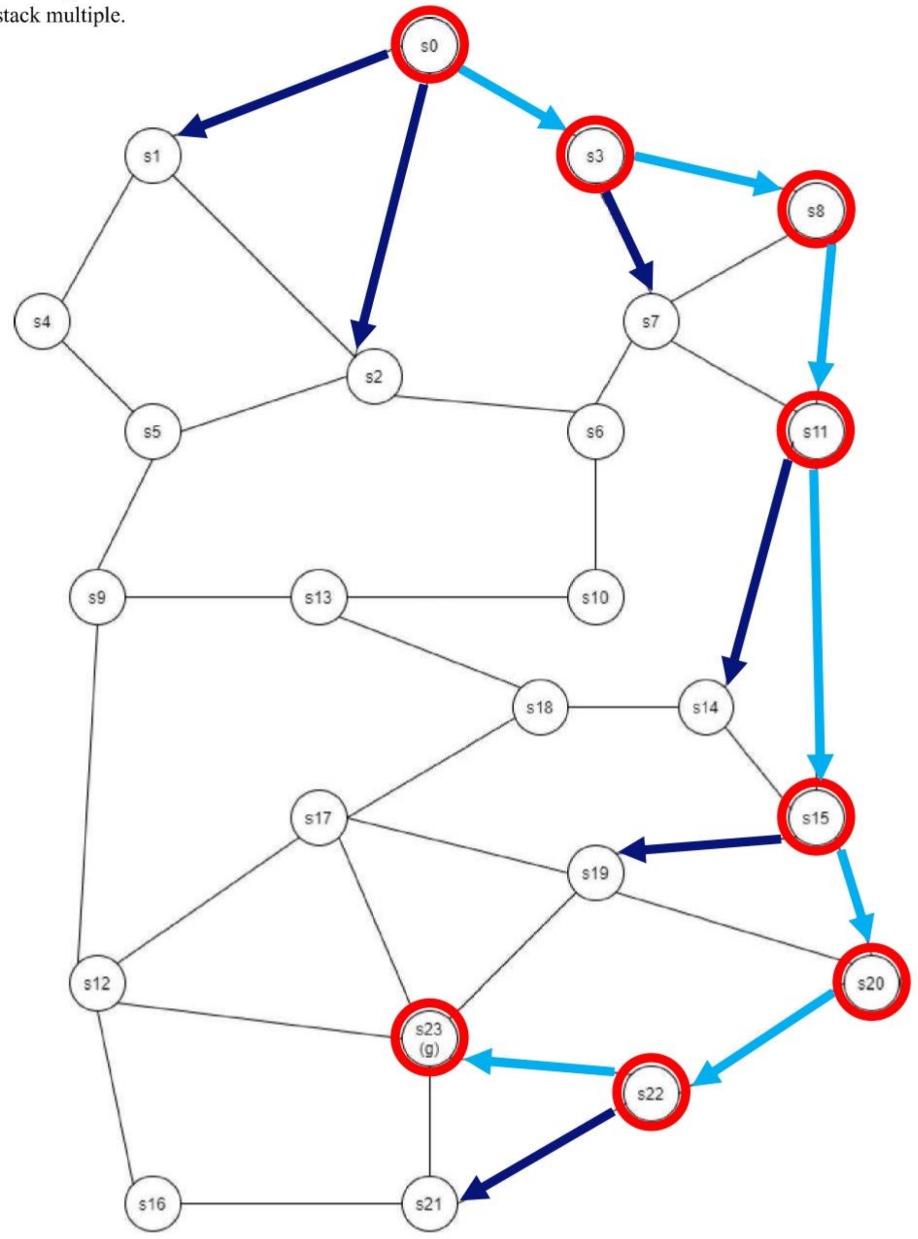
Exercise 4.1.1: Depth-first search (DFS) algorithm

Using depth-first search (DFS) algorithm

by stacking the lowest of the s numbers when it is possible to stack multiple.



In the figure above, we show how we ran DFS to find a route from city s0 to s23. Blue edges point from expanded nodes, which are designated by red circles, to generated nodes. Cyan edges represent the path of the found route.

