COMP3711: Design and Analysis of Algorithms

Tutorial 6b

HKUST

Question 1

Give an $O(n^2)$ time dynamic programming algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers, i.e, each successive number in the subsequence is greater than or equal to its predecessor.

For example, if the input sequence is

$$\langle 5, 24, 8, 17, 12, 45 \rangle,$$

the output should be either $\langle 5, 8, 12, 45 \rangle$ or $\langle 5, 8, 17, 45 \rangle$.

Solution 1

We first give an algorithm which finds the **length** of the longest increasing subsequence; later, we will modify it to report a subsequence with this length.

Let $X_i = \langle x_1, \cdots, x_i \rangle$ denote the prefix of X consisting of the first i items.

Define $c[i]$ to be the length of the longest increasing subsequence that **ends** with x_i.

It is clear that the length of the longest increasing subsequence in X is given by

$$\max_{1 \le i \le n} c[i].$$

Solution 1

$c[i]$ is the length of the longest increasing subsequence **that ends with $\boldsymbol{x_i}$**.

If $c[i] \neq 1$, longest increasing subsequence **that ends with $\boldsymbol{x_i}$** has form $\langle Z, x_i \rangle$ where Z is the longest increasing subsequence **that ends with $\boldsymbol{x_r}$**
for some $r < i$ and $x_r < x_i$.

If sequence has only one item or all items are $>$ than x_i answer must be 1.

Otherwise length of the longest increasing subsequence **ending in $\boldsymbol{x_i}$** is $1 +$ the length of the longest increasing subsequence ending at a number x_r to the left of x_i such that x_r is no greater than the x_i.

This yields the following recurrence relation:

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

Solution 1

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

We do not write the pseudocode but just note that we store the $c[i]$'s in an array whose entries are computed in order of increasing i .

After computing the c array we run through all the entries to find the maximum value.

This is the length of the longest increasing subsequence in X.

For every i it takes $O(i)$ time to calculate c_i.

=> the running time is $O(\sum_{i=1}^{n} i) = O(n^2)$.

Solution 1

$$c[i] = \begin{cases} 1 & \text{if } i = 1 \\ 1 & \text{if } x_r > x_i \text{ for all } 1 \le r < i \\ \max\limits_{\substack{1 \le r < i \\ x_r \le x_i}} c[r] + 1 & \text{other cases} \end{cases}$$

To report the optimal subsequence, we need to store for each i, not only $c[i]$, but also the value of r which achieves the maximum in the recurrence relation.

Denote this by $r[i]$. Then we can trace the solution as follows:

Suppose $c[k] = \max\limits_{1 \le i \le n} c[i]$.
Then x_k is the last item in the optimal subsequence.
The 2nd to last item is $x_{r[k]}$, the 3rd to last item is $x_{r[r[k]]}$ and so on until we have found all the items of the optimal subsequence.

Running time of adding this step is $O(n)$ so entire algorithm is still $O(n^2)$.

Solution 1: Alternative Solution

This problem can also be solved using the Longest Common Subsequence Algorithm

Let $\quad X = \langle x_1, \cdots, x_n \rangle$ be the original input.
Set $\quad Y = \langle y, \cdots, y_m \rangle$ be the items from X sorted.

Example: $\mathrm{X} = \langle 5, 24, 8, 17, 12, 45, 12 \rangle$, $\quad \mathrm{Y} = \langle 5, 8, 12, 12, 17, 24, 45 \rangle$

Then LCS(X, Y) is exactly the Longest Increasing Subsequence of X (why?)

Solution 1: Alternative Solution

This problem can also be solved using the Longest Common Subsequence Algorithm

Let $X = \langle x_1, \cdots, x_n \rangle$ be the original input.
Set $Y = \langle y, \cdots, y_m \rangle$ be the items from X sorted.

Example: $\mathrm{X} = \langle 5, 24, 8, 17, 12, 45, 12 \rangle$, $\mathrm{Y} = \langle 5, 8, 12, 12, 17, 24, 45 \rangle$

Then LCS(X, Y) is exactly the Longest Increasing Subsequence of X (why?)

Since LCS(X, Y) uses $O(n^2)$ time, this new algorithm also uses $O(n^2)$ time.

Surprisingly, there is also an $O(n \log n)$ algorithm for solving the problem. See https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf

Question 2

The subset sum problem is: Given a set of n positive integers,
$S = \{x_1, x_2, \cdots, x_n\}$ and an integer W
determine whether there is a subset $S' \subseteq S$, such that the sum of the elements in S' is equal to W.

For example, Let $S = \{4,2,8,9\}$.

If $W = 11$, then the answer is "yes"
because the elements of $S' = \{2,9\}$ sum to 11.
If $W = 7$, the answer is "no".

