CSEP521: Algorithms November 23, 2022

Homework 3

Anup Rao Due: December 4, 2022

Read the fine print[1]. An algorithm is said to run in polynomial time if it runs in time $O(n^d)$ for some constant d on inputs of size n. Each problem is worth 10 points:

1. 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

[1] In solving the problem sets, you are allowed to collaborate with fellow students taking the class, but **each submission can have at most one author**. If you do collaborate in any way, you must acknowledge, for each problem, the people you worked with on that problem. The problems have been carefully chosen for their pedagogical value, and hence might be similar to those given in past offerings of this course at UW, or similar to other courses at other schools. Using any pre-existing solutions from these sources, for from the web, constitutes a violation of the academic integrity you are expected to exemplify, and is strictly prohibited. Most of the problems only require one or two key ideas for their solution. It will help you a lot to spell out these main ideas so that you can get most of the credit for a problem even if you err on the finer details. Please justify all answers. Some other guidelines for writing good solutions are here: `http://www.cs.washington.edu/education/courses/cse421/08wi/guidelines.pdf`.

the length of the palindrome as $p(i, j)$.

Input: A list $c[1, \ldots, n]$ 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 $O(n^2)$.

2. 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.

3. Give an algorithm to find a vertex cover of smallest size in a bipartite graph. Hints:

   (a) Construct a flow network from the input bipartite graph just as in the maximum matching algorithm.

   (b) Show that every min-cut in this flow network gives a vertex cover whose size is the same as the capacity of the cut.

   (c) Show that every minimum sized vertex cover in the bipartite graph gives a cut whose capacity is the same as the size of the vertex cover.

   (d) Conclude by giving an algorithm to find the smallest vertex cover.

   Solution: The flow network will have vertices s, t as well as all the vertices of the bipartite graph. For every edge u on the left, the network has an edge (s, u) with capacity 1. For every edge v on the right, the network has the edge (v, t) of capacity 1 for every edge (u, v) from left to right, the network has the edge (u, v) with capacity ∞.

   Now, as in the hint, we prove that every min-cut X, Y corresponds to a vertex cover $(A - X) \cup (B - Y)$ of smallest possible size. To see this, first we prove that given any vertex cover V of the original graph, we obtain an s, t cut in the flow network whose capacity is $|V|$. The s, t cut is given by the partition

   $$\{s\} \cup (A - V) \cup (B \cap V), \{t\} \cup (A \cap V) \cup (B - V).$$

   This is clearly a valid partition of the vertices of the flow network. The fact that V is a vertex cover implies that no edge of infinite capacity is cut. Such an edge would have to go from $A - V$ to $B - V$, and there are no such edges because V is a vertex cover. So, the capacity of this cut is finite. Moreover the number of edges from s that are cut is exactly $|A \cap V|$, and the number of edges into t that are cut is exactly $|B \cap V|$. So, the capacity of the cut is $|A \cap V| + |B \cap V| = |V|$. This proves that every vertex cover gives a cut whose capacity is the size of the vertex cover.

   Finally, we show that every min-cut gives a vertex cover whose size is the capacity of the cut. Given any cut (X, Y) consider the set of vertices $(A - X) \cup (B - Y)$. We claim that this is a vertex cover. Indeed, since this is a min-cut, it cannot cut any edge of infinite capacity. This means that for every edge (u, v) from A to B, either $u \notin X$ or $v \notin Y$. Thus, (u, v) is covered by $(A - X) \cup (B - Y)$. The size of this vertex cover is $|A - X| + |B - Y|$, which exactly the

same as the capacity of the cut (X, Y). So, the capacity of the min-cut is the same as the size of the smallest vertex cover.

The final algorithm is to run the capacity scaling algorithm on this flow network.

4. You are running a truck business and need to fill a truck that can carry a total weight of 100 tons and volume 30 cubic meters. You can put three types of materials into the truck.

   (a) Item 1 has density 2 tons per cubic meter, maximum available amount is 40 cubic meters and the revenue associated with it is 1000 dollars per cubic meter.

   (b) Item 2 has density 5 tons per cubic meter, maximum available amount is 20 cubic meters and the revenue associated with it is 2000 dollars per cubic meter.

   (c) Item 3 has density 7 tons per cubic meter, maximum available amount is 15 cubic meters and the revenue associated with it is 1500 dollars per cubic meter.

   Write a linear program to calculate how much of each amount the truck should carry to maximize profits (no need to solve it).

   Solution: In the following linear program, a, b, c denote the volume of each of the materials that can be carried.

$$\begin{aligned} &\text{maximize } 1000a + 2000b + 1500c \\ &\text{subject to} \\ 2a + 5b + 7c &\leq 100 \\ a &\leq 40 \\ b &\leq 20 \\ c &\leq 15 \\ a + b + c &\leq 30 \\ a, b, c &\geq 0 \end{aligned}$$