Problem description

A company has several vehicles at different locations. Each one of those vehicles is going to make a trip from an origin to a destination, with a certain available capacity and duration.

Our task is to assemble a system from the ground, capable of supporting the management of requests for transportation of groups of people from an origin location to a destination location, both being given.

General Formalization

All throughout the project, the following data structures are commonly used:

- A graph, **g**, made up of a list of nodes (places)
- A list of nodes, **nodes**, comprising a list of edges (routes)
- A list of edges, **adj**, containing the follow elements:
    - Origin node
    - Destination node
    - Duration of the route
    - Capacity of the vehicle

Scenario 1

In the first two scenarios, we are dealing with groups of people that do **not** split.

Scenario 1_1

The first subset of the scenario 1 aims to maximize the size of the group, for a given origin and destination nodes.

Formalization of Scenario 1_1

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The maximum group size, **max_group_size**;
- The paths from **start** to **target**, **routes;**

Restrictions:

- Given a route, **R**, with the most capacity out of a list of routes between **start** and **target**, max_group_size cannot be greater than R's capacity.

Objective:

- Output the **maximum size** of the group and a possible routing from **start** to **target**.

Relevant Algorithms and Complexity

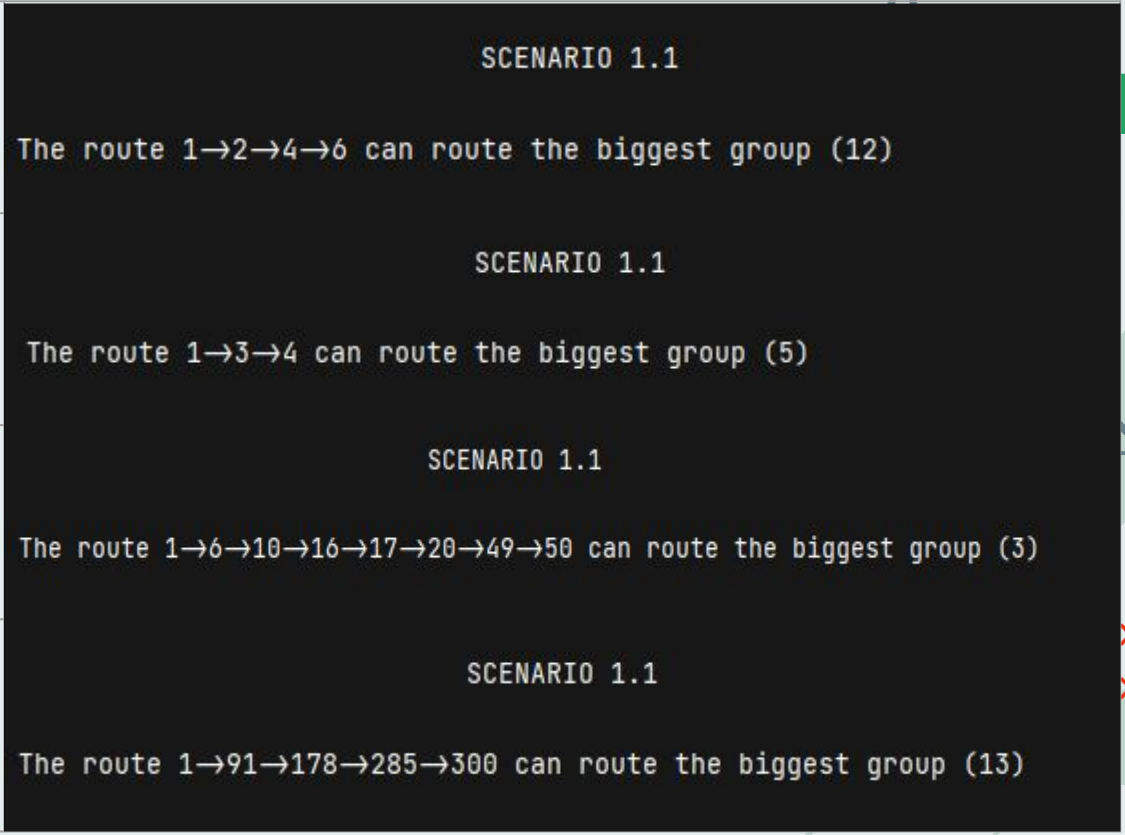
Algorithm Chosen:

