Assignment 9 **Valeria Fionda, Mouna Kacimi, Werner Nutt, Simon Razniewski**

Hashing and Dynamic Programming

Instructions: Your assignment should represent your own effort. However, you are not expected to work alone. It is fine to discuss the exercises and try to find solutions together, but each student shall write down and submit his/her solutions separately. It is good academic standard to acknowledge collaborators, so if you worked together with other students, please list their names.
Be prepared to present your solution at the lab. If you are not able to explain your solution, this will be considered as if you had not done your work at all.
You can write up your answers by hand (provided your handwriting is legible) or use a word processing system like Latex or Word. Experience shows that Word is in general difficult to use for this kind of task.
For a programming task, your solution must contain (i) an explanation of your solution to the problem, (ii) the Java code, in a form that we can run it, (iii) instructions how to run it. Also put the source code into your solution document. For all programming tasks, it is not allowed to use any external libraries ("import") or advanced built-in API functions (for example, `String.indexof("a")` or `String.substring(1,5)`), if not stated otherwise.
Please, include name, matriculation number and email address in your submission.

1. Open Addressing: Probing with a Step Width that is a Prime Number

Consider the problem of defining a hashing function $h(k, i)$ for open addressing on a table of length m, that is, with slots numbers $0, 1, 2, \ldots, m-1$. We have claimed in the lecture that a function $h(k, i) = h_1(k) + i \cdot h_2(k) \bmod m$ produces a permutation for every k if $h_2(k)$ and m are relatively prime, that is, if $\gcd(h_2(k), m) = 1$. This exercise is about proving the claim.

Let m, w be integers such that the greatest common divisor $\gcd(m, w) = 1$. Prove that the function

$$f\colon \{\, 0, ..., m-1 \,\} \to \{\, 0, ..., m-1 \,\}, \quad f(i) = i \cdot w \bmod m$$

is a permutation, in other words, a bijective function.

Hint: It suffices to show that f is injective, that is, $f(i) = f(j)$ implies $i = j$.

(5 Points)

2. Least Dangerous Climbing Route

Imagine a climber trying to climb on top of a wall. A wall is constructed out of square blocks of equal size, each of which provides one handhold. Some handholds are more dangerous than others. From each block, the climber can reach three blocks of the row right above: one right on top, one to the right and one to the left (unless right or left are not available because the block is the end of the wall). The goal is to find the least dangerous path from the bottom of the wall to the top where the danger rating of a path is the maximum of the danger ratings of blocks used on that path.

We represent the problem as follows. The input is an $n \times m$ grid in which each cell has a rating $R(i, j)$ associated with it. The bottom row is row 1, the top row is row n. From a cell (i, j) a climber can reach in one step cell $(i + 1, j - 1)$ (if $j > 1$), cell $(i + 1, j)$, and cell $(i + 1, j + 1)$ (if $j < m$).

Here is an example of an input grid:

```
   .-------------------.
 6 | 3 | 4 | 1 | 2 | 1 |
   |---+---+---+---+---|
 5 | 8 | 2 | 3 | 1 | 5 |
   |---+---+---+---+---|
 4 | 0 | 6 | 4 | 9 | 7 |
   |---+---+---+---+---|
 3 | 6 | 5 | 8 | 3 | 4 |
   |---+---+---+---+---|
 2 | 2 | 7 | 1 | 9 | 5 |
   |---+---+---+---+---|
 1 | 6 | 8 | 3 | 5 | 1 |
   '-------------------'

     1   2   3   4   5
```

Figure 1: Climbing Grid

For instance, on that 6×5 climbing grid, $R(1, 3) = 3$ and the climber can move to $(2, 2)$, $(2, 3)$, $(2, 4)$, each having a cost of 7, 1, 9, respectively. For a climbing path from the bottom to the top, the *danger rating* is the maximum of all values on the blocks that make up the path.

1. Write a recursive algorithm in Java that finds a path from bottom to top with minimal danger factor and prints it. Test the algorithm with the grid in Figure 1.

2. Write a memoized version of the recursive “best path” algorithm from Task 1 in Java. Compare the complexity of both algorithms.

Hint 1: First, solve the optimization problem. Then create an additional data structure that keeps track of the choices made when finding the optimum.

Hint 2: For the optimization problem, consider the more specific problem to find for a given cell in the top row the best path leading to that cell.

(10 Points)

3. Best Career Path

Consider a network with n nodes, numbered 1 to n, which are all connected one to the other. Making a step from node i to node j can either bear a cost, indicated by a negative value, or bring about a benefit, indicated by a positive value. Some steps have a cost (or benefit) of 0. In particular, staying at a node, that is, going from node i to node i, has neither a cost nor brings about a benefit. As an example, you can think of this as modeling possible professional careers. Changing from job i to job j may bring about an increase in salary or a cost, for instance, when studying instead of working.
We model such a network by an $n \times n$ matrix B, where the entry $B[i, j]$ indicates the benefit (or the cost, if negative), of a step from node i to node j. Staying in a position does not have any economic effect, thus $B[i, i] = 0$ for all $i = 1, ..., n$.
We want to find out what is the best path between any nodes i and j. A path p from i to j is a sequence of numbers $p = (i_0, i_1, ..., i_m)$ where $i = i_0$ and $j = i_m$. The number m indicates the number of steps on the path and is called the *length* of the path p. The gain of path p is the sum of the values of all steps, that is, $g(p) = \sum_{l=1}^{m} B[i_{l-1}, i_l]$. The maximal gain of a path from i to j is

$$G(i, j) := \max\{\, g(p) \mid p \text{ is a path from } i \text{ to } j \,\}.$$

We assume that it is not possible to gain from walking in a cycle. Otherwise, no maximum exists and the problem is not well-defined. This means, the values in B must be such that for any path p from a node i to itself, we have $g(p) \leq 0$. The following matrix has this property:

$$B = \begin{bmatrix} 0 & 1 & 0 & 1 \\ -2 & 0 & 0 & -2 \\ 0 & 2 & 0 & 1 \\ -3 & -1 & -3 & 0 \end{bmatrix}$$

You are asked to develop an algorithm that takes such a B as input and computes $G(i, j)$ for all i, j. In the following, we give you some hints.

Hint 1: Cycles bring no gain. Therefore, the greatest possible benefit of moving from i to j can be achieved by visiting any intermediate node at most once.

Hint 2: Consider the following variant of the problem. Let $G_{aux}(i, j, k)$ be the maximal gain that can be achieved by walking from i to j along a path that uses only the nodes $1, ..., k$ as intermediate points. Develop an algorithm for G_{aux}.

These are your tasks:

1. Explain how $G_{aux}(i, j, k)$ can be used to compute $G(i, j)$. Develop a recurrence for $G_{aux}(i, j, k)$.
2. Write pseudo-code for algorithms that compute arrays with all values of G_{aux} and G and explain why your algorithms are correct.
3. Extend the pseudo-code algorithm G_{aux} so that it computes an auxiliary data structure for printing an optimal path from i to j. Use that data structure to write an algorithm `PrintBestPath`(i, j) that prints out such a path.
4. Implement your algorithms in Java and run them on the test matrix B above.

Hint 3: To find the recurrence for G_{aux}, observe that the best path from i to j, visiting only nodes among $1, \dots, k$,

- either visits only nodes among $1, \dots, k-1$, or
- visits k exactly once, and therefore can be divided into a path from i to k and a path from k to j, where neither of the two paths visits k.

Hint 4: To print an optimal path from i to j, it is good to know for all i, j an intermediate point k such that walking from i to j across k brings the greatest gain.

(15 Points)

Submission: Thu, 24 May 2012, 8:30 am. Preferrably by email.

If you want to submit a hand-written solution, hand it over to your lab tutor or in the lecture. However, all code has to be submitted in electronic form.