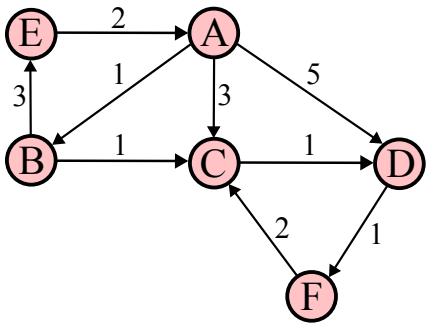
FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE
Exercise 2a: Uninformed Search – Solutions Winter Semester 2023/24
Sebastian Mair

1 Presented Problems

Problem 2.1: Search Algorithms Basics

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, which child node should be visited next during expansion, we choose the node that is first alphabetically.



Perform Breadth-First, Depth-First and Uniform-Cost search. For each step, write down the node which is currently expanded, the frontier, and the reached set or lookup table (if applicable).
Hint: You can generally write $X(C, P)$ for a node n, where $X =$ `n.STATE`, $C =$ `n.PATH-COST`, and $P =$ `n.PARENT.STATE`. You can omit C for Breadth-First and Depth-First search (as it is not needed during search).

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

node	-	A(-)	B(A)	C(A)	D(A)
frontier	A(-)	B(A), C(A), D(A)	C(A), D(A), E(B)	D(A), E(B)	E(B)
reached	A	A, B, C, D	A, B, C, D, E	A, B, C, D, E	A, B, C, D, E

In the last step, we generate the child node $F(D)$ when expanding $D(A)$ and stop searching, as F is a goal state. Note that we perform an early goal check in BFS, i.e., we already check whether a state is a goal as soon as the state is visited for the first time.
To reconstruct the solution path from the goal node F(D), we follow the parent pointers upwards and reverse the order of the resulting chain, resulting in A - D - F. We can take the parent-pointer of a node of the time it was expanded (we will never expand more than one node for a single state in BFS).

Solution: Depth-First: (Recall that Depth-First uses a LIFO queue for the frontier.)

node	-	A(-)	D(A)	F(D)
frontier	A(-)	B(A), C(A), D(A)	B(A), C(A), F(D)	B(A), C(A)

Note that we do *NOT* perform an early goal check in DFS, i.e., we only check if a node is a goal as soon as this node is popped from the frontier.
To reconstruct the solution path from the goal node F(D), we again follow the parent pointers upwards and reverse

the order of the resulting chain, resulting in A - D - F. We can take the parent-pointer of a node of the time it was expanded (in DFS, we have that for any state that lies on the solution path, only one node wrapping that state is expanded).

Solution: Uniform-Cost:

Solution: Note: for Uniform-Cost Search, *reached* is actually a lookup table mapping states to nodes. However, we only write it as a set of nodes here, as the corresponding state mapping to a node is directly apparent from the node itself: e.g., the node "A(0, -)" corresponds to the entry A→A(0, -) in the lookup table. When expanding the next node, we also write down the generated child nodes.

node	-	A(0, -)	B(1, A)	C(2, B)	D(3, C)	E(4, B)	F(4, D)
children	-	B(1, A), C(3, A), D(5, A)	C(2, B), E(4, B)	D(3, C)	F(4, D)	A(6, E)	-
frontier	A(0, -)	B(1, A), C(3, A), D(5, A)	C(2, B), E(4, B), D(5, A)	D(3, C), E(4, B)	E(4, B), F(4, D)	F(4, D)	
reached	A(0, -)	A(0, -), B(1, A), C(3, A), D(5, A)	A(0, -), B(1, A), C(2, B), D(5, A), E(4, B)	A(0, -), B(1, A), C(2, B), D(3, C), E(4, B)	A(0, -), B(1, A), C(2, B), D(3, C), E(4, B), F(4, D)	A(0, -), B(1, A), C(2, B), D(3, C), E(4, B), F(4, D)	A(0, -), B(1, A), C(2, B), D(3, C), E(4, B), F(4, D)

Note that we do *NOT* perform an early goal check in UCS, i.e., we only check if a node is a goal as soon as this node is popped from the frontier.
To reconstruct the solution path from the goal node F(D), we again follow the parent pointers upwards and reverse the order of the resulting chain, resulting in A - B - C - D - F. Here, we can take the parent-pointer of each node of the *reached* lookup table of the last search step, or again take the parent-pointer of each node of the time it was expanded. Like in BFS, we will never expand more than one node for a single state in UCS.

