Algorithms

Example Sheet 2

Petar Veličković

Lent Term 2017

Warm-up

1. Implement a recursive function that will compute F_n, the n-th Fibonacci number, both with and without *memoization*. Compare their running times for several values of n.

2. You're playing an '80s computer game on a map represented by an $N \times N$ matrix. You start in the upper left corner, and your objective is to reach the bottom right corner. At each step, you are only allowed to go right or down. Each cell contains some amount of coins—explain the algorithm you would use to determine the path that will maximise your number of coins. An example, with highlighted optimal path, is given below:

1	**3**	**5**	3	1
4	2	**5**	**3**	4
2	2	2	**4**	5
4	4	1	**5**	3
5	1	2	**3**	**1**

3. When would you approach a problem using a greedy strategy?

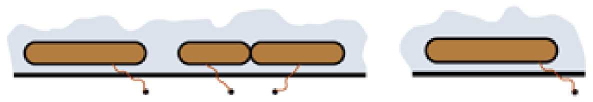
4. Provide a counterexample that demonstrates that choosing the item with the highest £/kg ratio is not guaranteed to give the optimal solution for the knapsack problem.

Exercises

1. Solve *Exercise 4* from the pre-sheet.

2. Provide algorithms that solve both the *unbounded* (there is infinitely many of each item) and *0-1* (there is exactly one copy of every item) versions of the knapsack problem.

3. There are N fishermen in a village—each of them owns a boat of length l_i, and has a position p_i along the riverbank where he may anchor his boat. Every boat has to be anchored such that at least one point of the boat is directly adjacent to the anchoring position. Here is an example of an allowed and disallowed anchoring, respectively:



   Provide an algorithm that determines the maximal number of boats that can be anchored simultaneously (without overlapping).

4. Consider the problem of booking activities at the sports hall (as given in the lecture notes)—however, now we also have a constraint that each activity a_i gives particular *value*, v_i, to the University, and our objective is to maximise the overall value of selected events[1]. Propose an algorithm to determine which activities should be scheduled in order to do so.

5. A *palindrome* is a string which reads the same way left-to-right as right-to-left (e.g. `anaana` is a palindrome, while `banana` is not). Determine the minimal amount of characters you need to insert into a given string (at any position) to make it a palindrome. For example, if you are given the string `Ab3bd`, you require two characters (in order to make, e.g. `dAb3bAd`). [*Hint:* It might be beneficial to attempt the implementation exercise first.]

6. Propose a solution (no need to implement anything!) for *microchallenge 1* from the course materials (the *segmented least squares* problem). An archived version of this microchallenge should be locatable at:
   `https://web.archive.org/web/20170204175128/http://www.cl.cam.ac.uk/teaching/1617/Algorithms/materials.html`

Implementation

The problem of finding the *longest common subsequence* of two strings, A and B, represents a classic example of dynamic programming algorithms, and a fundamental problem for areas such as bioinformatics (where it is applied to aligning two DNA sequences). Reminder: a subsequence of a string is a string obtained when deleting some characters from it (possibly none).
The optimal, $O(n \cdot m)$-time algorithm (where n and m are the lengths of the two strings) proceeds as follows:

- Let $dp_{i,j}$ represent the longest common subsequence length of $A_{1:i}$ and $B_{1:j}$ (for $0 \leq i \leq n$, $0 \leq j \leq m$).
- Base case: $i = 0$ or $j = 0$. In this case, either the first or the second string is empty, and therefore $dp_{i,0} = dp_{0,j} = 0$.

[1]**N.B.** this is a generalisation of the original problem, where $v_i = 1$ for all i.

- Recurrence relation: for general $dp_{i,j}$:
  - If $A_i = B_j$, then we may pair up these two characters and reduce the problem to the LCS length of the remaining prefixes, i.e. $dp_{i,j} = dp_{i-1,j-1} + 1$.
  - Otherwise, we may either discard A_i or B_j from consideration and take whatever's best out of the two options[2]: $dp_{i,j} = \max(dp_{i-1,j}, dp_{i,j-1})$
- $dp_{n,m}$ will then contain the optimal LCS length.

Implement the algorithm as per the above description, and provide a subroutine that will actually extract the LCS itself with guidance from the computed dp matrix. Verify that your algorithm works on inputs `aleks` and `abcdef` ($dp_{5,6} = 2$, and the output should be `ae`).

Job interviews

1. You are given N intervals on the real number line, each represented with its starting and finishing coordinate (s_i, f_i). Determine the size, k, of the largest subset of these intervals $S' = \{(s_1, f_1), \dots, (s_k, f_k)\}$, such that, when sorted by length, each interval is fully contained by all of its successors, i.e.:

$$s_k \leq s_{k-1} \leq \dots \leq s_2 \leq s_1 \leq f_1 \leq f_2 \leq \dots \leq f_{k-1} \leq f_k$$

[2]**N.B.** these have already been solved!