T-79.4202 **Autumn 2013**
Principles of Algorithmic Techniques
Programming Tutorial 3, December 3–4
Problems

For both of the problem sets 1 and 2 described below, design, implement and test a program for *one* of the alternatives. You may use any programming language of your choice, but please observe that your programs must meet the input/output interface requirements given on the last two pages. Valid input/output pairs for testing your programs are available in Noppa.

The **deadline** for submitting your solutions is Monday 2, 23:59 p.m. In order to earn credit, you must also attend the respective tutorial session and be ready to explain and demonstrate your solutions. For further details and instructions, see `https://noppa.aalto.fi/noppa/kurssi/t-79.4202/harjoitustyot`.

1. **Greedy algorithms:**

   (a) Minimal vertex cover.

   Let (V,E) be a graph. A subset of vertices $S \subseteq V$ is a *vertex cover* if each $e \in E$ has at least one of its endpoints in S.

   Trivially, the full vertex set V is a vertex cover. An interesting (and NP-hard) problem is to find a *minimum* vertex cover, i.e. one that has as few vertices as possible. Let us relax this target a little: call a vertex cover S *minimal* if no proper subset of S is a vertex cover. Note that a minimum vertex cover is always minimal, but the opposite does not hold (can you see why?).

   Implement a greedy algorithm that finds a minimal vertex cover.

   (b) Bottleneck spanning trees.

   Trucko Inc. is beginning to operate on a hilly island whose road network contains a lot of tunnels and bridges. Thus a majority of the roads have quite restrictive height limits. The company is able to operate only one type of vehicle, which must be low enough to reach every city. The company is fine with using detours and not taking the shortest path as long as the height limits are met on some path between each pair of cities. This given, the company wants the truck to be as high as possible to maximize cargo space.

   Find the optimum vehicle height, and a corresponding spanning tree on the given road network.

   In other words: find a spanning tree whose bottleneck edge (that is, an edge with smallest value in the tree) has largest value among all spanning trees.

2. **Dynamic programming:**

   (a) Optimizing fuel station locations. (Essentially 6.3. in Dasgupta et al.)

   Brownish Petroleum Co. is considering opening a series of fuel stations along a national highway. The n possible locations lie along a straight line, and the distance of each location j from the start of the highway is m_j miles, satisfying $0 \leq m_1 < m_2 < \cdots < m_n$.

   The constraints are as follows. At each location, at most one fuel station can be opened. Any two opened stations must be at least k miles apart, where k is a positive integer. The expected profit from opening a station at location m_i is p_i, where $p_i > 0$ and $1 \leq i \leq n$.

   Implement an efficient dynamic-programming algorithm to maximise total expected profit subject to the given constraints.

   (b) Constructing change from coin denominations. (Essentially 6.17. in Dasgupta et al.)

   Given an unlimited supply of coins of denominations $x_1 < x_2 < \cdots < x_n$, we wish to make change for a value v. That is, we wish to find a set of coins whose total value is v. This might not be possible: for instance, if the denominations are 5 and 10 then we can make change for 15 but not for 12.

   Give an $O(nv)$ dynamic-programming algorithm for finding a set of allowed coins that has total value v.

Input/output interface

Input 1(a)

A graph with n vertices and m edges, in DIMACS format. You may assume that $n < 500$. The DIMACS format stars with a line

`p edge` n m

and an edge from vertex u to v is represented by the line

`e` u v

Output 1(a)

As the last line of output, print the string "`Cover:`" and elements of your vertex cover S. For vertices 9, 23, 3 and 7 one would print:

```
Cover: 9 23 3 7
```

Furthermore, your program should print some intermediate results of your choice to help you demonstrate how your program works, but please note that the intermediate results should not use the string "`Cover:`" used in the final result line.

Input 1(b)

A graph representing height limits (integers in centimeters) between cities. The graph is in the DIMACS format, with an additional integer after each edge representing the height limit on this edge. Thus the edge lines are of the form

`e` u v h

where h means height limit on edge from u to v. Any nonexistent edge is interpreted as having height limit 0.

Output 1(b)

Give each spanning tree edge on its own line, in the format "*u v value*". As the last line, give "`bottleneck:`" and the bottleneck edge value (that is, the height of vehicle). Example:

```
1  2  220
2  3  400
1  4  333
bottleneck: 220
```

Input 2(a)

The input file contains $2+n$ integers followed by n double floats, divided into three lines as follows:

$$\begin{array}{cccc} k & n & & \\ m_1 & m_2 & \ldots & m_n \\ p_1 & p_2 & \ldots & p_n \end{array}$$

Output 2(a)

You are assumed to solve the problem using dynamic programming, so the program should be solving some kind of smaller subproblems one after another. Whenever your algorithm solves a (previously unsolved) subproblem optimally, print two lines as in this example format:

```
#180 profit: 320.24
placements: 1 44 86 143
```

This corresponds to solution of subproblem 180 having profit 320.34 when one opens locations 1, 44, 86 and 143. Report the locations in increasing order.

Input 2(b)

The input file contains $2+n$ integers: the target value v, number of denominations n, and the coin denominations, divided into two lines as follows:

$$\begin{array}{cccc} v & n & & \\ x_1 & x_2 & \ldots & x_n \end{array}$$

You may assume that $n < 40$ and $v < 50000$.

Output 2(b)

Print a way of decomposing the target value v into coin denominations in the form

$$\text{[target]} = \text{[multiplicity]}*\text{[coin denomination]} + \text{[multiplicity]}*\text{[coin denomination]} + \ldots$$

where the coin denominations are reported in order from the largest to the smallest denomination. Coin denominations with zero multiplicity should not be reported. For example, if one is using three coins of value 4 and four coins of value 5 (and no coins of other values) to construct change of 32, print the line:

```
32 = 4*5 + 3*4
```

If it is not possible to make change for the desired value 32 with the given coin denominations, print a message indicating this:

```
Cannot form change of 32.
```