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:

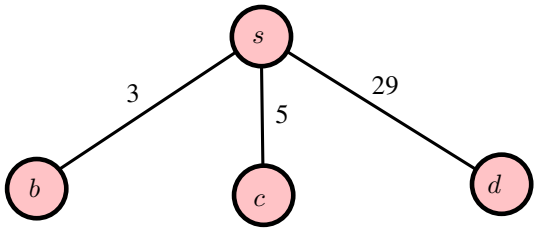
1. take the bus (b) – I can carry up to 30 bricks, costing 3 €
2. take my car (c) – it can carry up to 100 bricks, costing 5 €
3. take a delivery (d) – it delivers all the bricks, costing 29 €

Assume that I apply actions in the order: bus, car, delivery; and we start at the initial node s. For clarity, we name states after the actions taken to get to them (e.g. after taking the bus twice, the state 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.
Hint: Throughout the solution of problem 2.2, we omit the parent-pointers in the notation of the nodes for the sake of simplicity.

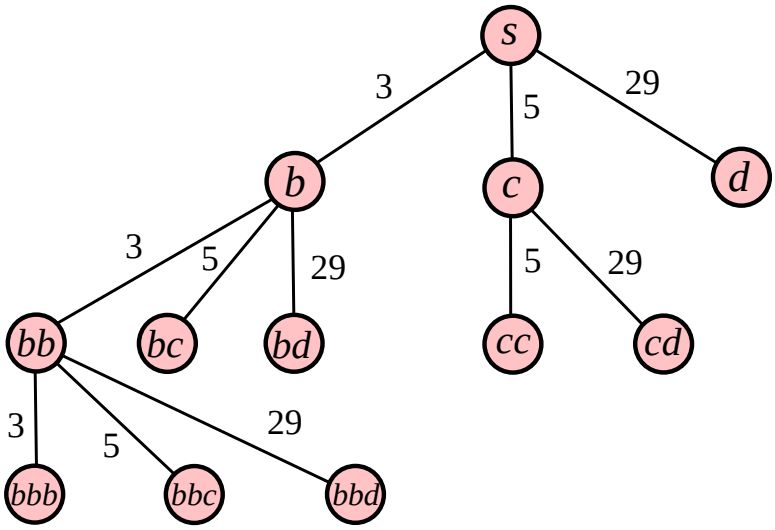


node	-	s
frontier	s	b, c
reached	s	s, b, c

Solution: Note that if we apply the order of actions given in Problem 2.2.4 in the Breadth-First search, the frontier 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	-	s(0)	b(3)	c(5)
frontier	s(0)	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)
reached	s(0)	s(0), b(3), c(5), d(29)	s(0), b(3), c(5), d(29), bb(6), bc(8), bd(32)	s(0), b(3), c(5), d(29), bb(6), bc(8), bd(32), cc(10), 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)
s(0), b(3), c(5), d(29), bb(6), bc(8), bd(32), cc(10), cd(34), bbb(9), bbc(11), bbd(35)	s(0), b(3), c(5), d(29), bb(6), bc(8), bd(32), cc(10), cd(34), bbb(9), bbc(11), bbd(35)

Problem 2.2.3: Perform depth-first search.

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

node	-	s	d
frontier	s	b, c, d	b, c

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

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

node	-	s	b	bb	bbb	bbbb
frontier	s	d, c, b	d, c, bd, bc, bb	d, c, bd, bc, bbd, bbc, bbb	d, c, bd, bc, bbd, bbc, bbbd, bbbc, bbbb	d, c, bd, bc, bbd, bbc, bbbd, bbbc

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 for both time and price cost. For each step, write down the node which is currently expanded, the frontier, and the reached set or lookup table (if applicable).
Hint: You can again write $X(C, P)$ for a node n, where $X =$ `n.STATE`, $C =$ `n.PATH-COST`, and $P =$ `n.PARENT.STATE`. You can omit C for Breadth-First and Depth-First search (as it is not needed during search).