- Maxmin path adaptation:
  - Given a directed graph **g,** a start node **s** and a end node **t**, finds, out of all possible routes between **s** and **t**, the one with the biggest capacity.
  - Array with capacities, **cap**, with all values initialized to 0, with the exception of the value at **cap[s]**, initialized with +∞.
  - Queue of graph nodes, **q.**
    - While q is not empty:
      - Node **v** = q.pop()
      - For each edge **e**, in the node **v** adjacency list:
        - If MIN(cap[e.origin], e.cap) > cap[e.dest]
          - cap[e.dest] = MIN(cap[e.origin], e.cap)
          - if e.dest == g.size - 1
            - max_group_size = cap[e.dest]

Complexity:

- O(n^2)

Results



in12.txt	`SCENARIO 1.1` `The route 1→2→4→6 can route the biggest group (12)`
in11.txt	`SCENARIO 1.1` `The route 1→3→4 can route the biggest group (5)`
in2.txt	`SCENARIO 1.1` `The route 1→6→10→16→17→20→49→50 can route the biggest group (3)`
in3.txt	`SCENARIO 1.1` `The route 1→91→178→285→300 can route the biggest group (13)`

Scenario 1_2

The goal is to maximize the group dimension while also minimizing the number of transfers needed to reach the desired destination.

No priority should be given between the above criteria, and alternatives presented should be non-comparables. That means that a bigger group may be transported even if there are more transfers in that journey.

Formalization of Scenario 1_2

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The paths from **start** to **target**, **routes;**
- The capacity of each of the routes, **cap;**
- The number of transfers in each of the routes, **transf;**

Restrictions:

- Output routes should be pareto-optimal, i.e, non comparable between themselves.

Objective:

- Output several alternative routes that take a group from **start** to **target**.

Relevant Algorithms and Complexity 1/2

Algorithm Chosen and steps taken:

- Depth-first search(**dfs**) variant:
    - Given a directed graph **g,** a start node **s** and a end node **t**, finds a list of routes between **s** and **t**.
    - Recursive algorithm:
        - Stopping criteria:
            - Found the last possible route between **s** and **t**.
        - For each edge **e**, in the node **s** adjacency list:
            - old capacity = capacity
            - Add **e.destination** (child) to path
            - capacity = MIN(**e.capacity, capacity**)
            - Run the algorithm for the child node
            - capacity = old capacity
            - Remove child from path in order to be able to explore other paths.
- After running the algorithm, we are met with a list of possible routes between **s** and **t**, and need to delete the ones that can be comparable
    - Run the previous scenario to find the route with the biggest capacity, **r**
    - Sort routes from highest to lowest **capacity**.
    - Eliminate routes that have a **greater or equal** number of transfers and that are different from the route **r**.
    - … (next slide)

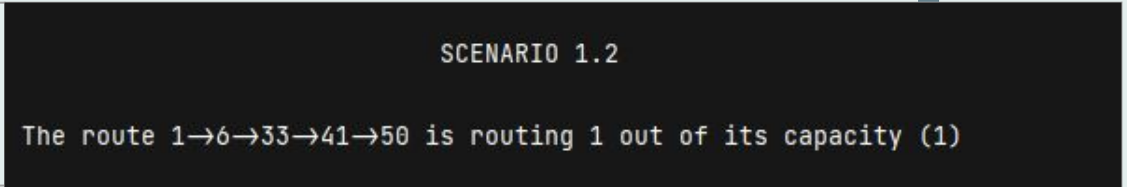
Relevant Algorithms and Complexity 2/2

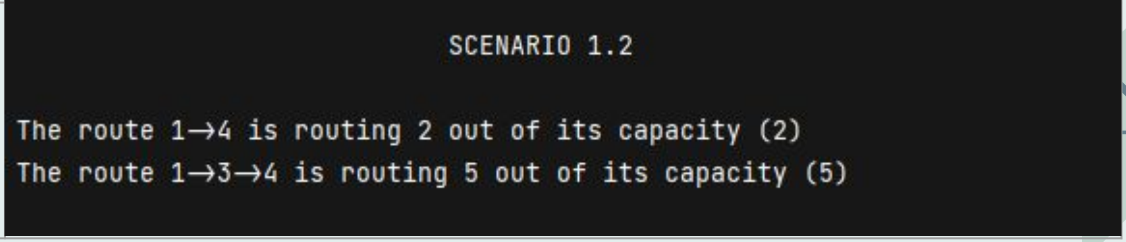
- Create a HashMap to link each key (**capacity**) to a value (**route**)
- For route **r** in the list of routes
    - If **r**’s capacity is not yet on the hashmap and **r**’s number of transfers is less than the current minimum of transfers
        - Add **r**’s capacity and **r** to the map
        - **r**’s number of transfers is now the minimum
    - If **r**’s capacity is not a key in the hashmap
        - Add **r**’s capacity and **r** to the map