Give a dynamic programming solution to the subset sum problem that runs in $O(nW)$ time.
Justify the correctness and running time of your algorithm.

Solution 2

Define a Boolean array $A[i,j]$, $0 \le i \le n$ and $0 \le j \le W$ as follows:

$A[i,j] = true$ if there is a subset of $\{x_1, x_2, \cdots, x_i\}$ that sums to j,
Otherwise $A[i,j] = False$.

- For all i, $A[i,0] = True$ (choosing no items equals 0)
- $A[0,j] = False$ if $j > 0$.
- If $x_i > j$ then item i is too large to use => $A[i,j] = A[i-1,j]$
- Otherwise, $A[i,j] = (A[i-1,j-x_i]$ OR $A[i-1,j])$

  This is because there are two true solution possibilities:

  (i) solution uses x_i.

  This can only happen if $j - x_i$ can be solved with first $i-1$ items

  (ii) solution does not use x_i

  in which case j can be solved with first $i-1$ items

$$\mathrm{A}[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Solution 2

$$A[i,j] = \begin{cases} \text{True} & \text{if } j = 0 \\ \text{False} & \text{if } i = 0 \text{ and } j > 0 \\ A[i-1,j] & \text{if } x_i > j \\ A[i-1,j-x_i] \text{ OR } A[i-1,j] & \text{Otherwise} \end{cases}$$

Dynamic-SubsetSum(x, n, W)

$A[0,0] = True$

for $j = 1$ to W do

$A[0,j] = False$

for $i = 1$ to n do

$A[i,0] = True$

for $j = 1$ to W do

if $x_i > j$ then

$A[i,j] = A[i-1,j]$

else $A[i,j] = (A[i-1,j-x_i] \text{ OR } A[i-1,j])$

It is easy to see that this runs in $O(nW)$ time.

There will be a solution if and only if $A[n,W] = True$.

Question 3: The (Restricted) Max-Sum Problem

Let *A* be a sequence of *n* positive numbers $a_1, a_2, \ldots, a_n$.

Find a subset *S* of *A* that has the maximum sum,

provided that, if we select $a_i \in S$, then we cannot select a_{i-1} or a_{i+1}.

For example, if *A*= 1, 8, 6, 3, 7,
the max possible sum is *S* = {8,7}

Solution 3: Dynamic Programming Solution

General idea:

Let A_i be the subsequence of A containing the first i numbers ($i \leq n$):

$$A_i = a_1, a_2, \dots, a_i$$

Let S_i be the solution of problem A_i.

Let W_i be the sum of numbers in S_i.

Two possibilities for a_i:

$a_i \in S_i \quad \Rightarrow \quad a_{i-1} \notin S_i, \quad and \quad W_i = W_{i-2} + a_i.$

$a_i \notin S_i \quad \Rightarrow \quad a_{i-1}\ can\ be\ in\ S_i, \quad and \quad W_i = W_{i-1}.$

Solution is larger of the two cases.

$$W_i = \max\{W_{i-2} + a_i, W_{i-1}\}.$$

Solve the problem incrementally from smaller to larger,

i.e. for A_1, A_2, ..., A_n. The final solution is A_n.

Solution 3: DP pseudocode

$W[1] = a_1$; $b[1] = true$
// in general $b[i] = true$ means that a_i is in S_i

If $a_2 > a_1$
    $W[2] = a_2$; $b[2] = true$ // a_2 is in S_2
  else
    $W[2] = a_1$; $b[2] = false$ // a_2 is not in S_2

for i =3 to n
    If $W[i-2] + a_i > W[i-1]$
        $W[i] = W[i-2] + a_i$
        $b[i] = true$ // a_i is in S_i
    else
        $W[i] = W[i-1]$
        $b[i] = false$ // a_i is not in S_i

← Initial Conditions

← Recursive Solution

Cost: $\Theta(n)$

Example:
for A= 1, 8, 6, 3, 7,
we have

$W[1] = 1$, $b[1] = 1$
$W[2] = 8$, $b[2] = 1$
$W[3] = 8$, $b[3] = 0$
$W[4] = 11$, $b[4] = 1$
$W[5] = 15$, $b[5] = 1$

Solution 3: Printing

$i = n$

while $i > 0$

  if $b[i]$ is true

    Output a_i

    $i = i-2$

  else

    $i = i-1$

$W[1] = 1$, $b[1] = 1$
$W[2] = 8$, $b[2] = 1$
$W[3] = 8$, $b[3] = 0$
$W[4] = 11$, $b[4] = 1$
$W[5] = 15$, $b[5] = 1$

Example: for A= 1, 8, 6, 3, 7, we have

$b[5] = 1$; therefore, we print $a_5 = 7$ and set i=3
$b[3] = 0$; therefore, we set i=2
$b[2] = 1$; therefore, we print $a_2 = 8$ and set i=0