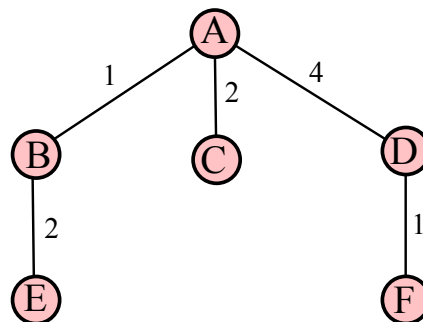


1 Presented Problems

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.


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.

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

F is not included (with arrow pointing to the empty cell in the frontier row)

Depth-First: (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

Goal 'F' is not written (with arrow pointing to the empty box in the frontier row)

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

equal depth, choose alphabetically (with arrow pointing to B(1), C(1), D(1) in the Depth-First table)
unequal depth, LIFO (with arrow pointing to C(1), D(1), E(2) in the Depth-First table)
F as a target is considered (with arrow pointing to F(5) in the Uniform-Cost table)

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 :

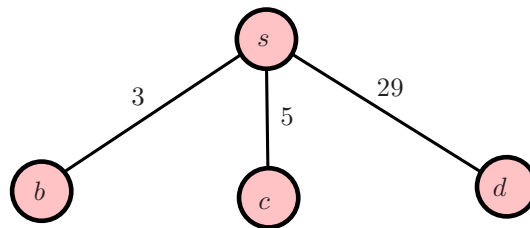
1. take the bus (b) – I can carry up to 30 bricks, costing 3 €
2. take my car (c) – it can carry up to 100 bricks, costing 5 €
3. take a delivery (d) – it delivers all the bricks, costing 29 €

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).

Since actions are not reversible in this problem (unlike e.g. route-finding problems), no loops can occur in our search. Thus, we define this problem as a tree and use tree search. 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?

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

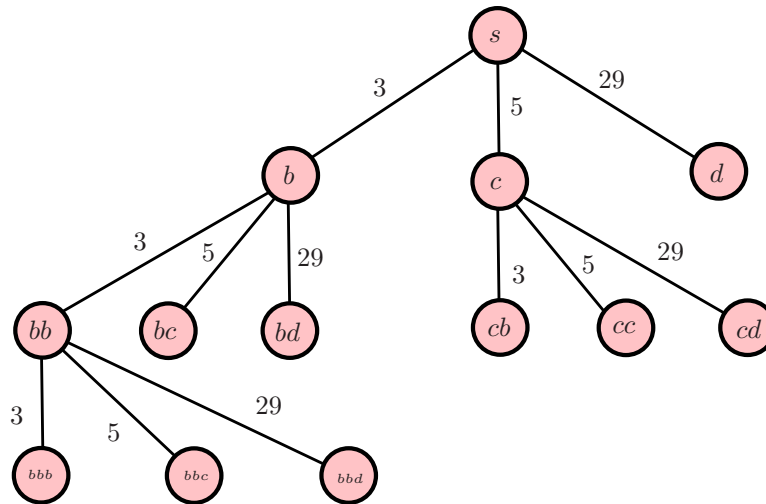


node with cost	$s(0)$
frontier with cost	$b(3), c(5)$

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?

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.

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.

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.

Breadth-First: The search is independent of the cost measure. After exploring 6 nodes, it returns the solution A-E-Y-D-B with cost in time of 440 minutes and in price of £184. Shallowest node: one with least depth "first"

node with depth	A(0)	E(1)	G(1)	C(2)	Y(2)
frontier with depth	E(1), G(1)	G(1), C(2), Y(2)	C(2), Y(2)	Y(2), P(3)	D(3), M(3), P(3)
explored	A	AE	AEG	AEGC	AEGCY

D(3)
M(3), P(3)
AEGCYD

S(4) & B(4)
were not
included!

Ⓜ Goal node is not added to Frontier
in breadth-first

Note that in step 5 when exploring node Y, we sort all nodes of equal depth alphabetically. If you strictly follow the FIFO rule (and only sort nodes alphabetically when adding them to the frontier), your frontier is PDM and you will expand node P next, while we will expand D. Since the problem statement asks you to explore the node which is first alphabetically if there is no clear preference for the next node to explore, you should expand node D (since P and D have the same depth).