Complexity:

- **O(v^v)** where **v** is the number of nodes

Results

in1.txt	 `SCENARIO 1.2` `The route 1→6→33→41→50 is routing 1 out of its capacity (1)`

in11.txt	 `SCENARIO 1.2`
	`The route 1→4 is routing 2 out of its capacity (2)`
	`The route 1→3→4 is routing 5 out of its capacity (5)`

Scenario 2

In the remaining scenarios, we are dealing with groups of people that **can be** split.

Scenario 2_1

This scenario’s purpose is to, given a group size, initial and final nodes, output a possible route that the group can take to reach the desired end node.

Formalization of Scenario 2_1

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The group size, **group_size**;
- The paths from **start** to **target**, **routes;**

Restrictions:

- The **group_size** cannot exceed the maximum flow of the network;

Objective:

- Output the route(s) that the group should take to get from a **start** node to a **target** node.

Relevant Algorithms and Complexity

Algorithm Chosen:

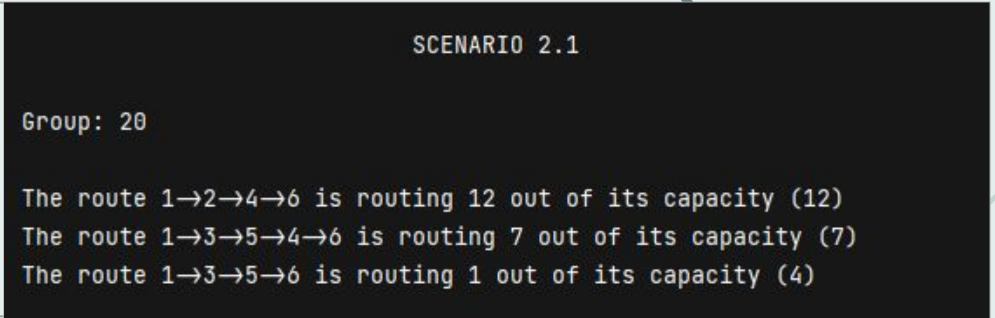
- Ford-Fulkerson algorithm

Complexity:

- **O(max_flow * E)**, where **E** is the number of edges in the graph, **g**.

Results

in12.txt

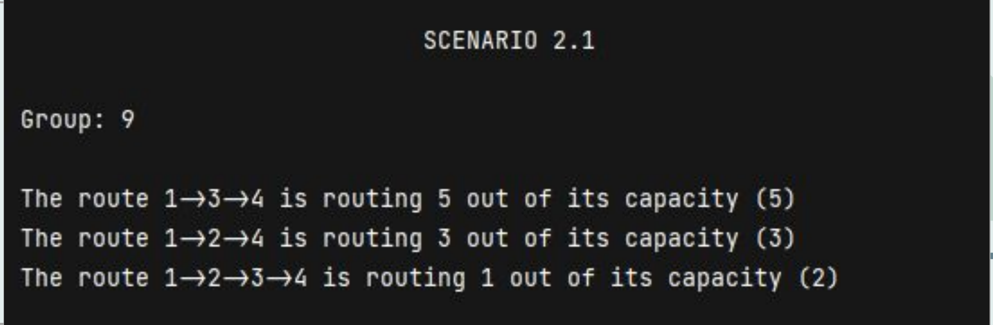


```
                          SCENARIO 2.1

Group: 20

The route 1→2→4→6 is routing 12 out of its capacity (12)
The route 1→3→5→4→6 is routing 7 out of its capacity (7)
The route 1→3→5→6 is routing 1 out of its capacity (4)
```

