DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

CSE2310 Algorithm Design
Digital test – part 2

January 28, 2020, 120 minutes

- Usage of the book, notes or a calculator during this test is not allowed.
- This digital test contains 4 questions (worth a total of 10 points) contributing 10/20 to your grade for the digital test. The other digital test contributes the other 10/20 to your grade. Note that the final mark for this course also consists for 40% of the unrounded score for the written exam (if both ≥ 5.0).
- Provide your answers by submitting an implementation solution for the respective assignment on WebLab.
- When an implementation is asked, please provide the **most efficient** implementation in terms of asymptotic time complexity. Providing a suboptimal implementation can lead to a subtraction of points.
- The spec tests in WebLab **do not represent your grade**.
- There is an API specification of the Java Platform (Standard Edition 8) available at `https://weblab.tudelft.nl/docs/java/8`.
- Although we cannot help you with the assignments (obviously), if you have clarification questions about the exam (during the exam), please use the `Discussion` functionality of WebLab.
- The material for this tests consists of module 3 (Dynamic Programming) and 4 (Network Flow) of the course and the corresponding book chapters and lectures.
- Total number of pages (without this preamble): 4.
- Exam prepared by M. de Weerdt and S. Hugtenburg

Learning goals coverage of the *digital test* – part 2, based on the learning objectives on BS:

Goal	2018–2019	resit	2019–2020
Memoization			
Weighted interval scheduling		1	
Segmented least squares	1		
Subset-sum		2	
Knapsack	2		
RNA			1
Sequence alignment			
Space-efficient alignment			
Bellman-Ford shortest paths			2
A new DP algorithm	1		2
Flows, residual graphs, augmenting paths			
Cuts, capacity of a cut			3
Demand and circulation with lower bounds		3	
(Scaling-)Ford-Fulkerson			
Maximum bipartite matching		3	4
Maximum Edge-Disjoint paths			
Survey Design			4
Image Segmentation			
Project Selection	3		3
Baseball elimination			
Max-flow min-cut	3		
New network flow model	3		4

In general, the digital test is about designing, implementing, applying and analyzing a new algorithm that is similar to one or more of the algorithms listed above. Note that the exams in 2018–2019 took 90 minutes, later exams have 120 minutes.

1. (2 points) The problem of understanding the two-dimensional structure of a RNA-molecule (RNA Secondary Structure) is defined as follows: given a sequence of bases $B = b_1, \ldots, b_n$ with $b_i \in \{A, U, C, G\}$, find a set S of base pairs (i, j) that is as large as possible and meets some criteria. If a base pair (i, j) meets these criteria, we write $p(i, j)$. (This function is given in the form of a compatibility matrix.) The following function then describes the size of the largest S set of base pairs that meet these criteria.

$$\mathrm{OPT}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 0 & \text{if } i > j \\ \max\left\{\mathrm{OPT}(i, j-1), \max\limits_{\{t \mid i \le t < j-4 \wedge p(t,j)\}} \{1 + \mathrm{OPT}(i, t-1) + \mathrm{OPT}(t+1, j-1)\}\right\} & \text{else} \end{cases}$$

Implement an **iterative** dynamic programming algorithm in WebLab such that it returns $\mathrm{OPT}(1, n) = |S|$.

Solution: This is equivalent to the RNA problem treated in the course. The most important is to decide in which order to deal with the subproblems. There are at least two valid orders:

```
for k ← 1 (or 5) to n − 1 do
    for i ← 1 to n − k do
        compute OPT(i, i + k)
    end for
end for
```

or

```
for j ← 1 to n do
    for i ← 1 to j − 1 do
        compute OPT(i, j)
    end for
end for
```

See WebLab for an exact implementation.

2. (3 points) You are working on a decision-support algorithm to plan research activities. There appear to be a great many ways to achieve the main goal of the research, the "holy grail" (g).

   The possible ways towards g are represented by a directed graph, where the states of knowledge (including the current state 1 as well as the state representing g) are represented by a set of n nodes V, and a set E of m edges represents research activities that can be executed to get from one state to another. Each research activity represented by $e \in E$ also carries some known costs c_e. (Although typically this is a positive number, it may happen that the activity in fact brings some profit. Unfortunately, there are no negative cycles.) Finding the cheapest way to get to g looks like a very familiar problem, but it is possible to invite collaborators on activities, reducing its costs by 50%. The management of the research group has decided to do this at most once, to be able to claim reaching g without much help.

