CSE421: Design and Analysis of Algorithms

Homework 5

Anup Rao Due:

1. Given a sequence of integers $x_1, \ldots, x_n$ (possibly including negative integers) and an interval of coordinates $I = [i, j]$, write x_I to denote the sum $\sum_{i \le k \le j} x_k$. Give a linear time algorithm to find the interval that maximizes x_I.

   Solution. Let $OPT(j)$ denote $\max_{i \le j} \sum_i x_i$. In words, $Opt(j)$ gives the optimal value of all intervals that end at j. Our algorithm will compute $Opt(j)$ for every choice of j. Then final solution is then given by $\max_j Opt(j)$.

   To compute $Opt(j)$ in terms of smaller j, note that there are two cases. If the optimal interval ending at j includes only x_j, then $Opt(j) = x_j$. Otherwise, the optimal interval must include $j - 1$, which case we must have $Opt(j) = Opt(j-1) + x_j$. So we always have

$$Opt(j) = \max\{x_j, x_j + Opt(j-1)\}$$

   Input: A sequence of integers
   Result: Value of best interval
   Set M to be an array of n elements; Set $M[1] = x_1$;
   for *integer* j *in* 2 *through* n **do**
   | Set $M[j] = \max\{x_j, x_j + M[j-1]\}$;
   end
   output $\max_j M[j]$;

   Runtime: The algorithm goes through the sequence twice. So the algorithm has runtime $O(n)$.

2. Given a sequence of characters $c_1, \ldots, c_n$, we say that a subsequence is a *palindrome* if it reads the same forwards and backwards. For example, "a,b,a,c,a,b,a" is a palindrome. Give an $O(n^2)$ time algorithm to find the longest palindrome subsequence in the input sequence $c_1, \ldots, c_n$. For example, in the sequence $c, l, m, a, l, f, d, c, a, f, m$, the longest palindrome subsequence is m, a, d, a, m. HINT: For $i < j$, let $p(i, j)$ denote the length of the longest palindrome in $x_i, \ldots, x_j$. Express $p(i, j)$ in terms of $p(i+1, j), p(i, j-1), p(i+1, j-1)$. Evaluate the values $p(i, j)$ in order of increasing $|i - j|$.

   Solution. As in the hint, we shall express $p(i, j)$ in terms of the optimal solution for smaller intervals.

There are a number of cases. If $i = j$, then the solution has value 1, since c_i is a palindrome by itself. If $i = j - 1$ then the optimal solution is 1 if $c_i \neq c_j$ and 2 if $c_i = c_j$. If $i < j - 1$ and $c_i = c_j$, then the optimal solution must match c_i to c_j, so the optimal solution has value $p(i, j) = p(i+1, j-1)+2$. If $i < j-1$ and $c_i \neq c_j$, then the optimal solution does not involve either c_i or c_j so it is equal to either $p(i + 1, j)$ or $p(i, j - 1)$.

We can compute the $p(i, j)$ values in increasing value of $|j-i|$. Putting all this together gives the algorithm, which computes the longest palindrome as $P(i, j)$ for each interval $[i, j]$, and the length of the palindrome as $p(i, j)$.

```
Input: A list c[1,...,n] of characters.
Result: The longest palindrome subsequence of c.
for j = 1 to n do
|  Set p(j,j) = 1, P(j,j) = c_j;
end
for j = 2 to n do
|  if c_j = c_{j-1} then
|  |  Set p(j-1,j) = 2, P(j,j) = c_{j-1}c_j;
|  end
|  else
|  |  Set p(j-1,j) = 1, P(j,j) = c_{j-1};
|  end
end
for k = 2 to n do
|  for i = 1 to n-k do
|  |  if c_i = c_{i+k} then
|  |  |  Set p(i,i+k) = 2 + p(i+1,i+k-1);
|  |  |  Set P(i,i+k) = c_i P(i+1,i+k-1) c_{i+k};
|  |  end
|  |  else
|  |  |  if p(i+1,i+k) > p(i,i+k-1) then
|  |  |  |  Set p(i,i+k) = p(i+1,i+k);
|  |  |  |  Set P(i,i+k) = P(i+1,i+k);
|  |  |  end
|  |  |  else
|  |  |  |  Set p(i,i+k) = p(i,i+k-1);
|  |  |  |  Set P(i,i+k) = P(i,i+k-1);
|  |  |  end
|  |  end
|  end
end
return P(1,n);
```

Runtime: The algorithm's runtime is proportional to the number of subproblems $P(i, j)$, which is $O(n^2)$.