Below is the process log after execution. It shows the steps taken by the algorithm, specifying which nodes were expanded as well as the status of the data structure after every expansion:

```
<s0>: [<s1>, <s2>, <s3>]

<s3>: [<s1>, <s2>, <s7>, <s8>]

<s8>: [<s1>, <s2>, <s7>, <s11>]

<s11>: [<s1>, <s2>, <s7>, <s14>, <s15>]

<s15>: [<s1>, <s2>, <s7>, <s14>, <s19>, <s20>]

<s20>: [<s1>, <s2>, <s7>, <s14>, <s19>, <s22>]

<s22>: [<s1>, <s2>, <s7>, <s14>, <s19>, <s21>, <s23>]

<s23>: [<s1>, <s2>, <s7>, <s14>, <s19>, <s21>]
```

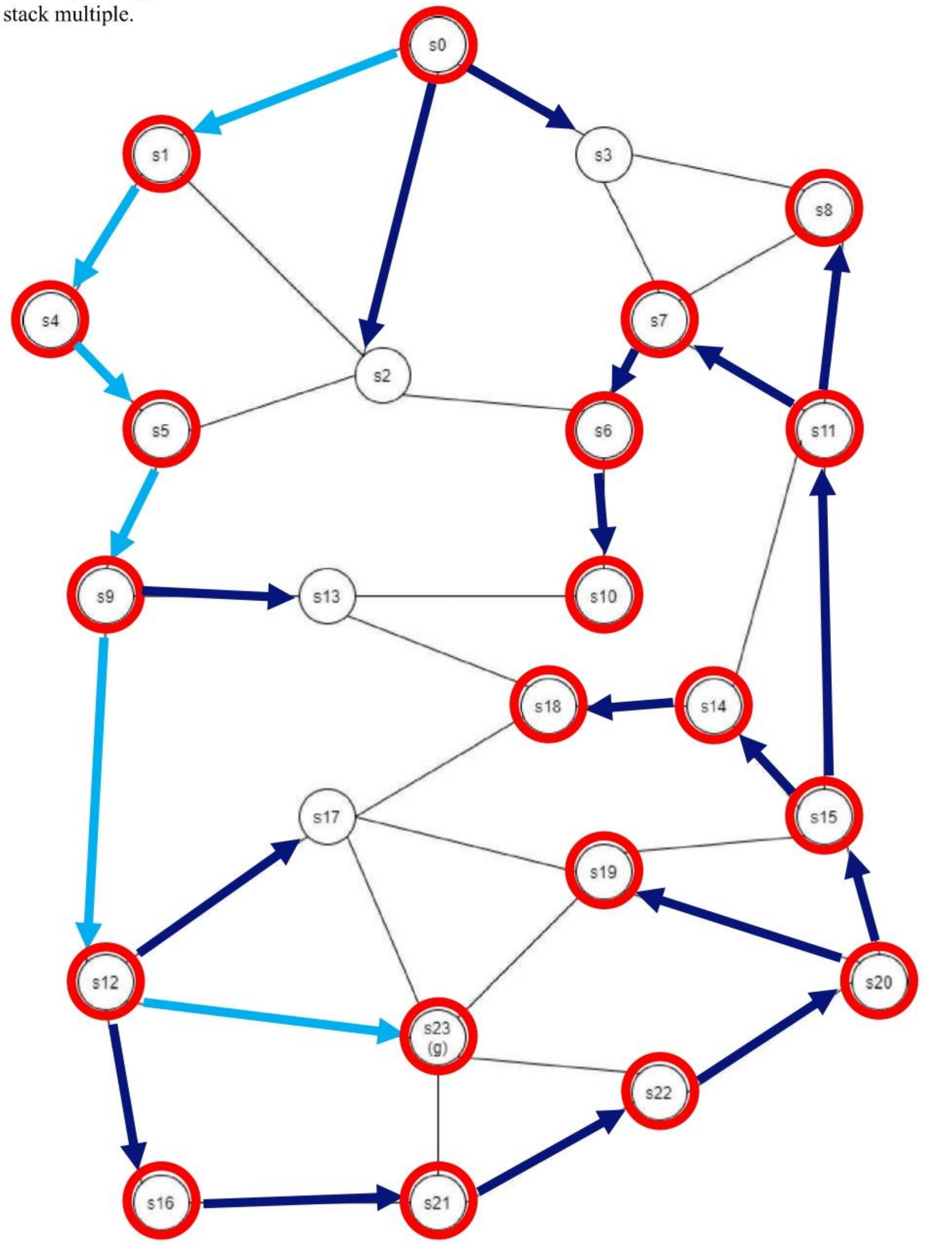
The same scheme is used in all exercises. In this exercise, when it is possible to stack multiple nodes, we stack the lowest of the s numbers. The nodes are stacked in LIFO (Last In, First Out) order. The following table shows further information about the execution:

Total number of expanded nodes	8
Total number of generated frontiers	13
Length of the found path	7 (assuming each step costs 1)

Exercise 4.1.2: Depth-first search (DFS) algorithm

Using depth-first search (DFS) algorithm

by stacking the highest of the s numbers when it is possible to stack multiple.



