CS2223 Algorithms B Term 2013 Exam 3 Solutions

By Prof. Carolina Ruiz
Dept. of Computer Science
WPI

Dec. 17, 2013

PROBLEM I: Dynamic Programming (40 points)

Consider the problem of calculating the binomial coefficient $\binom{n}{k}$ (also called "n choose k" or "C(n,k)") when $0 \leq k \leq n$:

$$\binom{n}{k} = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k}, & \text{if } 0 < k < n \end{cases}$$

1. **(25 Points)** Write a dynamic programming algorithm (in pseudo–code) that receives n and k as input (where $0 \leq k \leq n$) and outputs $\binom{n}{k}$ using the definition of $\binom{n}{k}$ above. Your algorithm must use *memoization*. Clearly explain the data structures that you use.

   Solution:

Note: This problem was adapted from Section 8.1.1 in Brassard & Bratley's "Fundamentals of Algorithms" textbook. page 260.

We will use an $(n+1)\times(k+1)$ matrix M to store solutions to subproblems. $M[i,j] = \binom{n}{k}$ for all k and n, where $0 \leq k \leq n$. For the base case of the recursion, we define $\binom{0}{k} = 0$ if $k > 0$, and $\binom{0}{0} = 1$. We will include an additional row for $n = 0$ for this base case values.

	0	1	…	j	…	k
0	1	0	…	0	…	0
1	1	…	…	…	…	…
…	1	…	…	…	…	…
i	…	…	…	$M[i-1,j-1]+M[i-1,j]$	…	…
…	1	…	…	…	…	…
n	1	…	…	…	…	…

Note that technically we don't need to fill up the whole matrix, as it is a "symmetric" matrix: $\binom{n}{k} = \binom{n}{n-k}$, so it would suffice to fill up either the lower or the upper triangular submatrix.

We can choose to fill up this matrix in a "bottom up" or in a "top down" way. Here we present both approaches for illustration purposes, but one of the two is enough.

- Bottom up approach.

```
$ChooseBottomUp(n, k)$:
    ## We assume here that $0 \leq k \leq n$.
    ## Also, we assume that a global variable $M[0, 1, \ldots, n][0, 1, \ldots, k]$ has been created
    if $(k == 0)$ or $(k == n)$:
        return 1

    ## $i$ will be an index over the rows, and $j$ an index over the columns
    for $j = 1$ to $k$:
        $M[0, j] = 0$

    for $i = 0$ to $n$:
        $M[i, 0] = 1$
        for $j = 1$ to $k$:
            $M[i, j] = M[i-1, j-1] + M[i-1, j]$

    return $M[n, k]$
```

- Top down approach.

```
$MainChooseTopDown(n, k)$:
    ## We assume here that $0 \leq k \leq n$.
    ## Also, we assume that a global variable $M[0, 1, \ldots, n][0, 1, \ldots, k]$ has been created
    if $(k == 0)$ or $(k == n)$:
        return 1

    ## $i$ will be an index over the rows, and $j$ an index over the columns
    for $j = 1$ to $k$:
        $M[0, j] = 0$

    for $i = 0$ to $n$:
        $M[i, 0] = 1$
        for $j = 1$ to $k$:
            $M[i, j] =$ EMPTY

    return $ChooseTopDown(n, k)$

$ChooseTopDown(i, j)$:
    if $M[i, j] ==$ EMPTY
        $M[i, j] = ChooseTopDown(i-1, j-1) + ChooseTopDown(i-1, j)$
    return $M[i, j]$
```

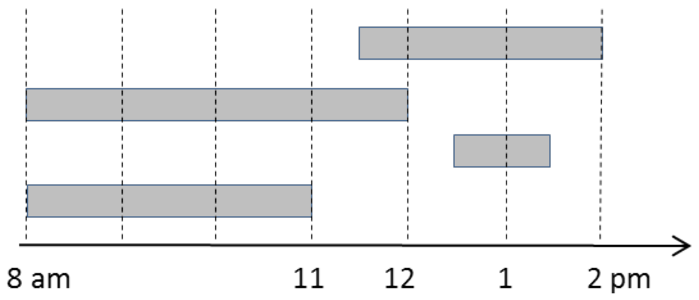
2. **(15 Points)** What is the time complexity of your algorithm? **Explain your answer.**

   Solution:

Both versions of the algorithm (bottom up and top down) have the same time complexity. The key observation is that the algorithm visits each position of the M matrix filling it with a value, which is either a constant or is calculated by adding together 2 other values in the matrix. Hence, calculating the value of a matrix cell takes constant time, and since there are $(n+1) \times (k+1)$ cells in the matrix, the total runtime of the algorithm is $\Theta(nk)$.

PROBLEM II: (35 points)
Consider the following scheduling problem: given a set of lectures (e.g., CS2223, CS2022, BCB4003, MA1024, ...), each with a start and finish time (e.g., CS2223 starts at 2 pm and finishes at 3 pm), schedule all the lectures using the minimum possible number of classrooms. Lectures assigned to the same classroom cannot overlap in time. In other words, lectures that overlap in time must be assigned to different classrooms. Assume that we have a large enough supply of classrooms, but we need to schedule all the lectures using the minimum possible number of classrooms.

Consider for example the following set of 4 lecture requests.



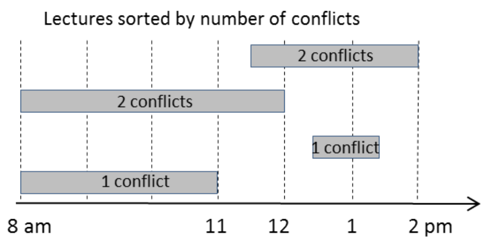
A general greedy algorithm to tackle this optimization problem is given below:

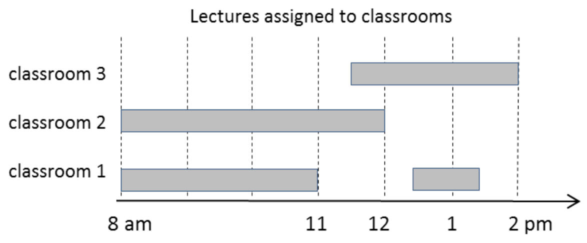
```
**AssignLecturesToClassrooms** $(n, lecturesList)$
# Inputs:
# $n$: number of lectures to be scheduled
# $lecturesList$: list of $n$ lectures to be scheduled

  Sort $lecturesList$ according to a given ordering
    # (e.g., Earliest start time, or Earliest finish time, or Shortest interval, or Fewest conflicts)
  resulting on a sorted list of lectures $lecture_1 \leq lecture_2 \leq \ldots \leq lecture_n$

  $numClassroomsUsed = 0$

  for $j = 1$ to $n$
    if $lecture_j$ is compatible with some classroom $k$, where $1 \leq k \leq numClassroomsUsed$
      Schedule $lecture_j$ in any such compatible classroom $k$.
    else
      Allocate a new classroom $numClassroomsUsed + 1$.
      Schedule $lecture_j$ in classroom $numClassroomsUsed + 1$.
      $numClassroomsUsed = numClassroomsUsed + 1$

  return $numClassroomsUsed$ and schedule.
```

1. Consider the alternative orderings below that could be used to sort the input list of lectures. For each ordering, show the sorted list of lectures, and the $numClassroomsUsed$ and the schedule output by the AssignLecturesToClassrooms algorithm above when applied to the example of four lecture requests depicted on the previous page. **Show your work.** As an illustration, the answer to the *Fewest conflicts ordering* is provided below.

   1.1. Fewest conflicts: For each lecture j, count the number of lectures it conflicts with. Consider lectures in ascending order of conflict count.

   Solution: 3 classrooms are needed if lectures are considered in ascending order of conflict count:



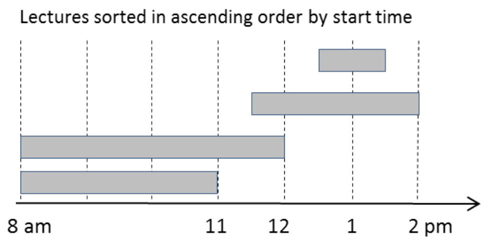


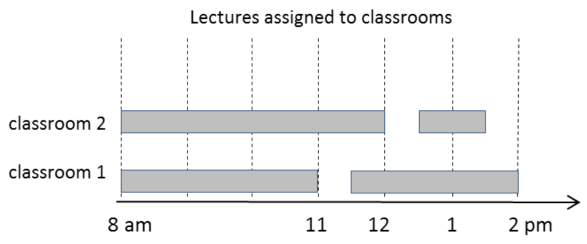
   1.2. (10 points) Earliest start time: Consider lectures in ascending order of start time.

   Solution:

---

2 classrooms are needed if lectures are considered in ascending order of start time. Note that here, if more than one classroom is compatible with a lecture, we choose the first compatible classroom.



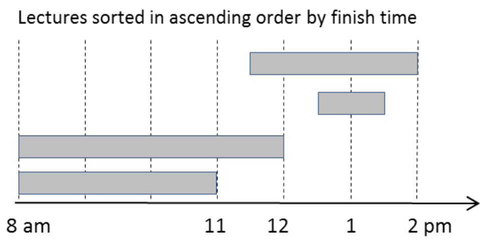


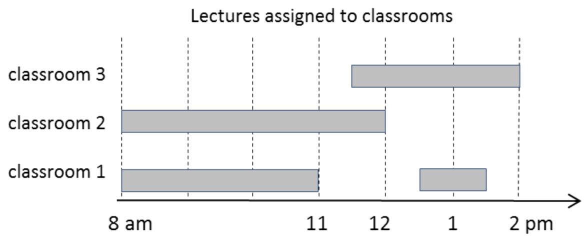
1.3. (10 points) Earliest finish time: Consider lectures in ascending order of finish time.

Solution:

---

3 classrooms are needed if lectures are considered in ascending order of finish time. Note that here, if more than one classroom is compatible with a lecture, we choose the first compatible classroom.



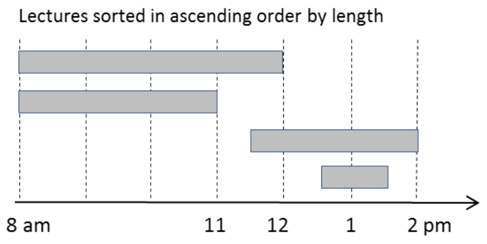


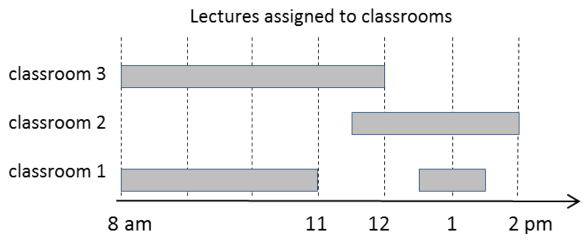
1.4. (10 points) Shortest interval: Consider lectures in ascending order of finish minus start times.

Solution:

---

3 classrooms are needed if lectures are considered from shortest to longest. Note that here, if more than one classroom is compatible with a lecture, we choose the first compatible classroom.





---

2. **(5 Points)** Which of the four orderings above do NOT yield an optimal schedule (i.e., a minimum number of classrooms) when used in the AssignLecturesToClassrooms algorithm? **Explain your answer.**

   Solution:

We can see that for the given input lecture requests, an optimal schedule would use 2 classrooms (clearly all lectures cannot be scheduled in less than 2 classrooms). Sorting lectures by start time and then assigning them to classrooms yielded a schedule that uses the minimun number of classrooms. On the other hand, sorting the lectures in ascending order by finish time, length, or number of conflicts did not.

Although not required for this exam problem, one can prove that the version of the AssignLecturesToClassrooms procedure that sorts the lectures in ascending order by start time always produces an optimal schedule. See below for a reference to such a proof.

Note: This exam problem was adapted from the slides for Chapter 4 of Kleinberg's and Tardos' Algorithm Design textbook, available at Prof. Wayne's Algorithms Course Webpage at Princeton University, see slides 15–22:
http://www.cs.princeton.edu/∼wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf

---

PROBLEM III: Graph Algorithms (25 points)

Assume that we are given a directed graph $G = (V, E)$ on which each edge (u, v) in E has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Therefore, the reliability of a path is the product of the reliabilities of the edges in the path.

Let s be a vertex in the graph. Consider the problem of designing an efficient algorithm to find the most reliable paths between s and other vertices in the graph. In other words, for each vertix u in the graph, the algorithm will output the most realible path from s to u. Such an algorithm can be obtained by making small modifications to Dijkstra's algorithm.

List all of the modifications to Dijkstra's algorithm that are needed to solve this problem and **explain each of them** carefully.

Solution:

Note: This problem is adapted from Cormen et. al.'s textbook Exercise 24.3-6 page 663. The textbook solution is available online from the textbook's website: See "Solutions to Exercises and Problems" at http://mitpress.mit.edu/books/introduction-algorithms

Our solution here is different from that of the textbook. We include below Dijkstra's algorithm (pseudo–code taken from the textbook) marking in red the parts that need to be changed in order to calculate the most reliable paths rather than the cheapest paths.

```
$Dijstra(G, w, s)$:
    $InitSingleSource(G, s)$
    $S = \emptyset$
    $Q = G.V$
    while $Q \neq \emptyset$:
        $u = ExtractMin(Q)$ $\rightsquigarrow$ change to $u = ExtractMax(Q)$
        $S = S \cup \{u\}$
        for each vertex $v \in G.Adj[u]$:
            $Relax(u, v, w)$ $\rightsquigarrow$ change to $Relax(u, v, r)$

$InitSingleSource(G, s)$:
    for each $v \in G.V$:
        $v.d = \infty$ $\rightsquigarrow$ change to $v.d = -\infty$
        $v.\pi = nil$
    $s.d = 0$ $\rightsquigarrow$ change to $s.d = 1$

$Relax(u, v, w)$:
    if $v.d > u.d + w(u, v)$: $\rightsquigarrow$ change to if $v.d < u.d \times r(u, v)$:
        $v.d = u.d + w(u, v)$ $\rightsquigarrow$ change to $v.d = u.d \times r(u, v)$
        $v.\pi = u$
```