in11.txt

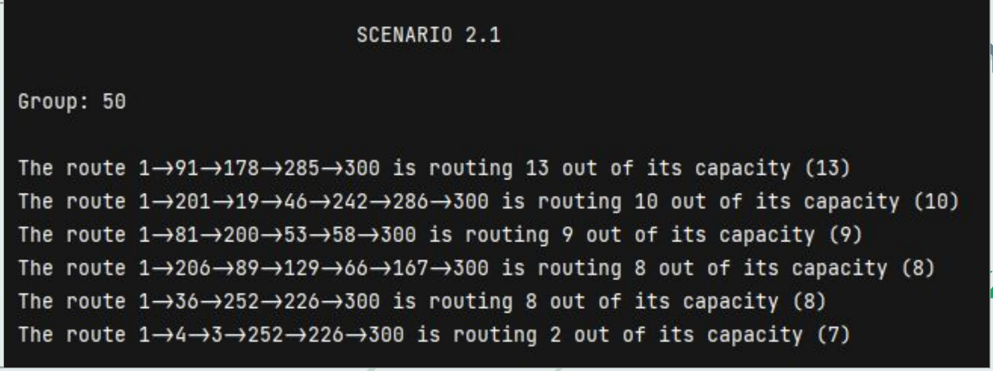


```
                          SCENARIO 2.1

Group: 9

The route 1→3→4 is routing 5 out of its capacity (5)
The route 1→2→4 is routing 3 out of its capacity (3)
The route 1→2→3→4 is routing 1 out of its capacity (2)
```

in3.txt



```
                          SCENARIO 2.1

Group: 50

The route 1→91→178→285→300 is routing 13 out of its capacity (13)
The route 1→201→19→46→242→286→300 is routing 10 out of its capacity (10)
The route 1→81→200→53→58→300 is routing 9 out of its capacity (9)
The route 1→206→89→129→66→167→300 is routing 8 out of its capacity (8)
The route 1→36→252→226→300 is routing 8 out of its capacity (8)
The route 1→4→3→252→226→300 is routing 2 out of its capacity (7)
```

Scenario 2_2

This scenario is considered as an add-on to the 2_1 scenario, since it adds the possibility to adjust a route so that a given number of people can be added to the group

Formalization of Scenario 2_2

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The previous group_size, **old_size;**
- The number of new elements, **new_elem;**
- Previous route, **r;**

Restrictions:

- The new group size (**old_size + new_elem**) cannot exceed the network maximum flow;

Objective:

- If needed, should correct route **r,** and output a new route, **r2**, so that the group's new size is **old_size + new_elem;**

Relevant Algorithms and Complexity

Algorithm Chosen:

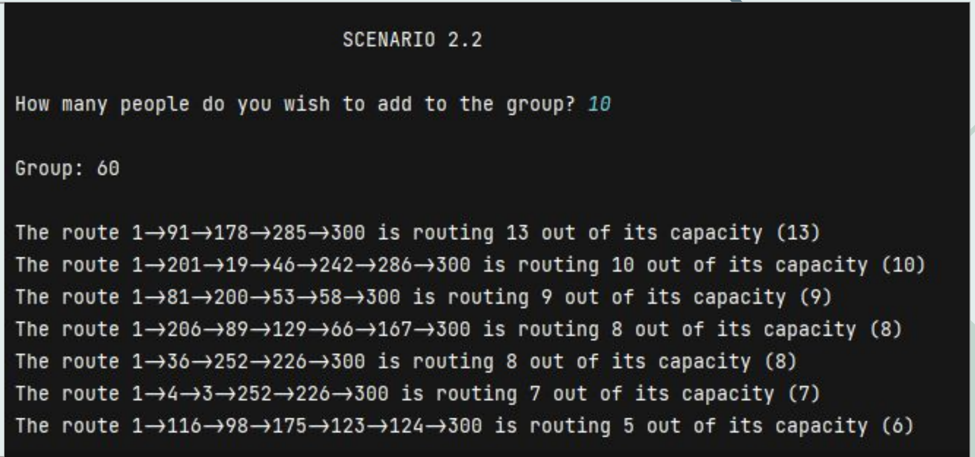
- In order to accomodate a given number of new people in the group, we may need to find a new routing, different than the one given by scenario 2_1:
  - For each route **r** in scenario's 2_1 solution
    - If **r**'s capacity is not full
      - Find how many of the new people we can include in the route, update **r**'s flow and decrement number of new people
        - **chunk** = MIN(**r.capacity** - **r.flow**, **new_people**)
        - **new_people -= chunk**
        - **r.flow += chunk**

Complexity:

- **O(n),** where n is the number of routes to iterate on

Results

in12.txt



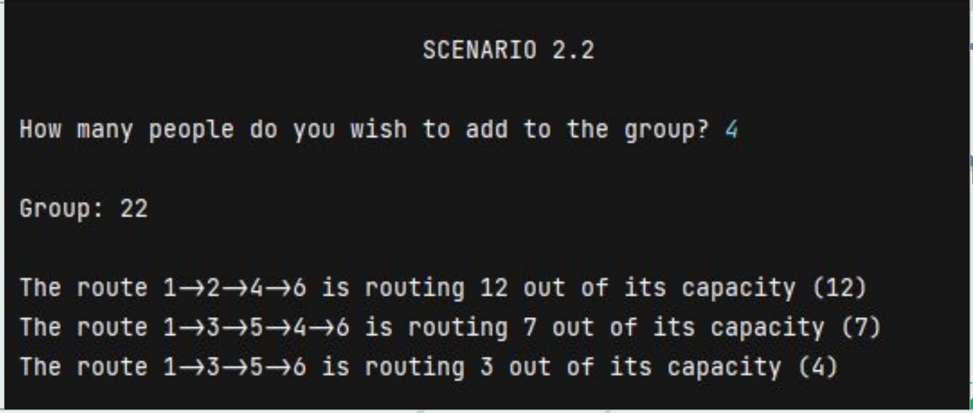
```
                    SCENARIO 2.2

How many people do you wish to add to the group? 10

Group: 60

The route 1→91→178→285→300 is routing 13 out of its capacity (13)
The route 1→201→19→46→242→286→300 is routing 10 out of its capacity (10)
The route 1→81→200→53→58→300 is routing 9 out of its capacity (9)
The route 1→206→89→129→66→167→300 is routing 8 out of its capacity (8)
The route 1→36→252→226→300 is routing 8 out of its capacity (8)
The route 1→4→3→252→226→300 is routing 7 out of its capacity (7)
The route 1→116→98→175→123→124→300 is routing 5 out of its capacity (6)
```

in11.txt



```
                    SCENARIO 2.2

How many people do you wish to add to the group? 4

Group: 22

The route 1→2→4→6 is routing 12 out of its capacity (12)
The route 1→3→5→4→6 is routing 7 out of its capacity (7)
The route 1→3→5→6 is routing 3 out of its capacity (4)
```

Scenario 2_3

This scenario should output the maximum group size and a route to get from point A to point B, both given.

This is the version of the scenario 1_1 in which the group can be separated into smaller groups.

Formalization of Scenario 2_3

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The maximum group size, **max_group_size**;
- The paths from **start** to **target**, **routes;**

Restrictions:

- Given a route, **R**, with the most capacity out of a list of routes between **start** and **target**, **max_group_size** cannot be greater than R's capacity.

Objective:

- Output the **maximum size** of the group and a possible routing from **start** to **target**.

Relevant Algorithms and Complexity

Algorithm Chosen:

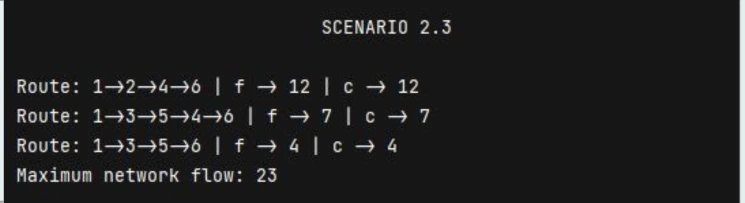
- Ford-Fulkerson algorithm

Complexity:

- **O(max_flow * E)**, where **E** is the number of edges in the graph, **g**.

Results

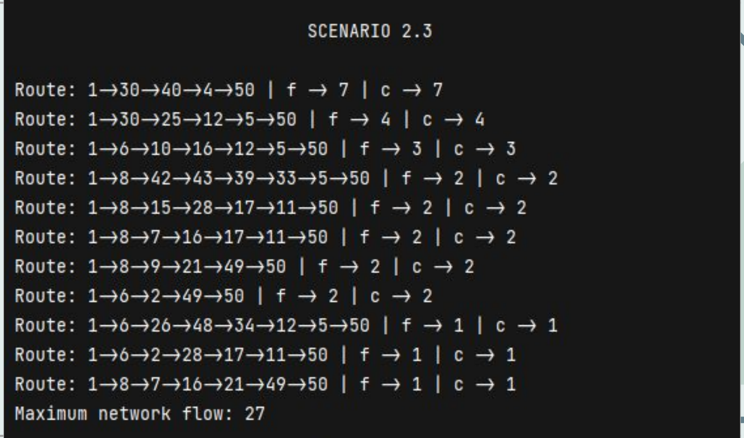
in12.txt



```
SCENARIO 2.3

Route: 1→2→4→6 | f → 12 | c → 12
Route: 1→3→5→4→6 | f → 7 | c → 7
Route: 1→3→5→6 | f → 4 | c → 4
Maximum network flow: 23
```

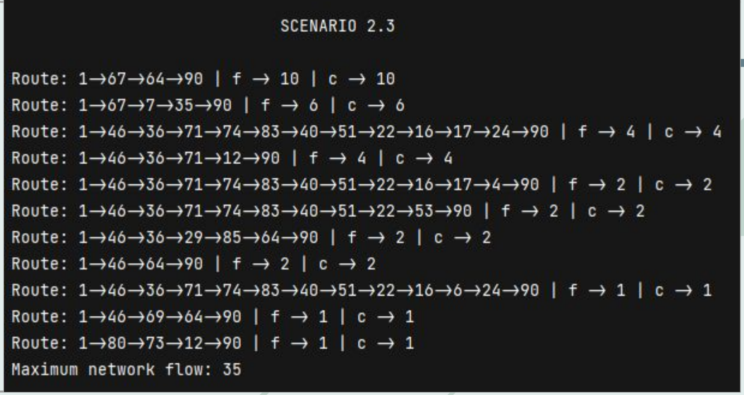
in2.txt



```
SCENARIO 2.3

Route: 1→30→40→4→50 | f → 7 | c → 7
Route: 1→30→25→12→5→50 | f → 4 | c → 4
Route: 1→6→10→16→12→5→50 | f → 3 | c → 3
Route: 1→8→42→43→39→33→5→50 | f → 2 | c → 2
Route: 1→8→15→28→17→11→50 | f → 2 | c → 2
Route: 1→8→7→16→17→11→50 | f → 2 | c → 2
Route: 1→8→9→21→49→50 | f → 2 | c → 2
Route: 1→6→2→49→50 | f → 2 | c → 2
Route: 1→6→26→48→34→12→5→50 | f → 1 | c → 1
Route: 1→6→2→28→17→11→50 | f → 1 | c → 1
Route: 1→8→7→16→21→49→50 | f → 1 | c → 1
Maximum network flow: 27
```

in7.txt



```
SCENARIO 2.3

Route: 1→67→64→90 | f → 10 | c → 10
Route: 1→67→7→35→90 | f → 6 | c → 6
Route: 1→46→36→71→74→83→40→51→22→16→17→24→90 | f → 4 | c → 4
Route: 1→46→36→71→12→90 | f → 4 | c → 4
Route: 1→46→36→71→74→83→40→51→22→16→17→4→90 | f → 2 | c → 2
Route: 1→46→36→71→74→83→40→51→22→53→90 | f → 2 | c → 2
Route: 1→46→36→29→85→64→90 | f → 2 | c → 2
Route: 1→46→64→90 | f → 2 | c → 2
Route: 1→46→36→71→74→83→40→51→22→16→6→24→90 | f → 1 | c → 1
Route: 1→46→69→64→90 | f → 1 | c → 1
Route: 1→80→73→12→90 | f → 1 | c → 1
Maximum network flow: 35
```

Scenario 2_4

Scenario that, given a route that makes up an acyclic graph, should identify when the group is united again, at the destination.

Formalization of Scenario 2_4

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The group's size, **size;**
- The previous route **r;**

Objective:

- Find out when the group will be united again, at **target**.

Relevant Algorithms and Complexity

Steps:

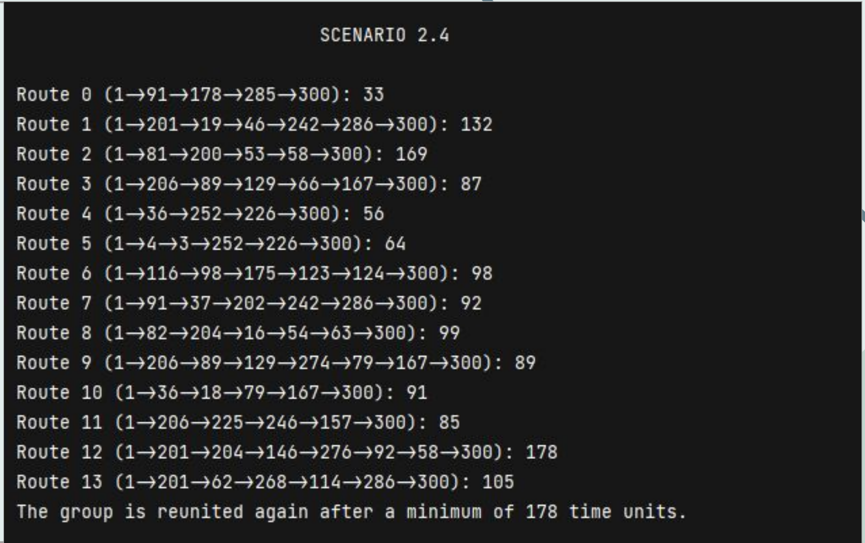
- Iterate through all all routes and, for each one, calculate the time it takes to transverse() it.

Complexity:

- **O(nr*n)**, where **nr** is the size of the routes list and **n** is the size of the adjacency list of each node.

Results

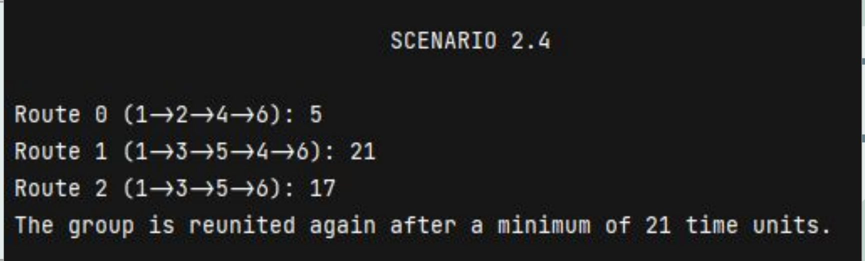
in3.txt



```
SCENARIO 2.4

Route 0 (1→91→178→285→300): 33
Route 1 (1→201→19→46→242→286→300): 132
Route 2 (1→81→200→53→58→300): 169
Route 3 (1→206→89→129→66→167→300): 87
Route 4 (1→36→252→226→300): 56
Route 5 (1→4→3→252→226→300): 64
Route 6 (1→116→98→175→123→124→300): 98
Route 7 (1→91→37→202→242→286→300): 92
Route 8 (1→82→204→16→54→63→300): 99
Route 9 (1→206→89→129→274→79→167→300): 89
Route 10 (1→36→18→79→167→300): 91
Route 11 (1→206→225→246→157→300): 85
Route 12 (1→201→204→146→276→92→58→300): 178
Route 13 (1→201→62→268→114→286→300): 105
The group is reunited again after a minimum of 178 time units.
```

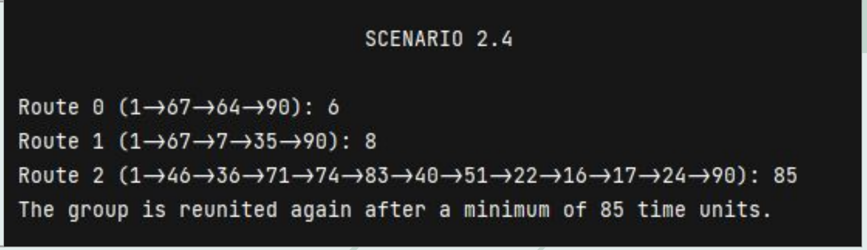
in12.txt



```
SCENARIO 2.4

Route 0 (1→2→4→6): 5
Route 1 (1→3→5→4→6): 21
Route 2 (1→3→5→6): 17
The group is reunited again after a minimum of 21 time units.
```

in9.txt



```
SCENARIO 2.4

Route 0 (1→67→64→90): 6
Route 1 (1→67→7→35→90): 8
Route 2 (1→46→36→71→74→83→40→51→22→16→17→24→90): 85
The group is reunited again after a minimum of 85 time units.
```

Scenario 2_5

Extension of the previous scenario that works out the maximum waiting time and the nodes(places) where elements of the group would have to wait.

Formalization of Scenario 2_5

Decision variables:

- The origin node, **start**;
- The destination node, **target**;
- The group's size, **size;**
- The previous route **r;**
- Maximum waiting time, **maxW;**
- Nodes in which group elements will wait **maxW** time, **wNodes;**

Objective:

- Find **maxW** and **wNodes**;

Relevant Algorithms and Complexity

Algorithm Chosen:

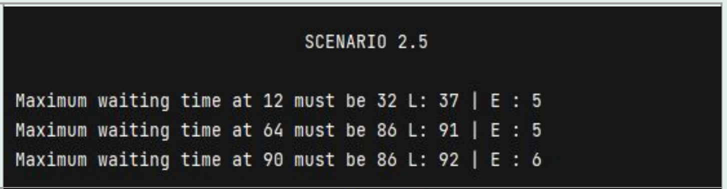
- Iterate through each route's nodes' adjacency list
    - Add the current edge duration to the sum of the previous edges durations
    - Set latest arrival time to the current path duration if bigger.
    - Set earliest arrival time to the current path duration if smaller.
- Print the waiting times on each node, if said node is used in any route and if earliest arrival and latest arrival at said node are not equal.

Complexity:

- **O(n^2)**

Results

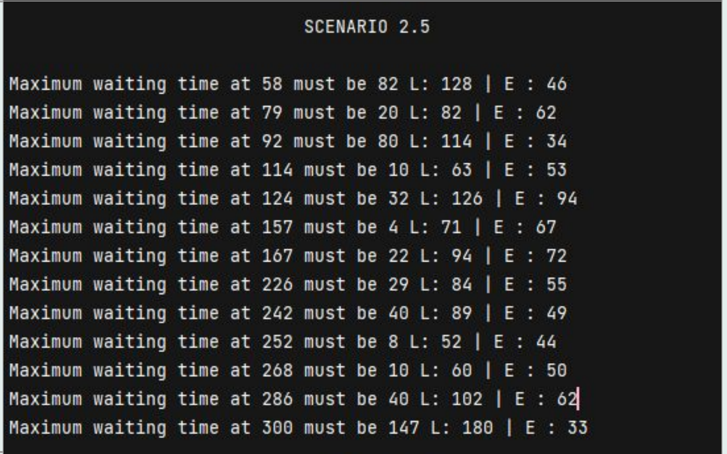
in7.txt



```
                    SCENARIO 2.5

Maximum waiting time at 12 must be 32 L: 37 | E : 5
Maximum waiting time at 64 must be 86 L: 91 | E : 5
Maximum waiting time at 90 must be 86 L: 92 | E : 6
```

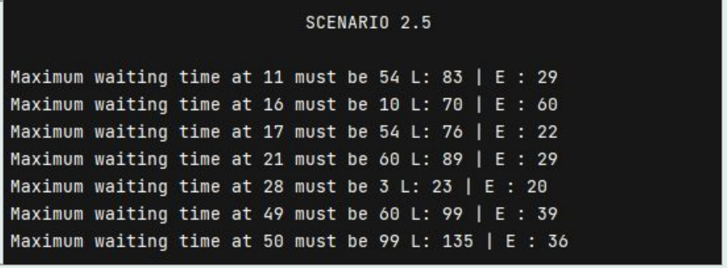
in3.txt



```
                    SCENARIO 2.5

Maximum waiting time at 58 must be 82 L: 128 | E : 46
Maximum waiting time at 79 must be 20 L: 82 | E : 62
Maximum waiting time at 92 must be 80 L: 114 | E : 34
Maximum waiting time at 114 must be 10 L: 63 | E : 53
Maximum waiting time at 124 must be 32 L: 126 | E : 94
Maximum waiting time at 157 must be 4 L: 71 | E : 67
Maximum waiting time at 167 must be 22 L: 94 | E : 72
Maximum waiting time at 226 must be 29 L: 84 | E : 55
Maximum waiting time at 242 must be 40 L: 89 | E : 49
Maximum waiting time at 252 must be 8 L: 52 | E : 44
Maximum waiting time at 268 must be 10 L: 60 | E : 50
Maximum waiting time at 286 must be 40 L: 102 | E : 62
Maximum waiting time at 300 must be 147 L: 180 | E : 33
```

in2.txt



```
                    SCENARIO 2.5

Maximum waiting time at 11 must be 54 L: 83 | E : 29
Maximum waiting time at 16 must be 10 L: 70 | E : 60
Maximum waiting time at 17 must be 54 L: 76 | E : 22
Maximum waiting time at 21 must be 60 L: 89 | E : 29
Maximum waiting time at 28 must be 3 L: 23 | E : 20
Maximum waiting time at 49 must be 60 L: 99 | E : 39
Maximum waiting time at 50 must be 99 L: 135 | E : 36
```

Conclusions

Due to the resources available on the syllabus and the contents addressed on the theoretical classes, we believe that we were able to work through all of the scenarios and achieving good results.

Group self-assessment

We use a scale from 0 to 5 to make a self-assessment of the group, where 5 is the one that worked best in the group

- Edgar Lourenço - 4/5
- Carlos Verissimo - 4.5/5
- Nuno Jesus - 5/5