We do the same procedure here as well.

<s0>: [<s3>, <s2>, <s1>]

<s1>: [<s3>, <s2>, <s4>]

<s4>: [<s3>, <s2>, <s5>]

<s5>: [<s3>, <s2>, <s9>]

<s9>: [<s3>, <s2>, <s13>, <s12>]

<s12>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s16>]

<s16>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s21>]

<s21>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s22>]

<s22>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s20>]

<s20>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s15>]

<s15>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s14>, <s11>]

<s11>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s14>, <s8>, <s7>]

<s7>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s14>, <s8>, <s6>]

<s6>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s14>, <s8>, <s10>]

<s10>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s14>, <s8>]

<s8>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s14>]

<s14>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>, <s18>]

<s18>: [<s3>, <s2>, <s13>, <s23>, <s17>, <s19>]

<s19>: [<s3>, <s2>, <s13>, <s23>, <s17>]

<s17>: [<s3>, <s2>, <s13>, <s23>]

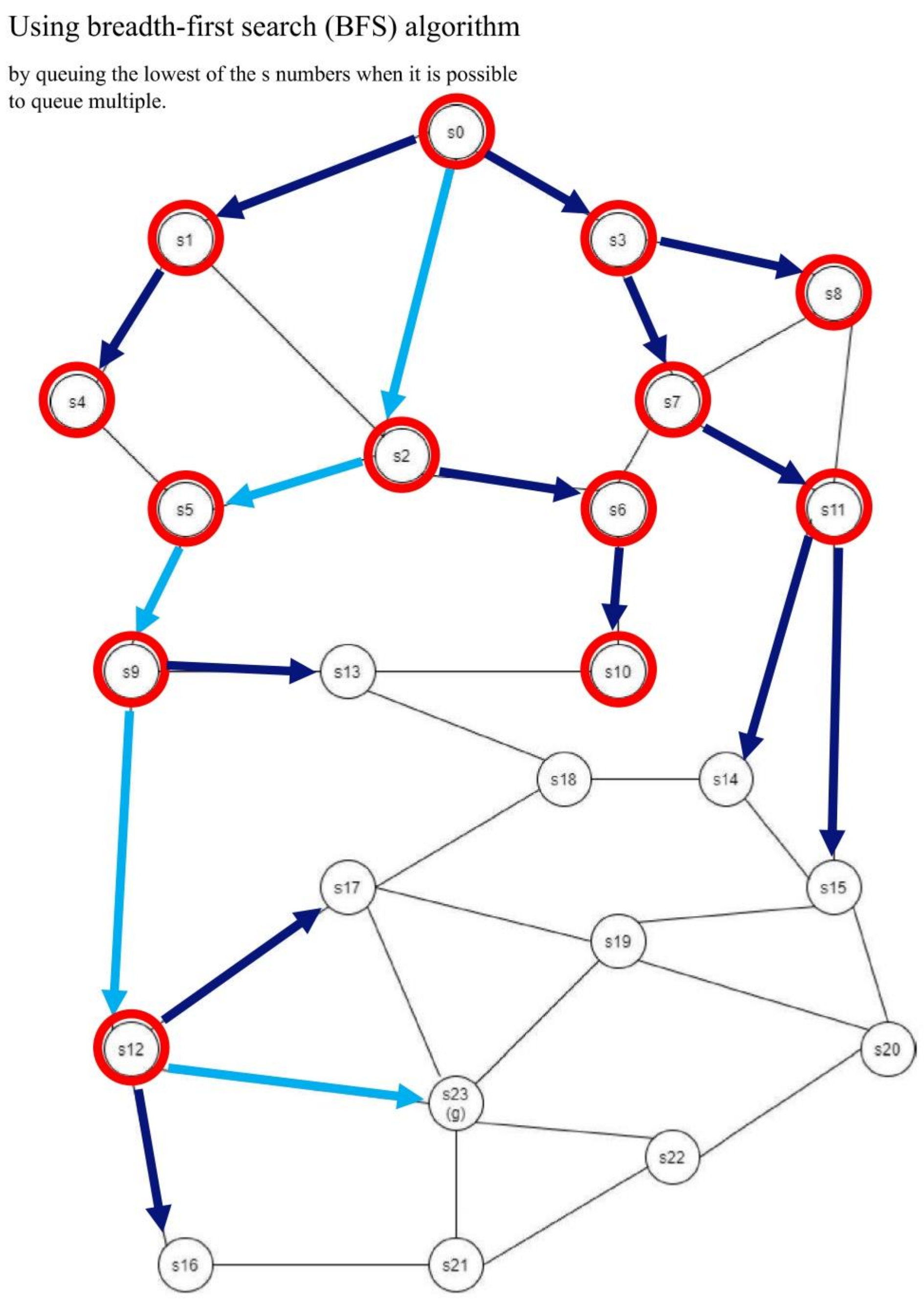
<s23>: [<s3>, <s2>, <s13>]

Total number of expanded nodes	21
Total number of generated frontiers	23
Length of the found path	6 (assuming each step costs 1)

<u>Exercise 4.1.3: Depth-first search (DFS) algorithm</u>

DFS is not optimal. The reason we get 1 route longer than the other is because we chose a different way to stack our nodes. DFS proceeds by expanding the deepest node until no successors are found. Nodes with no successors are dropped from the frontier once expanded, and the algorithm searches the next deepest node that still has unexplored successors. In the first run, we expanded through s3, whereas in the second run, we expanded through s1. Thus, we get 2 different routes.

Exercise 4.2.1: Breadth-first search (BFS) algorithm


Using breadth-first search (BFS) algorithm
by queuing the lowest of the s numbers when it is possible to queue multiple.
s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
s13
s14
s15
s16
s17
s18
s19
s20
s21
s22
s23 (g)

<s0>: [<s1>, <s2>, <s3>]

<s1>: [<s2>, <s3>, <s4>]

<s2>: [<s3>, <s4>, <s5>, <s6>]

<s3>: [<s4>, <s5>, <s6>, <s7>, <s8>]

<s4>: [<s5>, <s6>, <s7>, <s8>]

<s5>: [<s6>, <s7>, <s8>, <s9>]

<s6>: [<s7>, <s8>, <s9>, <s10>]

<s7>: [<s8>, <s9>, <s10>, <s11>]

<s8>: [<s9>, <s10>, <s11>]

<s9>: [<s10>, <s11>, <s12>, <s13>]

<s10>: [<s11>, <s12>, <s13>]

<s11>: [<s12>, <s13>, <s14>, <s15>]

<s12>: [<s13>, <s14>, <s15>, <s16>, <s17>, <s23>]

The nodes are stacked in FIFO (First In, First Out) order.

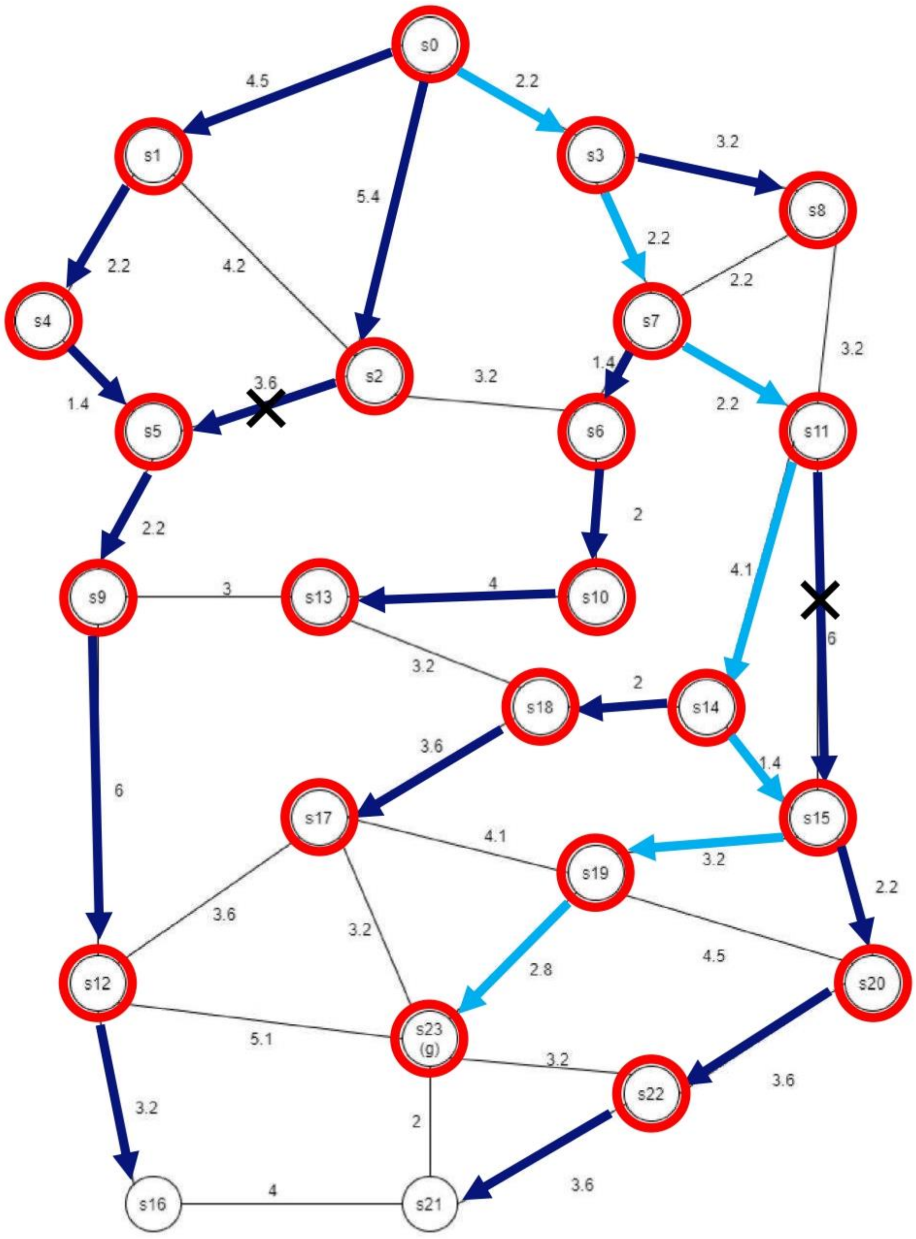
Total number of expanded nodes	13
Total number of generated frontiers	18
Length of the found path	5 (assuming each step costs 1)

Exercise 4.2.2: Breadth-first search (BFS) algorithm

BFS always finds a solution. It is optimal for unit step costs. Unfortunately, it has exponential time complexity. This means that it may not be able to find a solution quickly when working on large finite state spaces. DFS on the other hand may never find a solution. It is not optimal. Yet, it has linear space complexity. This means that it may find a solution even when working on large finite spaces, since it will require less memory. So, when working with robots, DFS is preferred, because the hardware will at least be able to cope with real search problems, and not obstructed by memory demands.