Solution: Breadth-First: The search is independent of the cost measure. Note that we use the parent-pointers to construct the resulting solution 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	-	A(-)	E(A)	G(A)	C(E)	Y(E)	P(C)	D(Y)
frontier	A(-)	E(A), G(A)	G(A), C(E), Y(E)	C(E), Y(E)	Y(E), P(C)	P(C), D(Y), M(Y)	D(Y), M(Y), W(P)	M(Y), W(P)
reached	A(-)	A(-), E(A), G(A)	A(-), E(A), G(A), C(E), Y(E)	A(-), E(A), G(A), C(E), Y(E)	A(-), E(A), G(A), C(E), Y(E), P(C)	A(-), E(A), G(A), C(E), Y(E), P(C), D(Y), M(Y)	A(-), E(A), G(A), C(E), Y(E), P(C), D(Y), M(Y), W(P)	A(-), E(A), G(A), C(E), Y(E), P(C), D(Y), M(Y), W(P)

Solution: Depth-First: The search is independent of the cost measure. After expanding 5 nodes, it returns the solution A-G-C-P-W-B with cost in time of 600 minutes and in price of £116.
As our graph does not have dead ends, DFS will never backtrack, so that the resulting path simply consists of all expanded nodes.

node	-	A(-)	G(A)	C(G)	P(C)	W(P)	B(W)
frontier	A(-)	E(A), G(A)	E(A), C(G)	E(A), P(C)	E(A), M(P), W(P)	E(A), M(P), B(W)	E(A), M(P)

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).

node	-	A(0, -)	E(150, A)	G(200, E)	C(220, E)	P(290, C)
children	-	E(150, A), G(350, A)	A(300, E), C(220, E), G(200, E), Y(300, E)	A(550, G), C(270, G), E(250, E)	E(290, C), G(290, C), P(290, C)	C(360, P), M(330, P), W(315, P)
frontier	A(0, -)	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)
reached	A(0, -)	A(0, -), E(150, A), G(350, A)	A(0, -), E(150, A), G(200, A), C(220, E), Y(300, E)	A(0, -), E(150, A), G(200, A), C(220, E), Y(300, E)	A(0, -), E(150, A), G(200, A), C(220, E), Y(300, E), P(290, C)	A(0, -), E(150, A), G(200, A), C(220, E), Y(300, E), P(290, C), W(315, P), M(330, P)

Y(300, E)	W(315, P)	M(330, P)	S(370, M)	B(400, W)
E(450, Y), M(380, Y), D(400, Y)	B(400, W), P(340, W)	P(370, M), S(370, M), Y(410, M)	B(420), D(430), M(410)	-
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)
A(0, -), E(150, A), G(350, A), C(220, E), Y(300, E), P(290, C), W(315, P), M(330, P), D(400, Y)	A(0, -), E(150, A), G(350, A), C(220, E), Y(300, E), P(290, C), W(315, P), M(330, P), D(400, Y), B(400, W)	A(0, -), E(150, A), G(350, A), C(220, E), Y(300, E), P(290, C), W(315, P), M(330, P), D(400, Y), B(400, W), S(370, M)	A(0, -), E(150, A), G(350, A), C(220, E), Y(300, E), P(290, C), W(315, P), M(330, P), D(400, Y), B(400, W), S(370, M)	A(0, -), E(150, A), G(350, A), C(220, E), Y(300, E), P(290, C), W(315, P), M(330, P), D(400, Y), B(400, W), S(370, M)

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).