   Implement an **iterative** algorithm that returns the minimum cost of a research plan to reach g, where the plan includes at most one activity using collaborators.

Solution:

The following function describes the minimum cost of a research plan using at most i steps to get to state s and using at most c collaborations.

$$\mathrm{OPT}(s,i,c) = \begin{cases} 0 & \text{if } s = 1 \\ \infty & \text{if } i = 0 \\ \min_{e \in \{(s',s) \in E\}} (c_e + \mathrm{OPT}(s', i-1, c)) & \text{if } c = 0 \\ \min_{e \in \{(s',s) \in E\}} \min\{c_e + \mathrm{OPT}(s', i-1, c), 0.5 \cdot c_e + \mathrm{OPT}(s', i-1, c-1)\} & \text{otherwise} \end{cases}$$

Like with Bellman-Ford's shortest path algorithm, this can be implemented using only an array of size $n \cdot c$. The answer can be found by computing $\mathrm{OPT}(g, n, 1)$. For an iterative implementation please refer to WebLab.

3. (1½ points) Consider the project selection problem, where we have a set of n projects each with a cost c_i and a benefit b_i. Certain projects require other projects to be completed. Given these costs, benefits and requirements, we model the problem as such:
   1. Create a source s and a sink t, and a vertex p_i for every project.
   2. Connect s to every p_i with capacity b_i and every p_i to the sink with capacity c_i.
   3. Connect p_i to p_j with capacity ∞ if project i requires project j.

   We already provide you with implementation to construct the graph and find the maximum flow according to the steps above. You should finish the methods `outputSelection` so that it outputs a selection of the projects that maximizes our profit (benefit - costs for our selected projects).

Solution: The code should return all ids of nodes that can be reached in the residual graph. This can be implemented with a best-first search:

```
Q.add( s)
S ← ∅
while ! Q.isEmpty()  do
    u ← Q.pop()
    for each neighbor v of c do
        if  v ∉ S and (u, v) has remaining capacity then
            add v to S
            Q.add( v)
        end if
    end for
end whilereturn S
```

4. (3½ points) In the third year of the bachelor, all students are going to do a research project. Since there are many more students than potential supervisors, each supervisor supervises students in groups. A group always meets on Mondays or always on Tuesdays (during Q4). A supervisor can supervise multiple groups (but at most two per day). Students are allocated to groups according to the following rules.
   - Supervisors and students need to be available at least on Monday or Tuesday (or both).
   - Each supervisor supervises at least 3 and at most 12 students.
   - Group meetings can have at most 5 students.

   Given a set of students, their availability and their selected supervisors, and the availability of a set of supervisors, find out whether a valid allocation exists by modeling this problem as a flow network.

For this assignment you are advised to use the code made available on WebLab for this assignment specifically. The `Graph`, `Node`, `Edge` and `MaxFlow` classes available to you are based on the extended network flow library created in the last tutorial of the course. You are free to make modifications to these classes, though our reference solutions do not modify them in any way. Functions you may require include:

- In the `Node` class:
  - `public Node(int id, int d)` to construct a new Node object with id `id` and a demand of `d`. Please note that a supply is represented by a negative demand.
  - `public void addEdge(Node to, int lower, int upper)` to construct a new edge between `this` and `to` with lower bound `lower` and upper bound `upper`.
- In the `Graph` class:
  - `public Graph(List<Node> nodes)` to construct a new graph with the nodes `nodes`.
  - `public boolean hasCirculation()` to determine if the graph has a circulation of the value D where D is the sum of the demands in the graph. This method will throw an `IllegalArgumentException` if the amount of supply in the network is not equal to the amount of demand.

Solution: For our reference implementation, please refer to Weblab.

- Add a node s with supply n and a node t with demand n (supply can also be spread over the student nodes)
- Add a node for every student x and connect from s with capacity 1 (and connect from s with capacity 1 if you have used s)
- Add a node for every supervisor y and connect y to t with capacity 12 and lower bound 3
- Add a node $x - Mon$ for every student available on Monday and a node $x - Tue$ for every student available on Tuesday, and connect x to their respective nodes with capacity at least 1, or connect a student available on Monday (or Tuesday) to the respective $y - Mon$ (or $y - Tue$)
- Add two nodes $y - Mon$ for every supervisor who is available on Mondays, add two nodes $y - Tue$ for every supervisor who is available on Tuesdays, and connect x to their respective nodes with capacity 5
- Connect (nodes related to) student x only with nodes related to selected supervisors y
- Return true if a circulation exists, so does a feasible allocation

End of exam questions