Item Navigation

Congratulations! You passed!

Grade received 100% **Latest Submission Grade** 100% **To pass** 80% or higher

Go to next item

Branch and Bound

Quiz • 1h

Review Learning Objectives

1. Implement the Branch-and-Bound algorithm for the Traveling Salesman problem. **1 / 1 point**

Submit your assignment

Due Jan 8, 11:59 PM IST

Try again

Receive grade

To Pass 80% or higher

Your grade

100%

View Feedback

We keep your highest score

Like **Dislike** **Report an issue**

```
46    # Initially sub_cycle is empty;
47    # 3. currently best solution current_min, so that we don't even consider paths of greater weight.
48    # Initially the min weight is infinite
49    def branch_and_bound(g, sub_cycle=None, current_min=float("inf")):
50        # If the current path is empty, then we can safely assume that it starts with the vertex 0.
51        if sub_cycle is None:
52            sub_cycle = [0]
53
54        # If we already have all vertices in the cycle, then we just compute the weight of this cycle and return it.
55        if len(sub_cycle) == g.number_of_nodes():
56            weight = sum([g[sub_cycle[i]][sub_cycle[i + 1]]['weight'] for i in range(len(sub_cycle) - 1)])
57            weight = weight + g[sub_cycle[-1]][sub_cycle[0]]['weight']
58            return weight
59
60        # Now we look at all nodes which aren't yet used in sub_cycle.
61        unused_nodes = list()
62        for v in g.nodes():
63            if v not in sub_cycle:
64                unused_nodes.append((g[sub_cycle[-1]][v]['weight'], v))
65
66        # We sort them by the distance from the "current node" -- the last node in sub_cycle.
67        unused_nodes = sorted(unused_nodes)
68
69        for (d, v) in unused_nodes:
70            assert v not in sub_cycle
71            extended_subcycle = list(sub_cycle)
72            extended_subcycle.append(v)
73            # For each unused vertex, we check if there is any chance to find a shorter cycle if we add it now.
74            if lower_bound(g, extended_subcycle) < current_min:
75                new_min = branch_and_bound(g, sub_cycle + [v], current_min)
76                if new_min < current_min:
77                   current_min = new_min
78                # WRITE YOUR CODE HERE
79                # If there is such a chance, we add the vertex to the current cycle, and proceed recursively.
80                # If we found a short cycle, then we update the current_min value.
81
82
83        # The procedure returns the shortest cycle length.
84        return current_min
85
```

Run

Reset

```
No Output
```

Correct

Good job!