CSE 6331 HOMEWORK 5

BRENDAN WHITAKER

1. *Consider the single-pair shortest path problem. We solved the problem using the forward approach. Now, solve it using the backward approach. Your answer must include: the definition of $f(x)$, a recurrence, boundary conditions, and the goal.*

   Recall that the problem is to find the shortest path from node u to node v in a directed acyclic graph. The graph $G = (V, E)$ is represented by a matrix:

$$d(i,j) = \begin{cases} length(i,j) & (i,j) \in E \\ 0 & i = j \\ \infty & otherwise \end{cases}.$$

   Define $f(x)$ to be the shortest distance from node u to node x. Then $f(x)$ is given by the recurrence:

$$f(x) = \min \{ \, d(y,x) + f(y) : indegree(y) \neq 0, y \neq u \, \}.$$

   Our boundary conditions are:

$$f(x) = \begin{cases} \infty & x \neq u, indegree(x) = 0 \\ 0 & x = u \end{cases}.$$

   And the goal is to compute $f(v)$.

2. *Implement the third approach of dynamic programming to the longest common subsequence problem. Your algorithm needs to print the actual longest common subsequence. Specifically, write two procedures: (1) a non-recursive procedure to compute $L(i,j), 1 \leq i, j \neq n$, and (2) a recursive procedure Longest(i,j) such that Longest$(1,1)$ will print the longest common subsequence.*

Algorithm 1: Compute-Array-L

Data: Two sequences:$A = (a_1, ..., a_n), B = (b_1, ..., b_n)$.
Result: The array $L(i,j)$ computed for $1 \leq i, j \leq n+1$.
begin
    global array $L[1..n+1, 1..n+1], \varphi[1..n, 1..n]$;
    initialize $L[i, n+1] \longleftarrow, L[n+1, j] \longleftarrow 0$ for $1 \leq i, j \leq n+1$;
    for $i \leftarrow n$ **to** 1 **do**
        for $j \leftarrow n$ **to** 1 **do**
            if $a_i \neq b_j$ **then**
                $L[i,j] \longleftarrow \max \{ \, L[i+1, j], L[i, j+1] \, \}$;
            else
                $L[i,j] \longleftarrow 1 + L[i+1, j+1]$;
            end
        end
    end
end

Date: SP18.

Algorithm 2: Longest(i, j)

Data: The computed array $L[1..n+1, 1..n+1]$.
Result: Prints the longest common subsequence of A and B.

begin
  `/*` Assume $L[1..n+1, 1..n+1]$ has already been computed. `*/`
  if $L[i, j] = 1 + L[i+1, j+1]$ **then**
    print a_i;
    Longest($i+1, j+1$);
  else if $L[i, j] = L[i+1, j]$ **then**
    Longest($i+1, j$);
  else
    Longest($i, j+1$);
  end
end

3. *Consider the all-pair shortest paths problem. Suppose global arrays* $D^k[1..n, 1..n]$ *and* $P^k[1..n, 1..n]$ $1 \leq k \leq n$, *have been computed as in the straightforward implementation. Write a recursive procedure Path*(k, i, j) *such that a call to Path*(n, i, j) *will print the shortest path from* i *to* j. *Note: a path is a sequence of vertices. You may print a vertex more than once (e.g., it is OK to print a path* (a, b, c, d) *as* (a, a, b, c, c, c, d).*)*

Algorithm 3: Path(i, j)

Data: The pre-computed arrays $D[1..n, 1..n]$ and $P[1..n, 1..n]$.
Result: Prints the shortest path from i to j.

begin
  `/*` Assume $D[1..n, 1..n]$ and $P[1..n, 1..n]$ have already been computed. `*/`
  if $P[i, j] = 0$ **then**
    return;
  else
    print i;
    if $i = j$ **then**
      return;
    else
      Path($P[i, j], j$);
    end
  end
end