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. *"deepest first"*
first alphabetically

node with depth	A(0)	E(1)	C(2)	P(3)
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)
explored	A	AE	AEC	AECP

M(4)	S(5)
G(1), Y(2), W(4), S(5)	G(1), Y(2), W(4)
AECPM	AECPMS

→ Goal 'B' is not written

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 with cost	A(0)	E(150)	G(200)	C(220)
frontier with cost	E(150), G(350)	G(200), C(220), Y(300)	C(220), Y(300)	P(290), Y(300)
explored	A	AE	AE G	AE G C

P(290)	Y(300)	W(315)
Y(300), W(315), M(330)	W(315), M(330), D(400)	M(330), B(400), D(400)
AEGCP	AEGCPY	AEGCPYW

M(330)	S(370)	B(400)
S(370), B(400), D(400)	B(400), D(400)	D(400)
AEGCPYWM	AEGCPYWMS	AEGCPYWMS

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	A(0)	E(29)	G(37)	C(63)
frontier with cost	E(29), G(38)	G(37), C(79), Y(112)	C(63), Y(112)	P(73), Y(112)
explored	A	AE	AE G	AE G C

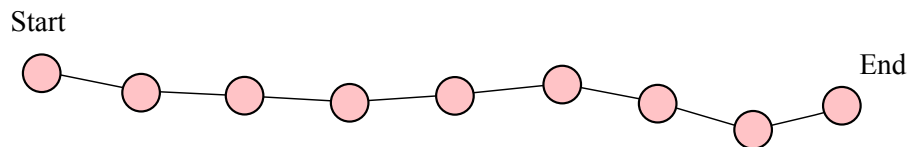
P(73)	M(78)	W(80)
M(78), W(80), Y(112)	W(80), S(89), Y(111)	S(89), Y(111), B(115)
AEGCP	AEGCPM	AEGCPMW

S(89)	D(98)	B(109)
D(98), B(109), Y(111)	B(109), Y(111)	Y(111)
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. $\mathcal{O}(n)$).

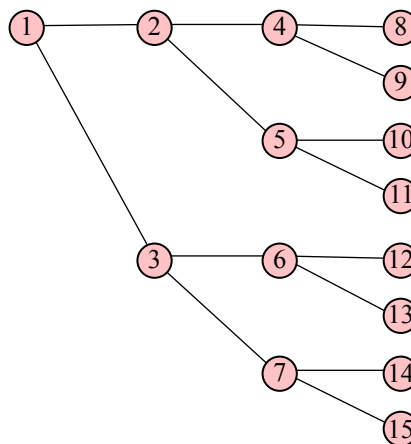
For example in the following graph (using the recursive implementation), depth-first will expand d nodes, whereas iterative deepening will expand $1 + \dots + d = \frac{d(d+1)}{2}$ nodes, which is $\mathcal{O}(n^2)$.



Another more general example is a tree where the only goal happens to be the left-most node 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?

