end.

node with cost and parent	A(0)	E(150, A)	G(200, E)	C(220, E)	P(290, C)	Y(300, E)	W(315, P)	M(330, P)	S(370, M)	B(400, W)
frontier with cost and parent	E(150, A), G(350, A)	G(200, E), C(220, E), Y(300, E)	C(220, E), Y(300, E)	P(290, C), Y(300, E)	Y(300, E), W(315, P), M(330, P)	W(315, P), M(330, P), D(400, Y)	M(330, P), B(400, W), D(400, Y)	S(370, M), B(400, W), D(400, Y)	B(400, W), D(400, Y)	D(400, Y)
explored	A	AE	AEG	AEGC	AEGCP	AEGCPY	AEGCPYW	AEGCPYWM	AEGCPYWMS	AEGCPYWMS

Solution: Uniform-Cost (for price cost): The solution is A-E-G-C-P-M-S-B with cost in time of 470 minutes and in price of £109 (which is optimal in price).

node with cost and parent	A(0)	E(29, A)	G(37, E)
frontier with cost and parent	E(29, A), G(38, A)	G(37, E), C(79, E), Y(112, E)	C(63, G), Y(112, E)
explored	A	AE	AEG

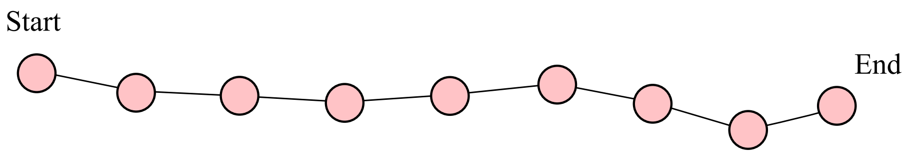
C(63, G)	P(73, C)	M(78, P)
P(73, C), Y(112, E)	M(78, P), W(80, P), Y(112, E)	W(80, P), S(89, M), Y(111, M)
AEGC	AEGCP	AEGCPM

W(80, P)	S(89, M)	D(98, S)	B(109, S)
S(89, M), Y(111, M), B(115, W)	D(98, S), B(109, S), Y(111, M)	B(109, S), Y(111, M)	Y(111, M)
AEGCPMW	AEGCPMWS	AEGCPMWSD	AEGCPMWSD

Problem 2.4: General Questions on Uninformed Search

Problem 2.4.1: (from *Russell & Norvig 3ed.* q. 3.18) Describe a state transition graph in which iterative deepening search performs much worse (in time) than depth-first search (for example, $\mathcal{O}(n^2)$ vs. $\mathcal{O}(n)$).

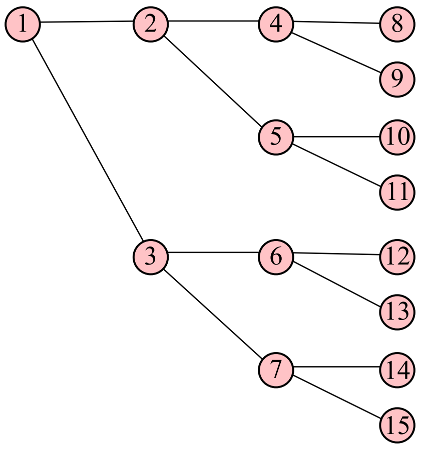
Solution: For example in the following graph (using the recursive implementation), depth-first will expand d nodes, whereas iterative deepening will expand $1 + \ldots + d = \frac{d(d+1)}{2}$ nodes, which is $\emptyset(n^2)$.



Solution: Another more general example is a tree where the only goal happens to be the left-most node of of the tree at a depth d and the branching factor is $b \geq d$. Then, the time complexity of depth-first and iterative deepening is $\mathcal{O}(d)$ and $\mathcal{O}(b^d)$, respectively. Further examples exist.

Problem 2.4.2: (from *Russell & Norvig 3ed.* q. 3.15) Consider a state transition graph where the start state is the number 1 and the successor function for state n returns two states, numbers $2n$ and $2n + 1$.

a. Draw the portion of the state transition graph for states 1 to 15.



b. Suppose the goal state is 11. List the order in which nodes will be generated (not explored) for Breadth-First search, Depth-Limited search with limit 2, and Iterative Deepening search. Would bidirectional search be appropriate for this problem?

Solution: Breadth First Search: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Depth-Limited Search with limit 2: 1, 2, 4, 5, 3, 6, 7
Note that we use the recursive-DLS algorithm from slide 64 of lecture 3. So the run of the code looks like this:

```
DLS(node=1,limit=2){
  DLS(node=2,limit=1){
    DLS(node=4,limit=0) = cutoff!
    DLS(node=5,limit=0) = cutoff!
    } = cutoff!
  DLS(node=3,limit=1) = cutoff!
    DLS(node=6,limit=0) = cutoff!
    DLS(node=7,limit=0) = cutoff!
    } = cutoff!
  } = cutoff!
```

Solution: Iterative Deepening:
(First Iteration) 1
(Second Iteration) 1, 2, 3
(Third Iteration) 1, 2, 4, 5, 3, 6, 7
(Fourth Iteration) 1, 2, 4, 8, 9, 5, 10, 11

Solution: Yes, bidirectional search would work for this problem.

c. What is the branching factor b in each direction of the bidirectional search?

Solution: Forwards: 2, Backwards: 1.

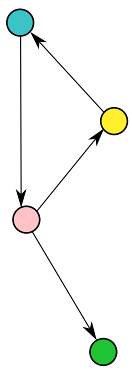
d. Does the answer to **c.** suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?

Solution: Yes, we can work backwards with the algorithm $n_{k-1} = \lfloor \frac{n_k}{2} \rfloor$ until we reach 1.

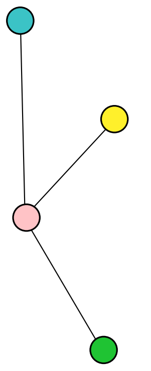
Problem 2.4.3: (from *Russell & Norvig, 2ed.* q. 3.6) Does a finite state transition graph always lead to a finite search tree? How about a finite state transition graph that is a tree?

Solution: No, a tree search algorithm (which does not kept track of nodes already visited in an explored set) can result in an infinite search tree. In the left part of the figure below, the finite state transition graph containing a cycle leads to an infinite search tree. If the state transition graph is a tree and actions are reversible, the search tree is also infinite (see right part of the figure).
If using graph search (with an explored set), a finite state transition graph will lead to a finite search tree. If, in addition, the state transition graph is a tree, you only need to keep track of the parent node to avoid cycles.

Finite graph (with a cycle):



Finite graph which is a tree:



The resulting search trees are infinite: