CS180 — Algorithms and Complexity

HW#4 — Dynamic Programming

Due: 11:55pm Wednesday March 4, 2015

D. Stott Parker, Yuh-Jie Chen, Xiaoran Xu

stott@cs.ucla.edu, eyjchen@cs.ucla.edu, xrxu@cs.ucla.edu

Using CCLE, please upload your answers as a PDF document by the deadline.

1. **Dynamic Coin Grabbing**

   In DYNAMIC COIN GRABBING, there is a sequence $c_1, \ldots, c_n$ of *coins*, where each coin c_i has value v_i. Here n is an even number. Two players take turns picking a coin from the sequence, but can only pick the first or the last coin of the (remaining) sequence. The objective is to collect coins with the largest total value.

   (a) Show a sequence of $n \geq 6$ coins for which it is not optimal for the first player to start by picking up the available coin of larger value. That is, give an example for which the natural greedy strategy is suboptimal.

   (b) Give an $O(n^2)$ algorithm to compute an optimal strategy for the first player. Given the initial sequence, your algorithm should precompute some information in $O(n^2)$ time, and then the first player should be able to make each move optimally in $O(1)$ time by looking up the precomputed information.

   (c) The first player has a very simple strategy to 'win' (get a higher total than the other player): if the sum of even-numbered coins is higher, always remove the coin c_i on the end whose original index i was even; and similarly for odd-numbered coins. Find a sequence of $n \geq 6$ coins for which the dynamic programming strategy guarantees the first player a higher total value than the simple strategy.

2. **Box Stacking**

   In its quest for global domination, Amazon buys boxes in different sizes from all sources. The boxes come in stacks, and to prevent unstable boxes Amazon insists that the stacks be properly ordered — so that a larger box is never stacked on top of a smaller box. Amazon however then must merge boxes from different sources. They are not sure how to do this (having trouble thinking outside the box), and hire you as a box consultant.

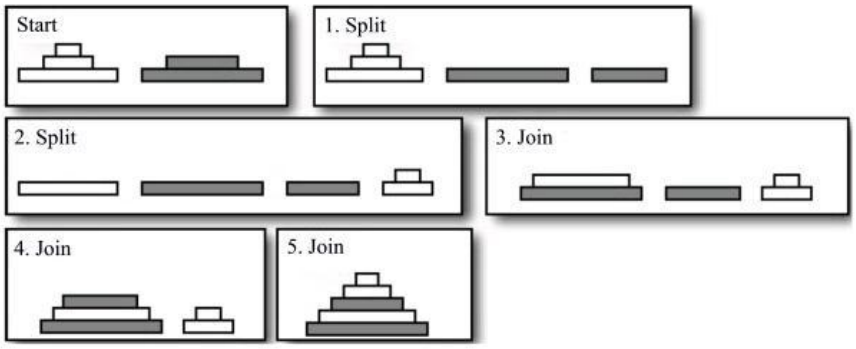

   Figure 1: *example of box stacking: the two stacks of boxes shown (white and gray) can be merged into a single stack in a total of 5 steps, of splitting and merging boxes.*

   Box stacking is implemented with two basic steps:

   - *Splitting*: a stack of boxes can be split into two stacks. (And there is no limit on the number of splits.)
   - *Joining*: when the boxes in one stack are larger than in another, the smaller stack can be put on top of the larger stack. (Note: only whole stacks are moved; there is no *shifting* of boxes from stack to stack. To achieve the effect of shifting boxes, a split and a join step must be performed.)

   Figure 1 shows an example of combining two stacks into one in a (minimal) sequence of 5 steps.

   (a) Give an algorithm for box stacking that uses as few split and join steps as possible, when given two stacks of boxes — which are represented as ascendingly sorted sequences of numbers (box sizes). Duplicate numbers are permitted in the input sequences, and the two stacks can contain identical numbers.
   Hint: the goal is to find the optimal split of one of two stacks.

   (b) Determine the time complexity of your algorithm.

3. **Longest Ascending Subsequence**

In the LONGEST ASCENDING SUBSEQUENCE problem we are given a sequence x of n positive integers x_1, x_2, ..., x_n and want to find an ascending (nondecreasing) subsequence $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_k}$ (where $1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq n$) such that this subsequence has maximal length — i.e., k is as large as possible.

For example, in the sequence

1	8	2	4	2	3	1	7

the longest ascending subsequence is 1, 2, 2, 3, 7, which has length 5.

(a) Suppose *Len*(x,i) gives the length of the longest ascending subsequence **ending at position** x_i, where $1 \leq i \leq n$.
(For the sequence x above, *Len*(x,1) =1, *Len*(x,2) = 2, *Len*(x,3) = 2, ... *Len*(x,8) = 5.)
Give a Dynamic Programming recursion equation for *Len*(x,i) in terms of *Len*(x,1), ..., *Len*(x,$(i-1)$).

(b) Show how to use these recursion equations to obtain a polynomial-time algorithm for solving this problem.

(c) What is the time complexity of this algorithm?

4. **Currency Exchange**

Recently currency markets have been volatile. It has been a good time for currency traders and arbitrageurs. Every day they study tables of exchange rates between leading currencies like this:

13Feb2015	GBP	EUR	USD	JPY	CHF	CAD	AUD	NZD	RUB	ZAR	MXN	AED
GBP	1.0000	1.3458	1.5386	182.9757	1.4272	1.9238	1.9848	2.0686	100.5796	18.0231	22.997	5.6465
EUR	0.743	1.0000	1.1432	135.9575	1.0605	1.4295	1.4748	1.537	74.7343	13.3918	17.0876	4.1956
USD	0.65	0.8747	1.0000	118.9266	0.9276	1.2504	1.2901	1.3445	65.3726	11.7143	14.9471	3.67
JPY	0.0055	0.0074	0.0084	1.0000	0.0078	0.0105	0.0108	0.0113	0.5497	0.0985	0.1257	0.0309
CHF	0.7007	0.943	1.078	128.2034	1.0000	1.3479	1.3907	1.4493	70.4719	12.628	16.113	3.9563
CAD	0.5198	0.6996	0.7997	95.1106	0.7419	1.0000	1.0317	1.0752	52.2812	9.3684	11.9538	2.9351
AUD	0.5038	0.6781	0.7752	92.1876	0.7191	0.9693	1.0000	1.0422	50.6744	9.0805	11.5864	2.8449
NZD	0.4834	0.6506	0.7438	88.456	0.69	0.93	0.9595	1.0000	48.6233	8.7129	11.1175	2.7297
RUB	0.0099	0.0134	0.0153	1.8192	0.0142	0.0191	0.0197	0.0206	1.0000	0.1792	0.2286	0.0561
ZAR	0.0555	0.0747	0.0854	10.1523	0.0792	0.1067	0.1101	0.1148	5.5806	1.0000	1.276	0.3133
MXN	0.0435	0.0585	0.0669	7.9565	0.0621	0.0837	0.0863	0.0899	4.3736	0.7837	1.0000	0.2455
AED	0.1771	0.2383	0.2725	32.405	0.2528	0.3407	0.3515	0.3663	17.8127	3.1919	4.0728	1.0000

www.exchangerates.org.uk/currency/currency-exchange-rates-table.html (as of February 13)

Each matrix entry is the conversion rate for converting from the row currency to the column currency.

Given a matrix R like this, we seek an *arbitrage* — a way of increasing money by starting with an amount in some currency and then increasing it with a cycle of currency trades.

For example, the USD $\rightarrow$ RUB rate is 65.3726, and the USD $\rightarrow$ RUB $\rightarrow$ EUR rate is $65.3726 \cdot 0.0134 = 0.87599284$, which is larger than the direct exchange rate USD $\rightarrow$ EUR (0.8747). Furthermore the path USD $\rightarrow$ RUB $\rightarrow$ EUR $\rightarrow$ USD has rate 1.0014350. So in theory a cycle of currency trades could gain 0.14%.

(a) Because each entry in the matrix R is a rate, if we take the *log* of entries in the matrix, then *sums* of matrix entries are equal to logs of corresponding products of rates. Prove that the weight of a path using the log-transformed matrix $L = \log R$ is the log of the product of rates along that path. Also show the weight of the same path in the matrix $C = -L = -\log R$ corresponds to the *inverse* of the rate in L.

(b) Give an efficient algorithm (in pseudocode) to compute the shortest paths from each currency to each other currency using the matrix $C = -L = -\log R$.

(c) The matrix C can have negative entries, and if we view it as the adjacency matrix of a graph, it can have negative cycles. Any negative cycle in the matrix C will correspond to a gain of money by currency trades — and an opportunity for arbitrage. Give an efficient algorithm for finding the maximally negative cycle in the matrix C. (If there is no negative cycle, it should determine that.)

Hint: section 6.10 of [KT] may help.