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!

```

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

Yes, bidirectional search would work for this problem.

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

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?

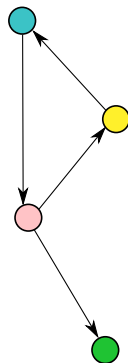
Yes, we can work backwards with the algorithm $n_{k-1} = \lfloor \frac{n_k}{2} \rfloor$ until we reach 1.

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?

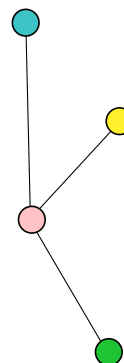
* No, a tree search algorithm (which does not keep track of nodes already visited in an explored set) can result in an infinite search tree. In the left part of the figure below, the finite state transition graph containing a cycle leads to an infinite search tree. If the state transition graph is a 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.

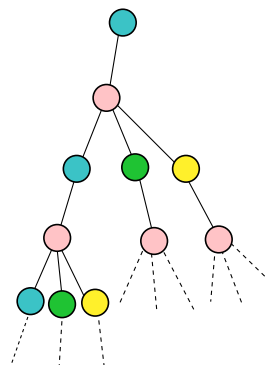
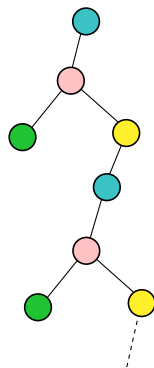
Finite graph (with a cycle):



Finite graph which is a tree:



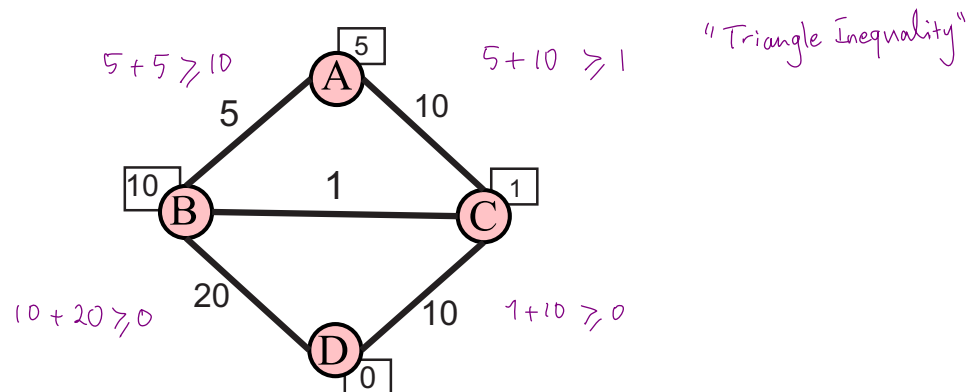
The resulting search trees are infinite:



1 Presented Problems

Problem 2.5: Heuristics for Informed Search

Consider the following graph. We start from node A and our goal node is D. The path costs are shown on the arcs and the heuristic values are shown at each node.



Problem 2.5.1 What are the requirements on a heuristic for A* tree search and graph search to be optimal?

- A* tree search requires the heuristic to be admissible.
- A* graph search requires the heuristic to be consistent.

→ For its definitions, we use $h(n)$ as the heuristic value at node n and $g(n, n')$ as the path cost from node n to node n' .

- An admissible heuristic always under-approximates the actual cost:

$$h(n) \leq g(n, \text{goal}) \quad (1)$$

- A consistent heuristic fulfills this triangle inequality for all nodes:

$$h(n) \leq g(n, n') + h(n') \quad (2)$$

Problem 2.5.2 Is the given heuristic admissible? Is it consistent? If not, why not?

Admissible? : Yes

Consistent? : No, since for nodes B and C the triangle inequality does not hold:

$$h(B) \not\leq c(B, C) + h(C) \quad (3)$$

Remember : in all informed search, it's normal to see a node more than without being updated

Problem 2.5.3 Perform A* graph search and tree search to verify your previous answers

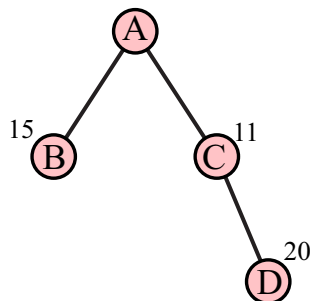
Recall that for A*, $f(n) = g(n) + h(n)$.

n : goal node

- A* graph search:

(NB) $g(n)$ is total of all path cost

(NB) While traversing a tree, no need for explored set



node with $f(n)$	A(5)	C(11)	B(15)	D(20)
frontier with $f(n)$	C(11) B(15)	B(15) D(20)	D(20)	
explored	A	AC	ACB	ACB

→ $g(A,D) + h(D)$

$$= c(A,C) + c(C,D) + h(D) = 10 + 10 + 0$$

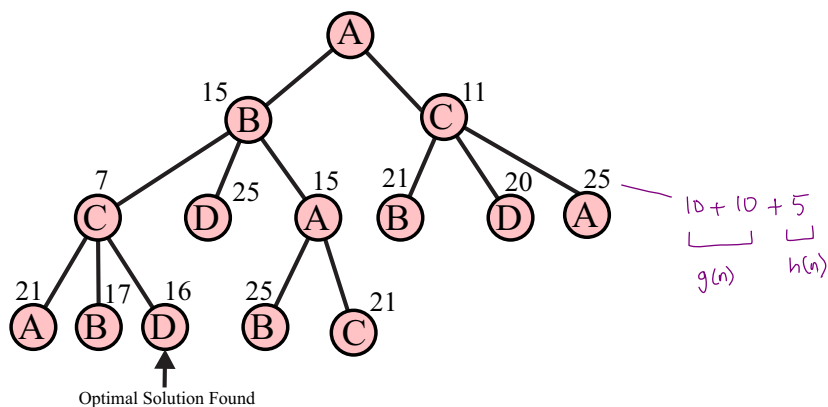
A* graph search did not find the optimal solution. Note that when exploring C in step 2, we do not add its child B to the frontier, since its $f(n) = 21$ is higher than the value of f of the B already on the frontier.

→ $g(A,B) + h(B)$

$$= c(A,C) + c(C,B) + h(B) = 10 + 1 + 10$$

- A* tree search:

