

Home work for  
**Fundamental Algorithms**  
SS 2007  
Sheet 3

**Exercise 6:** Consider the following

**Problem KNAPSACK:**

**in:**  $n \in \mathbb{N}$ ,  $u \in \mathbb{N}^n$ ,  $w \in \mathbb{N}^n$ ,  $c \in \mathbb{N}$ .

**out:**  $I \subseteq [n]$ , such that  $\sum_{i \in I} w_i \leq c$  and  $\sum_{i \in I} u_i$  is maximum.

Formulate problem KNAPSACK as a shortest path problem in a suitable DAG  $G$ . In case that  $c$  is polynomial in  $n$  use a shortest path algorithm to solve KNAPSACK in polynomial time.

**Exercise 7:** Let  $G = (V, E)$  be a directed graph where the nodes correspond to computers and the edges correspond to communication links. For each edge  $(u, v) \in E$  a reliability value  $r(u, v) \in [0, 1]$  is given which is the probability that the communication link between  $u$  and  $v$  does not fail. We assume that these probabilities are independent.

Give an efficient algorithm to find the most reliable path from one given computer to another given one, where the reliability of a path  $P$  is defined to be the product of the reliabilities of the edges on the path. Prove the correctness of your algorithm.

**Exercise 8:** (Programming task: to be returned by email until May 11th 1:00 pm)

Write a program that

- a) reads a directed graph  $G$  from an input file, with the input specification as given below,
- b) represents the graph internally by adjacency lists,
- c) solves the Single Source Shortest Path problem with root  $s$  using the Dijkstra algorithm, if possible, or the Bellman-Ford algorithm.
- d) outputs the result according to the output specification given below.

To complete this task you must send by email to obelix@upb.de

- 1) The source code of your program
- 2) The command to compile your program (this must be possible on computer mistel.cs.upb.de (Linux system))
- 3) The command to start your program. The input file name must be given as the first argument, the source  $s \in V$  of the shortest paths as the second argument.

Input specification:

```
<comment line>*  
n = <int>  
m = <int>  
<int> : <list of edges>  
:  
<int> : <list of edges>
```

where <comment line> is a line starting with #, and <list of edges> is a list of edges separated by a blank (the adjacency lists), and an edge is given as <int>w<int>, where the first integer is the target of the edge, and the second integer is the weight of the edge. Weights can be assumed to be from the set  $\{-9999, \dots, 9999\}$ . An example is given below:

```
# This is a graph with 3 nodes, 5 edges  
n = 3  
m = 5  
0 : 1w11 2w-3  
1 : 2w6  
2 : 1w-5 0w9
```

Output specification:

Output should consist of exactly one line of the following form:

If  $G$  does not contain negative weight cycles reachable from  $s$ :  
delta <int> : <list of edges> where the first integer represents the root, and the edges after the “:” give the target and its distance from the root.

If  $G$  contains negative weight cycles reachable from  $s$ :  
delta <int> : negative-cycles

An example for source  $s = 0$  is given below:

```
delta 0 : 0w0 1w-8 2w-3
```

Another one may look like

```
delta 178 : negative-cycles
```