3. You are given a rectangular piece of cloth with dimensions $X \times Y$, where X and Y are positive

integers, and a list of n products that can be made using the cloth. For each product i you know that a rectangle of cloth of dimensions $a_i \times b_i$ is needed and that the selling price of the product is c_i Assume the a_i, b_i and c_i are all positive integers. You have a machine that can cut any rectangular piece of cloth into two pieces either horizontally or vertically. Design an algorithm that runs in time that is polynomial in X, Y, n and determines the best return on the $X \times Y$ piece of cloth, that is, a strategy for cutting the cloth so that the products made from the resulting pieces give the maximum sum of selling prices. You are free to make as many copies of a given product as you wish, or none, if desired.

Solution. The crux of this problem is to identify precisely which actions are available to the machine:

- Make a vertical cut
- Make a horizontal cut
- Do nothing (and sell the current item)

Input: Dimensions of cloth X,Y, and a list of item values and dimensions.
Result: Best possible value of the cloth
Let cut be an X by Y dimensional array with every entry initialized to 0.
for $x \in [0, X-1]$ **do**
  for $y \in [0, Y-1]$ **do**
    for $x_{cut} \in [1, x-1]$ **do**
      $cut[x,y] = max(cut[x,y], cut[x_{cut}, y] + cut[x - x_{cut}, y])$
    end
    for $y_{cut} \in [1, y-1]$ **do**
      $cut[x,y] = max(cut[x,y], cut[x, y_{cut}] + cut[x, y - y_{cut}])$
    end
    for $item \in Items$ **do**
      if $item_{dimensions} == (x,y)$ **then**
        $cut[x,y] = max(cut[x,y], item_{value})$
      end
    end
  end
end
return $cut[X-1, Y-1]$
// Note: This does not actually retrieve the necessary cuts. The cuts could be retrieved by storing which actions are taken along the way, and storing those actions along side their corresponding values in cut.

Run time: The outer two loops lead to $O(XY)$ iterations over the inner most piece, which does tries every possible vertical cut, horizontal cut, and item. The overall runtime is $O(XY) \cdot O(X+Y+n) = O(XY(X+Y+n))$.

Proof of correctness: We have to prove that $OPT(x,y) = cut(x,y)$. Here, OPT refers to the optimum solution to the problem and cut refers to the solution returned by the above algorithm. It is sufficient to prove

$$OPT(x,y) \geq cut(x,y) \tag{1}$$
$$OPT(x,y) \leq cut(x,y) \tag{2}$$

To prove equation (1), we use the fact that the solution returned by $cut(x,y)$ is a feasible solution and hence $OPT(x,y)$ can only do better, impying $OPT(x,y) \geq cut(x,y)$.
We prove equation 2 by induction on the size of xy.
Base Case: $(x,y) = (1,1)$. It is clear here that $OPT(1,1)$ could be 0 or the maximum price given by a product of dimension 1×1. In both cases, $OPT(1,1) = cut(1,1)$.
Induction Hypothesis: $OPT(x',y') \leq cut(x',y')\ \forall x' \leq x, y' \leq y$.
To prove: $OPT(x+1,y) \leq cut(x+1,y)$. Let us consider the optimum solution. It is true that there exist an i such that the piece given by dimensions $(x+1) \times y$ is cut horizontally or vertically. This says that $OPT(x+1,y) = OPT(i,y) + OPT(x+1-i,y)$(when cut horizontally) or $OPT(x+1,y) = OPT(x+1,i) + OPT(x+1,y-i)$(when cut vertically). By induction hypothesis $OPT(x',y') \leq cut(x',y')$ for all $x' \leq x$ and $y' \leq y$. This implies $OPT(x+1,y) \leq cut(x+1,y)$. A similar argument would give $OPT(x,y+1) \leq cut(x,y+1)$. This completes the proof.

4. Say you have access to a function dict that returns true if its input is a valid English word, and false otherwise. We are given as input a sentence from which the punctuation has been stripped (for example: "dynamicprogrammingisfabulous"). Assuming calls to dict take unit time, give an $O(n^2)$ time algorithm to figure out whether an input string of length n can be split into a sequence of valid words or not.

Solution: Let the input have the characters $x_1, \ldots, x_n$.

Let $M[j]$ be set to true if the first j characters can be split into valid words, and false otherwise.

Then we have $M[j]$ is true if and only if there is some $i < j$ such that $x_i, \ldots, x_j$ is a valid word, and $M[i-1]$ is true. So, we can compute $M[j]$ iteratively for all j:

So, the algorithm is:

(a) Set $M[0]$ to be false.
(b) For $j = 1, 2, \ldots, n$
    i. For $i = 1, 2, \ldots, j$
        A. If dict$(x_i, \ldots, x_j)$ returns true and $M[i-1]$ is true, set $M[j]$ to be true. Otherwise set $M[j]$ to be false.

The algorithm consists of two for loops, each of which can iterate at most n times. So the running time is $O(n^2)$.