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

TI2306 Algorithm Design
Resit Digital test – part 2

April 18, 2019, 90 minutes

- Usage of the book, notes or a calculator during this test is not allowed.
- This digital test contains 3 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 $\frac{2}{3}$ of the unrounded score for the written exam (if both ≥ 5.0).
- The storyline throughout the questions can be ignored: each question can be answered independently of the others and in any order.
- 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/java8/api`.
- 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 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
Memoization		
Weighted interval scheduling		1
Segmented least squares	1	
Subset-sum		2
Knapsack	2	
RNA		
Sequence alignment		
Space-efficient alignment		
All-pairs shortest path		
A new DP algorithm	1	
Flows, residual graphs, augmenting paths		
Cuts, capacity of a cut		
Demand and circulation with lower bounds		3
(Scaling-)Ford-Fulkerson		
Maximum bipartite matching		3
Maximum Edge-Disjoint paths		
Survey Design		
Image Segmentation		
Project Selection	3	
Baseball elimination		
Max-flow min-cut	3	
New network flow model	3	

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.

You are participating in the famous British television programme: The Antiques Roadshow. Your goal is to collect a set of items with the largest maximal value on your way to an auction. However every item is only available for pick-up if you race there and ignore some of the items immediately before it. This means that if you want to collect item i you must race past $i - k_i$ until $i - 1$. The question you now ask yourself is: what items should I collect?

You are given a sequence of n items each with a value v_i. Furthermore, each item i is incompatible with 0 or more items immediately (and consecutively) before i in this sequence. For each item $1 \leq i \leq n$ this number is given by skip$[i]$ (the number k_i in the example text above), and we know that $0 \leq \text{skip}[i] < i$ for all i. Further we define the value of a set of items as the sum of values of the items in the set.

1. (3 points) Implement an iterative dynamic programming algorithm that finds the subset of compatible items with the *maximum* value and returns this value.

   Solution: This is equivalent to a maximum weighted intervals problem (where the items are jobs/intervals that may not overlap).

   $$OPT(i) = \begin{cases} 0 & \text{if } i \leq 0 \\ \max\{v_i + OPT(i - 1 - \text{skip}(i)), OPT(i-1)\} & \text{otherwise} \end{cases}$$

   The value $OPT(n)$ then returns the maximum value over all compatible subsets.

   So a one-dimensional array can be used. The value $OPT(i)$ uses $OPT(i')$ with $0 \leq i' \leq i - 1$, so the algorithm should compute the values in increasing i.

   $M \leftarrow \text{Array}[0, \ldots, n]$
   $M[0] \leftarrow 0$
   for $i = 1, \ldots, m$ **do**
   $M[j] \leftarrow \max\{v_i + M[i - 1 - \text{skip}(i)], M[i-1]\}$
   end for
   return $M[n]$

   Rubric:

   - There is an array that is being used to store smaller subproblems that are re-used: 1.0
   - The size of the array is $\Theta(n)$: 0.5
   - The maximum is taken over the two correct alternatives: 1.0
   - Return M[n] from the array: 0.5
   - -1 point for a recursive implementation *with* memoization.

2. (2 points) The director of the The Antiques Roadshow has decided to introduce a new mechanic. At an auction you are only allowed to sell items with a total value of at most V. Since you have already bought your items and are already at the auction, you now need to decide which of the items you will sell so that you are closest to that value V. To that end, you implement a dynamic programming solution to determine what value $V' \leq V$ you can reach with your items.

   This program fills a 2D-array `M` with the following values, so that the total amount we can put up for auction is found by computing $V' = V - \texttt{M}(n, V)$.

   $$\texttt{M}(i, v) = \begin{cases} v & \text{if } i = 0 \\ 0 & \text{if } v = 0 \\ \texttt{M}(i-1, v) & \text{if } v_i > v \\ \min\{\texttt{M}(i-1, v - v_i), \texttt{M}(i-1, v)\} & \text{otherwise} \end{cases}$$

   The auction is about to begin however, and although you now know what value you can reach, you now need to determine what items to actually put up for auction.

Given a sequence of items with values v_i, a maximum value V and a 2D-array `M` of size $(n+1) \times (V+1)$, return a list of ids of the items that we should put up for auction, sorted *descendingly by id*.

Solution:

$i \leftarrow n$; $v \leftarrow V$
while $i > 0$ **do**
  if $v_i > v$ or $M[i-1][v-v_i] > M[i-1][v]$ **then**
    Skip it
  else
    print i
    $v \leftarrow v - v_i$
  end if
  $i \leftarrow i - 1$
end while

The condition can also be inversed and then looks as follows: $v_i \leq v$ and $M[i-1][v-v_i] \leq M[i-1][v]$ (where the last can also be $<$)

Rubric:

- Correct initialisation of i and v: 0.5
- Correct if statement and action: 1
- Correct order of the jobs: 0.5

3. (5 points) A number of students register with DelfFlext to earn some money by helping out with study-related tasks. A contract between a student and DelfFlext contains a minimum and maximum number of hours per month, as well as a list of tasks the student is able to do (taking into account required background knowledge, time constraints, and the student's own selection). Each task requires a certain minimum number of student (assistant) hours per month. (We suppose there is no difference in allocating one student for two hours or two students for one hour.)

   Every month DelfFlext aims to allocate students to the tasks for that month such that all requirements above are met. You are asked to help out with this allocation by writing a program that outputs true if such an allocation is possible, and false if it is not.

   Model this problem as a circulation with lower bounds, such that a valid circulation exists if and only if a feasible allocation exists.

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 those you have seen in the lab work of the course. You are free to make modifications to these classes, though our reference solution does not modify them in any way. We have included the notion of supply/demand and lower bounds in this implementation already. 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 hasCirculationOfAtLeast(int y)` to determine if the graph has a circulation of at least value `y`.

Solution:

Solution, including rubric:

- Add a node for every student x. (0.5 point)

- Add a node for every task y. (0.5 point)
- Define a source s and an edge from s to x with its minimum hours (c_x^-) as lower bound and maximum hours (c_x^+) as capacity. (1 point)
- Let the task nodes have a demand equal to the minimum number of hours required (min_y). (1 point)
- Let s have the total demand as supply. (1 point)
- Add an edge from every student x to task y where x has y on his/her list, with either infinite capacity or (at least) the number of hours of that student or of that task. (1 point)
- Return true iff a circulation (with value total min hours) exists, so does a feasible allocation: 0.5 point

End of exam questions