$$\begin{aligned} A(21) &= c(A,B) + c(B,C) \\ &+ c(C,A) + h(A) \\ &= 5 + 1 + 10 + 5 \\ &= 21 \end{aligned}$$



node with $f(n)$	A(5)	C(11)	B(15)	C(7)	A(15)	D(16)
frontier with $f(n)$	C(11) B(15)	B(15) D(20) B(21) A(25)	C(7) A(15) D(20) B(21) A(25) D(25)	A(15) D(16) B(17) D(20) A(21) B(21) A(25) D(25)	D(16) B(17) D(20) A(21) B(21) C(21) A(25) B(25) D(25)	B(17) D(20) A(21) B(21) C(21) A(25) B(25) D(25)

A* tree search found the optimal solution.

Problem 2.6: Application of Search Algorithms: Train Journey

We want to travel from Aberdeen to Birmingham. The (semi-fictional) map of the British rail system is available in Fig. 1. Routes have the same cost in both directions for both time and price. If there is no clear preference for the next node to explore, we explore first the node which is first alphabetically.

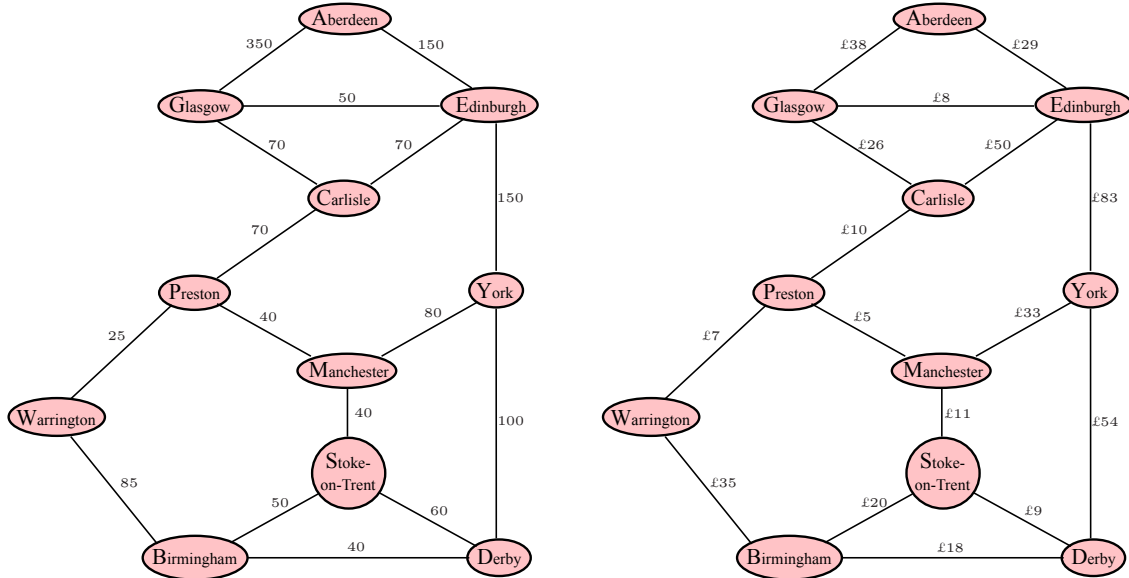


Figure 1: Rail map: left, cost in time (minutes); right, cost in price (£ sterling)

Problem 2.6.1 In the table below, we are given a heuristic for cost in time, which is based on the straight-line distances to Birmingham and the maximum speed of the train being no more than 120km/h. Perform Greedy Best-First and A* Search for time cost using tree search. For each step, write down the node which is currently expanded and the nodes in the frontier set, each with its value of the evaluation function $f(n)$.

node n	A	G	E	C	Y	P	M	W	S	D	B
heuristic function $h(n)$	259	203	197	138	86	76	56	55	31	28	0

Greedy Best-First tree search (recall that $f(n) = h(n)$): The solution is A-E-Y-D-B with cost in time of 440 minutes and in price of £184.

node with $f(n)$	A(259)	E(197)	Y(86)	D(28)	B(0)
frontier with $f(n)$	E(197), G(203)	Y(86), C(138), G(203), A(259)	D(28), M(56), C(138), E(197), G(203), G(203), A(259)	B(0), S(31), M(56), Y(86), C(138), E(197), G(203), G(203), A(259)	S(31), M(56), Y(86), C(138), E(197), G(203), G(203), A(259)

NB It's a tree search

Note that node G(203) is added to the frontier in step 2 again, since in tree search (according to slide 17 of lecture 3) we add nodes to the frontier without checking whether they are already in the frontier.

A* tree search (recall that $f(n) = g(n) + h(n)$): A* finds the optimal solution, which is A-E-C-P-W-B with cost in time of 400 minutes and in price of £131.

← previously there

- newly added

node with $f(n)$	A(259 = 0 + 259)	E(347 = 150 + 197)	C(358 = 220 + 138)
frontier with $f(n)$	E(347 = 150 + 197) G(553 = 350 + 203)	C(358 = 220 + 138) Y(386 = 300 + 86) G(403 = 200 + 203) G(553 = 350 + 203) A(559 = 300 + 259)	P(366 = 290 + 76) Y(386 = 300 + 86) G(403 = 200 + 203) E(487 = 290 + 197) G(493 = 290 + 203) G(553 = 350 + 203) A(559 = 300 + 259)

$h(n)$
 $150 + 70 + 70 = 290$
 "path cost"

P(366 = 290 + 76)	W(370 = 315 + 55)	M(386 = 330 + 56)
W(370 = 315 + 55) M(386 = 330 + 56) Y(386 = 300 + 86) G(403 = 200 + 203) E(487 = 290 + 197) G(493 = 290 + 203) C(498 = 360 + 138) G(553 = 350 + 203) A(559 = 300 + 259)	M(386 = 330 + 56) Y(386 = 300 + 86) B(400 = 400 + 0) G(403 = 200 + 203) P(416 = 340 + 76) E(487 = 290 + 197) G(493 = 290 + 203) C(498 = 360 + 138) G(553 = 350 + 203) A(559 = 300 + 259)	Y(386 = 300 + 86) B(400 = 400 + 0) S(401 = 370 + 31) G(403 = 200 + 203) P(416 = 340 + 76) P(446 = 370 + 76) E(487 = 290 + 197) G(493 = 290 + 203) Y(496 = 410 + 86) C(498 = 360 + 138) G(553 = 350 + 203) A(559 = 300 + 259)

Y(386 = 300 + 86)	B(400 = 400 + 0)
B(400 = 400 + 0) S(401 = 370 + 31) G(403 = 200 + 203) P(416 = 340 + 76) D(428 = 400 + 28) M(436 = 380 + 56) P(446 = 370 + 76) E(487 = 290 + 197) G(493 = 290 + 203) Y(496 = 410 + 86) C(498 = 360 + 138) G(553 = 350 + 203) A(559 = 300 + 259) E(647 = 450 + 197)	S(401 = 370 + 31) G(403 = 200 + 203) P(416 = 340 + 76) D(428 = 400 + 28) M(436 = 380 + 56) P(446 = 370 + 76) E(487 = 290 + 197) G(493 = 290 + 203) Y(496 = 410 + 86) C(498 = 360 + 138) G(553 = 350 + 203) A(559 = 300 + 259) E(647 = 450 + 197)

Problem 2.6.2 For the problem with cost in price (see Fig. 1 right), would a heuristic based on distance be valid?



Since the price cost is not necessarily correlated with distance, such a heuristic might not be admissible. For example, Carlisle is further from Birmingham than York, but costs less to travel from, whereas Aberdeen is further than York and costs more to travel from. An effective heuristic could be obtained by analyzing rail companies' pricing strategies.

Problem 2.6.3 Which search algorithm would we use if we want to minimize train changes (assuming that we change train at every station)?



To minimize train changes, we have to find the shallowest solution. Breadth-First search finds this solution.

Problem 2.6.4: Is bidirectional search a good option for the train journey search?



Yes, as all actions are reversible and there is only one goal, we can use bidirectional search. Since the cost in time and price are the same in both directions, the backwards search can be performed just like the forward search.

$$\text{i.e. } c(A, B) = c(B, A)$$

2 Additional Problems

Problem 2.7: Graph Search

Consider the graph of Problem 2.1 (of Exercise 2a). With the heuristic given in the table below, perform Greedy Best-First and A* Search, each with tree search and with graph search. For each step, write down the node which is currently expanded and the nodes in the frontier set, each with its value of the evaluation function $f(n)$ (and the nodes in the explored set in case of graph search).

node n	A	B	C	D	E	F
heuristic function $h(n)$	4	5	2	1	7	0

Greedy Best-First tree search:

$$f(n) = h(n)$$

node with $f(n)$	A(4)	D(1)	F(0)
frontier with $f(n)$	D(1), C(2), B(5)	F(0), C(2), A(4), B(5)	C(2), A(4), B(5)

A* tree search:

$$f(n) = g(n) + h(n)$$

node with $f(n)$	A(4 = 0 + 4)	C(4 = 2 + 2)	D(5 = 4 + 1)	F(5 = 5 + 0)
frontier with $f(n)$	C(4 = 2 + 2), D(5 = 4 + 1), B(6 = 1 + 5)	D(5 = 4 + 1), B(6 = 1 + 5), A(8 = 4 + 4)	F(5 = 5 + 0), B(6 = 1 + 5), A(8 = 4 + 4), A(12 = 8 + 4)	B(6 = 1 + 5), A(8 = 4 + 4), A(12 = 8 + 4)

Greedy First-Best graph search:

node with $f(n)$	A(4)	D(1)	F(0)
frontier with $f(n)$	D(1), C(2), B(5)	F(0), C(2), B(5)	C(2), B(5)
explored	A	AD	AD



A* graph search:

node with $f(n)$	A(4 = 0 + 4)	C(4 = 2 + 2)	D(5 = 4 + 1)	F(5 = 5 + 0)
frontier with $f(n)$	C(4 = 2 + 2), D(5 = 4 + 1), B(6 = 1 + 5)	D(5 = 4 + 1), B(6 = 1 + 5)	F(5 = 5 + 0), B(6 = 1 + 5)	B(6 = 1 + 5)
explored	A	AC	ACD	ACD



Problem 2.8: General Questions on Search

Problem 2.8.1 Can Uniform-Cost search be more time-effective or memory-effective than the informed search algorithms mentioned in the lectures? What about cost-effectiveness?

Greedy Best-First is not optimal, so may be less cost-effective than Uniform-Cost, and may be less time-effective and memory-effective, since the heuristic may lead the algorithm onto paths that appear to be dead ends. A* is optimal, so will give the same (optimal) cost as Uniform-Cost. Similarly, with a bad heuristic (e.g. $h = 0$ for all nodes), A* can perform as badly in memory or time as Uniform-Cost, but not worse (since for $h \geq 0$, any nodes explored in A* will be explored in Uniform Cost, but not vice-versa.)

Problem 2.9: Application of Search Algorithms in Daily Life

Which search algorithm (out of those covered in lectures 3 and 4) do you think your brain uses in the following cases, and what are the advantages of this algorithm? (Note that there are no hard-and-fast solutions for these; it is a matter of personal preference. The purpose of this exercise is to think about situations where you use search algorithms in daily life.)

a. Planning a route from Arad to Bucharest with a map.

Since you can see the whole map, you are informed and have some sort of heuristic (you would tend to prefer the roads leading towards Bucharest, for example). Therefore, you might first use Greedy First-Best, as you do not know the exact path cost; perhaps also using Bidirectional Search. To find the optimal solution, you might use A* or Uniform-Cost afterwards.

b. Finding an Easter egg:

- i. in a building you are unfamiliar with, on your own.
- ii. in a building you are familiar with, on your own.
- iii. in a building you are familiar with, with a team of people who all have mobile phones.
- iv. if the Easter egg has a bell attached (which constantly rings).

→ On your own, only nodes close together can be explored (you cannot jump halfway across the building to explore another node). Therefore, some sort of Depth-First search would be sensible.

i. If you do not know the building, you may opt for Depth-Limited Search (as you do not want to get lost) or Iterative Deepening. If you do know the building, you may opt for Depth-First Search, Depth-Limited Search, or Iterative Deepening.

iii. If you have a team of people, nodes all along the frontier can be explored. You could opt for Breadth-First Search.

iv. With the attached bell, you have a heuristic - how loud or soft the sound is. On your own, perhaps you would prefer Greedy First-Best, as it limits the amount of backtracking you need to do. If you have a team of people, you could perform A* Search.

c. Playing a strategy game, e.g. chess.

This is very individual! It is left to you to decide what you do. Personally, I use some kind of Depth-Limited Search.

A*