Algorithms & Datastructures
Laboratory Exercise Sheet 12

Wolfgang Pausch <wolfgang.pausch@uibk.ac.at>
Heiko Studt <heiko.studt@uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>
Tomas Vitvar <tomas.vitvar@uibk.ac.at>

June 15th, to be discussed on June 22nd

Exercise 1) State Space Search The graph in Figure 1 represents a state space where vertices $0, 1, 2, ..., 19$ denote states and edges denote transitions between states. Each state transition has associated costs of that transition (which can be ignored for this exercise, except for `SuccessorFn`). For the following implementation tasks use and/or modify the code skeleton given below.

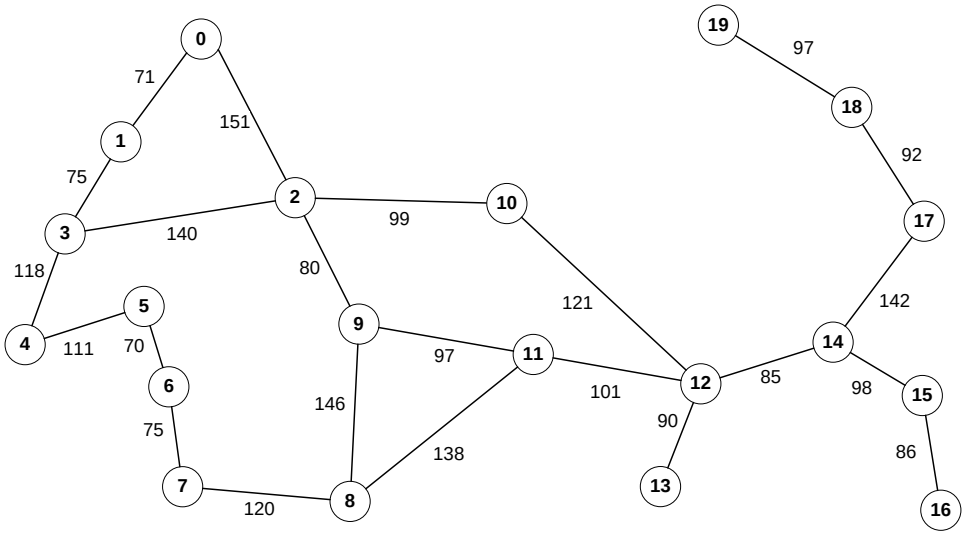

Figure 1: State Space

1. Implement a method `SuccessorFn(State x)` which returns a set of successor states for the state `x`. Note, that the state space is a non-directed graph, thus the successor function should always take into account the complete incidence matrix.

2. Implement a method `findPath_BFS(int initial, int goal)` using breadth-first search and `findPath_DFS(int initial, int goal)` using depth-first search. The methods should return a path (i.e, sequence of states) from an initial state to a goal state together with the total length. Use previously implemented successor function in both implementations and take care of repeating states.

3. What happens if you do not take care of repeating states?

4. Implement a method `findPath_DFS(int initial, int goal, int depth_limit)` using a depth-limited search. Depth-limited search is a depth-first search with some predetermined depth limit (that is, the algorithm stops when it reaches the depth limit in the search tree).

5. Implement a method `findPath_IDS(State initial, State goal)` using iterative deepening depth-first search. Iterative deepening depth-first search is a strategy that uses depth-limited search and iteratively increases the limit until it finds a solution.

6. Prove that iterative deepening depth-first search is faster than breath-first search.

```
public class State {

    int id;
    // implement me!
}
```

```
public class StateSpace {

    int[][] data;

    public StateSpace() {
        data = new int[20][20];

        data[0][1]   = 71;  data[0][2]   = 151;
        data[1][3]   = 75;  data[2][3]   = 140;
        data[3][4]   = 118; data[4][5]   = 111;
        data[5][6]   = 70;  data[6][7]   = 75;
        data[7][8]   = 120; data[8][9]   = 146;
        data[2][9]   = 80;  data[2][10]  = 99;
        data[9][11]  = 97;  data[8][11]  = 138;
        data[11][12] = 101; data[10][12] = 121;
        data[12][13] = 90;  data[12][14] = 85;
        data[14][15] = 98;  data[15][16] = 86;
        data[14][17] = 142; data[17][18] = 92;
        data[18][19] = 97;

        data[1][0]   = 71;  data[2][0]   = 151;
        data[3][1]   = 75;  data[3][2]   = 140;
        data[4][3]   = 118; data[5][4]   = 111;
        data[6][5]   = 70;  data[7][6]   = 75;
        data[8][7]   = 120; data[9][8]   = 146;
        data[9][2]   = 80;  data[10][2]  = 99;
        data[11][9]  = 97;  data[11][8]  = 138;
        data[12][11] = 101; data[12][10] = 121;
        data[13][12] = 90;  data[14][12] = 85;
        data[15][14] = 98;  data[16][15] = 86;
        data[17][14] = 142; data[18][17] = 92;
        data[19][18] = 97;
    }

    public void print() {
        System.out.print(String.format("%3s |"," "));
        for (int i = 0; i < data[0].length; i++)
            System.out.print(String.format("%5d",i));
        System.out.println();
        for (int i = 0; i < data[0].length + 1; i++)
```

```
            System.out.print(String.format("%s","-----"));
        System.out.println();

        for (int i = 0; i < data[0].length; i++) {
            System.out.print(String.format("%3d |",i));
            for (int j = 0; j < data[i].length; j++) {
                System.out.print(String.format("%5d",data[i][j]));
            }
            System.out.println();
        }
    }

    public List<State> successorFn(State x) {
        // implement me!
    }

    public State findPath_DFS(int initial, int goal) {
        // implement me!
    }

    public State findPath_DFS(int initial, int goal, int depth_limit) {
        // implement me!
    }

    public State findPath_BFS(int initial, int goal) {
        // implement me!
    }

    public State findPath_IDS(int initial, int goal) {
        // implement me!
    }

}
```

Exercise 2) Strongly Connected Components

1. Given an example of a graph having at least two strongly connected components.
2. Apply the algorithm from the lecture to find all strongly connected components of the graph given in Figure 2. Show the steps of the algorithm.

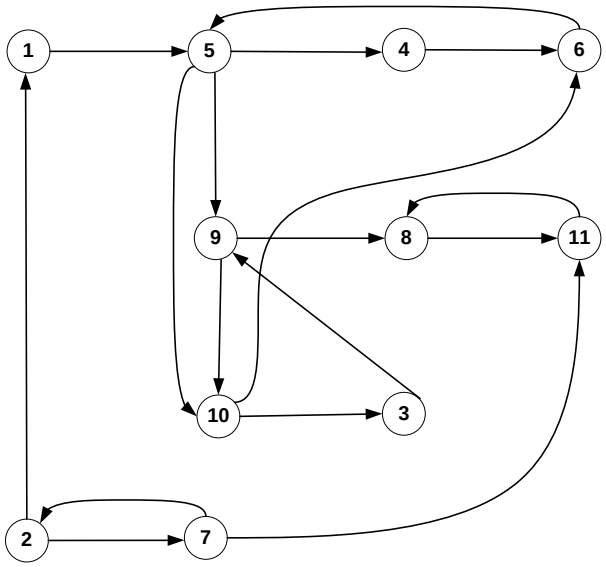

Figure 2: Graph