Exercise 4.3: Dijkstra's algorithm

Using Dijkstra's algorithm



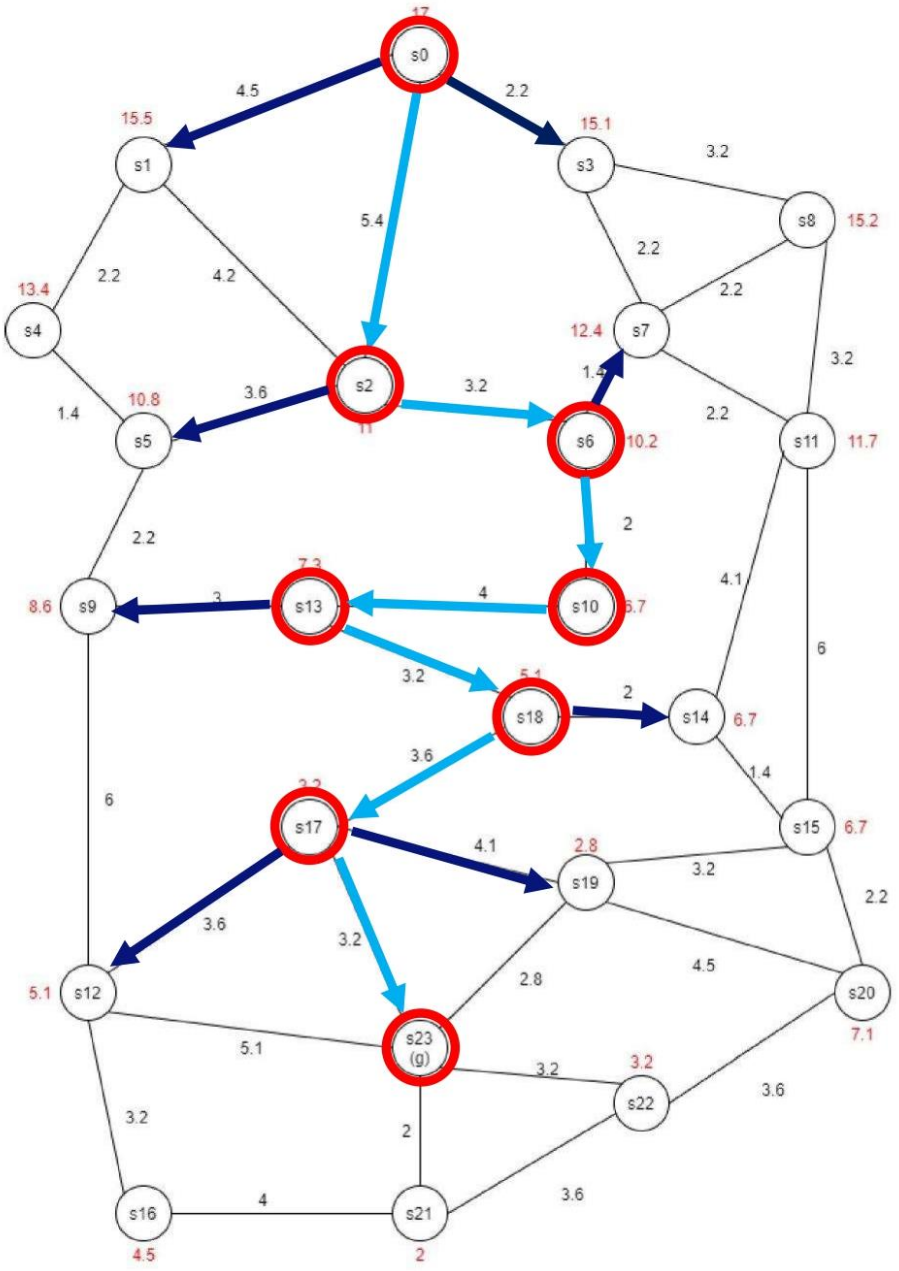
```
<s0>: [(2.2, <s3>), (5.4, <s2>), (4.5, <s1>)]

<s3>: [(4.4, <s7>), (5.4, <s2>), (4.5, <s1>), (5.4, <s8>)]

<s7>: [(4.5, <s1>), (5.4, <s2>), (5.4, <s8>), (5.8, <s6>), (6.6, <s11>)]

<s1>: [(5.4, <s2>), (5.8, <s6>), (5.4, <s8>), (6.6, <s11>), (6.7, <s4>)]

<s2>: [(5.4, <s8>), (5.8, <s6>), (6.7, <s4>), (6.6, <s11>), (9.0, <s5>)]

<s8>: [(5.8, <s6>), (6.6, <s11>), (6.7, <s4>), (9.0, <s5>)]

<s6>: [(6.6, <s11>), (7.8, <s10>), (6.7, <s4>), (9.0, <s5>)]

<s11>: [(6.7, <s4>), (7.8, <s10>), (9.0, <s5>), (10.7, <s14>), (12.6, <s15>)]

<s4>: [(7.8, <s10>), (8.1, <s5>), (12.6, <s15>), (10.7, <s14>)]

<s10>: [(8.1, <s5>), (10.7, <s14>), (12.6, <s15>), (11.8, <s13>)]

<s5>: [(10.3, <s9>), (10.7, <s14>), (12.6, <s15>), (11.8, <s13>)]

<s9>: [(10.7, <s14>), (11.8, <s13>), (12.6, <s15>), (16.3, <s12>)]

<s14>: [(11.8, <s13>), (12.7, <s18>), (12.1, <s15>), (16.3, <s12>)]

<s13>: [(12.1, <s15>), (12.7, <s18>), (16.3, <s12>)]

<s15>: [(12.7, <s18>), (14.3, <s20>), (15.3, <s19>), (16.3, <s12>)]

<s18>: [(14.3, <s20>), (16.3, <s12>), (15.3, <s19>), (16.3, <s17>)]

<s20>: [(15.3, <s19>), (16.3, <s12>), (16.3, <s17>), (17.9, <s22>)]

<s19>: [(16.3, <s12>), (17.9, <s22>), (16.3, <s17>), (18.1, <s23>)]

<s12>: [(16.3, <s17>), (17.9, <s22>), (18.1, <s23>), (19.5, <s16>)]

<s17>: [(17.9, <s22>), (19.5, <s16>), (18.1, <s23>)]

<s22>: [(18.1, <s23>), (19.5, <s16>), (21.5, <s21>)]

<s23>: [(19.5, <s16>), (21.5, <s21>)]
```