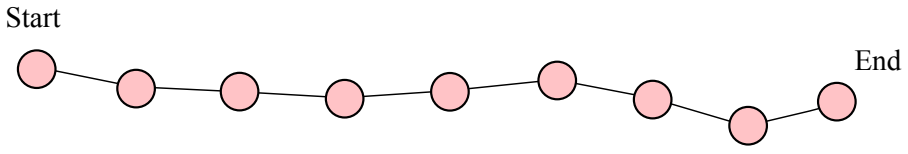
node	-	A(0, -)	E(29, A)	G(37, E)	C(63, G)	P(73, C)
children -	E(29, A), G(38, A)	A(58, E), C(79, E), G(37, E), Y(112, E)	A(75, G), C(63, G), E(45, G)	E(113, C), G(89, C), P(73, C)	C(83, P), M(78, P), W(80, P)	
frontier	A(0, -)	E(29, A), G(38, A)	G(37, E), C(79, E), Y(112, E)	C(63, G), Y(112, E)	P(73, C), Y(112, E)	M(78, P), W(80, P), Y(112, E)
reached	A(0, -)	A(0, -), E(29, A), G(38, A)	A(0, -), E(29, A), G(37, E), C(79, E), Y(112, E)	A(0, -), E(29, A), G(37, E), C(63, G), Y(112, E)	A(0, -), E(29, A), G(37, E), C(63, G), Y(112, E), P(73, C)	A(0, -), E(29, A), G(37, E), C(63, G), Y(112, E), P(73, C), M(78, P), W(80, P)

M(78, P)	W(80, P)	S(89, M)	D(98, S)	B(109, S)
P(83, M), S(89, M), Y(111, M)	B(115, W), P(87, W)	B(109, S), D(98, S), M(100, S)	B(116, D), S(107, D), Y(152, D)	-
W(80, P), S(89, M), Y(111, M)	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)
A(0, -), E(29, A), G(37, E), C(63, G), Y(111, M), P(73, C), M(78, P), W(80, P), S(89, M)	A(0, -), E(29, A), G(37, E), C(63, G), Y(111, M), P(73, C), M(78, P), W(80, P), S(89, M), B(115, W)	A(0, -), E(29, A), G(37, E), C(63, G), Y(111, M), P(73, C), M(78, P), W(80, P), S(89, M), B(109, S), D(98, S)	A(0, -), E(29, A), G(37, E), C(63, G), Y(111, M), P(73, C), M(78, P), W(80, P), S(89, M), B(109, S), D(98, S)	A(0, -), E(29, A), G(37, E), C(63, G), Y(111, M), P(73, C), M(78, P), W(80, P), S(89, M), B(109, S), D(98, S)

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. $\mathcal{O}(n)$).

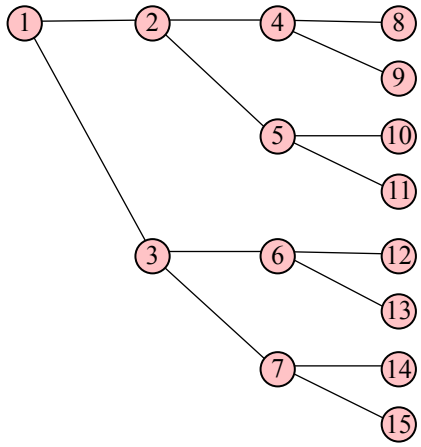
Solution: For example in the following graph, depth-first will expand d nodes, whereas iterative deepening will expand $1 + \ldots + d = \frac{d(d+1)}{2}$ nodes, which is $\emptyset(n^2)$.



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 expanded) 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, 3, 6, 7, 4, 5

Solution: Iterative Deepening:
(First Iteration) 1
(Second Iteration) 1, 2, 3
(Third Iteration) 1, 2, 3, 6, 7, 4, 5
(Fourth Iteration) 1, 2, 3, 6, 7, 14, 15, 12, 13, 4, 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.

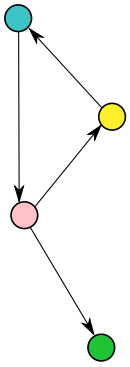
d. Does the answer to **c.** suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?

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

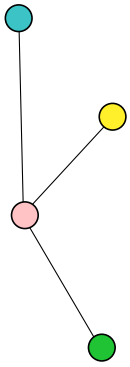
Problem 2.4.3: (from *Russell & Norvig, 2ed.* q. 3.6) Does a finite state transition graph always lead to a finite search tree? How about a finite state transition graph that is a tree?

Solution: No, a tree-like 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 may lead to an infinite search tree. If the state transition graph is a tree and actions are reversible, the search tree can still be infinite (see right part of the figure).
If using graph search (with keeping track of the reached states), 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: