Homework2 for EECS 340

Yu Mi,yxm319

February 12, 2018

1 Give a recursive algorithm to find the average (mean) value of an array of 2^k decimal numbers, where $k \in \mathbb{N}$.

Answer: The proposed algorithm is as follow:

Algorithm A1: Average(L)
Data: A list of 2^k decimal numbers L.
Result: The average of all the numbers in L.
if L.length()$= 0$ **then**
  return $L[0]$
else
  $length \leftarrow L.length()$
  return $0.5\times$(Average($L[0, length/2-1]$+Average($L[length/2, length]$))
end if

2 R-12.6

Question:Suppose we are given a set of telescope observation requests, specified by triples, of (s_i, f_i, b_i), defining the start times, finish times, and benefits of each observation request as

$$L = (1,2,5),(1,3,4),(2,4,7),(3,5,2),(1,6,3),(4,7,5),(6,8,7),(7,9,4)$$

Solve the telescope scheduling problem for this set of observation requests.
Answer: The time of scheduling can be shown in Fig.1, the number in the bar means the value of such task.

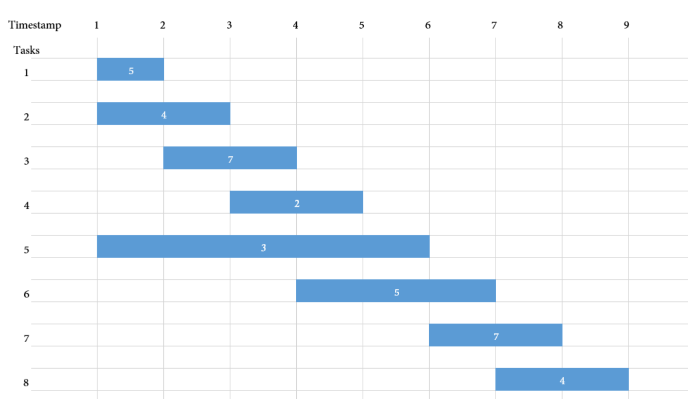

Figure 1: Time of tasks

Based on what we have discussed on class, we can have a table of B_i which stands for the maximum benefit that can be achieved with the first i requests in the task list.

To fill this table, we follow the algorithm as follow:

$B[0] \leftarrow 0$
for $i = 1$ to n **do**
  $B[i] \leftarrow max(B[i-1], B[P[i]] + b_i)$
end for

Here the $P[i]$ stands for the array which gives the predecessor index for each request i, and b_i means the value of each single task. The table is shown as Table 1.

Table 1: B_i values

i	0	1	2	3	4	5	6	7	8
B_i	0	5	4	12	6	3	17	13	21

As we can see, the highest value is B_8, which includes task $1, 3, 6, 8$ that we should select. The corresponding triples are $(1, 2, 5), (2, 4, 7), (4, 7, 5), (7, 9, 4)$.

3 Implement *det-bogoSort* in pseudocode using recursion

Answer: This algorithm is described as follows:

Algorithm BogoSort(S, L, L_{temp})
Data: Input list L, as is described in the question, an initially empty set of lists S, and an initially empty list L_{temp}
Result: A sorted copy of L
if $size(L) = 0$ **then**
  $flag \leftarrow true$
  for $i \leftarrow 1$ to $size(L_{temp}) - 1$ **do**
    if $L_{temp}[i-1] > L_{temp}[i]$ **then**
      $flag \leftarrow false$
      break
    end if
  end for
  if $flag = true$ **then**
    return L_{temp}
  end if
else
  for $i \leftarrow 0$ to $size(L) - 1$ **do**
    L_{temp}.append($L[i]$)
    L.remove(i)
    BogoSort(S, L, L_{temp})
  end for
end if

NOTE: in the operations of lists, L_{temp}.append($L[i]$) means to append the element $L[i]$ at the end of list L_{temp}. And L.remove(i) means to remove the ith element in list L.

4 Write pseudo-code for a new recursive function *moving-average*

Answer: The algorithm is shown as follows:

Algorithm moving-average(S, t_{start}, t_{end})
Data: Stock-price time series data S,a start time t_{start}, and a ending time t_{end}.
Result: The average stock price from t_{start} through t_{end}.
$i \leftarrow 0$
if $t_{start} = t_{end}$ **then**
  return 0
end if
for $i \leftarrow t_{end}$ to t_{start} **do**
  if $t_{start}\%(i - t_{start}) = 0$ **and** $((i - t_{start})$ is a power of 2) **then**
    return $\frac{i-t_{start}}{t_{end}-t_{start}} \times$ pow-of-two-average(S, t_{start}, i)+$\frac{t_{end}-i}{t_{end}-tstart} \times$ moving-average(S, i, t_{end})
    break
  end if
end for
return 0

5 Write pseudo-code for a recursive function to compute *checker-sum(A)*$_{i,j}$ given the triple (A, i, j)

Answer: The algorithm is shown as follows:

Algorithm checker-sum(A, i, j)
Data: A matrix A, and index i and j
Result: The checkerboard sum of matrix A
if $j > 1$ **then**
  return compute-row(A, i, j) $-$ checker-sum($A, i, j - 1$)
else
  return compute-row(A, i, j)
end if

Algorithm compute-row(A, i, j)
Data: A matrix A, and index i and j
Result: The checkerboard sum of row j in matrix A
if $i > 1$ **then**
  return $A(i, j)$ $-$ compute-row($A, i - 1, j$)
else
  return $A(i, j)$
end if

6 Read section 17.1 in the textbook and then solve the following problems

6.1 R-17.7

Question: Show that the CLINQUE problem is in *NP*

6.2 C-17.3

Question: Show that we can deterministically simulate in polynomial time any nondeterministic algorithm A that runs in polynomial time and makes at most $O(\log n)$ calls to the **choose** method, where n is the size of the input to A.