The nodes with the lowest path cost are expanded first.

Total number of expanded nodes	22
Total number of generated frontiers	25
Length of the found path	18.1

Exercise 4.4: Greedy best-first search algorithm

Using greedy best-first search (GBFS) algorithm



<s0>: [(11, <s2>), (15.5, <s1>), (15.1, <s3>)]

<s2>: [(10.2, <s6>), (10.8, <s5>), (15.1, <s3>), (15.5, <s1>)]

<s6>: [(6.7, <s10>), (10.8, <s5>), (15.1, <s3>), (15.5, <s1>), (12.4, <s7>)]

<s10>: [(7.3, <s13>), (10.8, <s5>), (15.1, <s3>), (15.5, <s1>), (12.4, <s7>)]

<s13>: [(5.1, <s18>), (10.8, <s5>), (8.6, <s9>), (15.5, <s1>), (12.4, <s7>), (15.1, <s3>)]

<s18>: [(3.2, <s17>), (10.8, <s5>), (6.7, <s14>), (15.5, <s1>), (12.4, <s7>), (15.1, <s3>), (8.6, <s9>)]

<s17>: [(0, <s23>), (2.8, <s19>), (6.7, <s14>), (5.1, <s12>), (12.4, <s7>), (15.1, <s3>), (8.6, <s9>), (15.5, <s1>), (10.8, <s5>)]

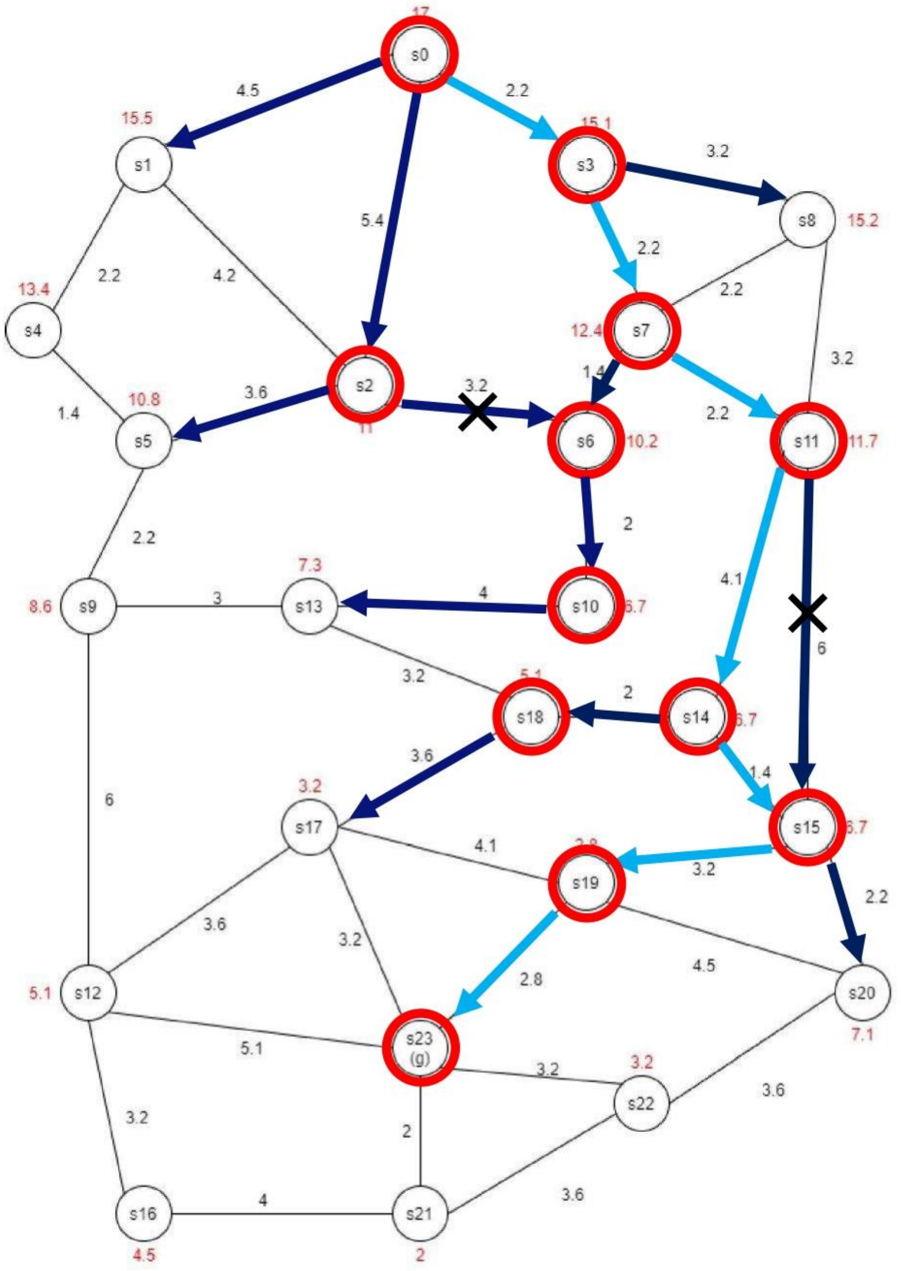
<s23>: [(2.8, <s19>), (6.7, <s14>), (5.1, <s12>), (12.4, <s7>), (15.1, <s3>), (8.6, <s9>), (15.5, <s1>), (10.8, <s5>)]

The nodes with the lowest distance to the target are expanded first.

Total number of expanded nodes	8
Total number of generated frontiers	15
Length of the found path	24.6

Exercise 4.5: A* search algorithm

Using A* search algorithm



<s0>: [(16.4, <s2>), (20.0, <s1>), (17.3, <s3>)]

<s2>: [(17.3, <s3>), (18.8, <s6>), (19.8, <s5>), (20.0, <s1>)]

<s3>: [(16.8, <s7>), (18.8, <s6>), (19.8, <s5>), (20.0, <s1>), (20.6, <s8>)]

<s7>: [(16.0, <s6>), (18.3, <s11>), (20.6, <s8>), (20.0, <s1>), (19.8, <s5>)]

<s6>: [(14.5, <s10>), (18.3, <s11>), (20.6, <s8>), (20.0, <s1>), (19.8, <s5>)]

<s10>: [(18.3, <s11>), (19.1, <s13>), (20.6, <s8>), (20.0, <s1>), (19.8, <s5>)]

<s11>: [(17.4, <s14>), (19.1, <s13>), (19.3, <s15>), (20.0, <s1>), (19.8, <s5>), (20.6, <s8>)]

<s14>: [(17.8, <s18>), (19.1, <s13>), (18.8, <s15>), (20.6, <s8>), (19.8, <s5>), (20.0, <s1>)]

<s18>: [(18.8, <s15>), (19.1, <s13>), (19.5, <s17>), (20.6, <s8>), (19.8, <s5>), (20.0, <s1>)]

<s15>: [(18.1, <s19>), (19.8, <s5>), (19.1, <s13>), (20.6, <s8>), (20.0, <s1>), (19.5, <s17>), (21.4, <s20>)]

<s19>: [(18.1, <s23>), (19.8, <s5>), (19.1, <s13>), (20.6, <s8>), (20.0, <s1>), (21.4, <s20>), (19.5, <s17>)]

<s23>: [(19.8, <s5>), (19.1, <s13>), (20.6, <s8>), (20.0, <s1>), (21.4, <s20>), (19.5, <s17>)]

The nodes with the cheapest solution, which is a sum of the path cost and the distance to the target, are expanded first.

Total number of expanded nodes	12
Total number of generated frontiers	19
Length of the found path	18.1

Indeed, the route and the path length are similar to those found by Dijkstra's. The advantage of using A* however is that it expands less nodes and generates less frontiers. This can be important when working on large finite state spaces.