Yaqzan Ali

250 644 709

1)

```
PRINT-LCS (c, X, i, j)
If (i = 0 or j = 0)
        Return
If (c[i,j] = c[i-1, j-1])
        PRINT-LCS(c, X, i-1, j-1)
        Print xi
Else if ( c[i-1, j] ≥ c[I, j-1] )
        PRINT-LCS(c, X, i-1, j)
Else
        PRINT-LCS(c, X, i, j-1)
```

2)

Let $i_1, \ldots, i_n$ be the items with values $v_1, \ldots, v_n$ and $w_1, \ldots, w_n$ be their weights. Let W be the maximum knapsack weight:

1) $w_1 \leq w_2 \leq \ldots \leq w_n$
2) $v_1 \geq v_2 \geq \ldots \geq n_n$

The Algorithm:

```
w = 0
S = null
For (i = 1 ; i ≤ n ; i++){
        If (w + wi ≤ W){
                w += wi
                S = S U {i}
        }
}
```

Proof:

For the optimal packing of S with $i_1 \in S$, the packing of $S^n = (S \setminus i_k)$, is optimal for the items

$I_2, \ldots, i_n$ and $W_n = W - w_1$. Indeed, if S^n is not optimal, we can improve the original packing S by improving S^n.

3)

Sort the list of points so that $x_1 \leq . . . \leq x_n$
While (list is not empty)

Pick the leftmost point in the list, x_1

Take the Interval $[x_1, x_1 + 1]$

Remove all points $x_1, x_1 + 1, . . . , x_{1+k}$, such that $x_{1+k} - x_1 \leq 1$ from the list

Proof:

Assume, by contradiction, that when looking at the leftmost point x_1, we selected an interval

$[x_1 - a, x_1 + b]$, such that $a \geq 0$, $b \geq 0$, and $b - a = 1$. Since x_1 is the leftmost point in the set of points, we know that we will not reach any points in in the interval $[x_1 - a, x_1)$. Therefore, we'd be able to maximize the total number of possible points covered iff we select the interval

$[x_1 - a, x_1 + b]$, assuming that $a = 0$, and $b = 1$.

4)

If we were to look at the possible source files S using n bits and using compressed files E (using n bits), we'd see that it is $2^{n+1} - 1$. Since all compression algorithm must assign each element $s \in S$ to a distinct element $e \in E$, the algorithm cannot possibly compress the source file.

5)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	B	A	B	B	A	B	B	A	B	B	A	B	A	B	B	A	B	B
next	0	0	1	1	2	3	4	5	6	7	8	9	0	1	1	2	3	4

6)

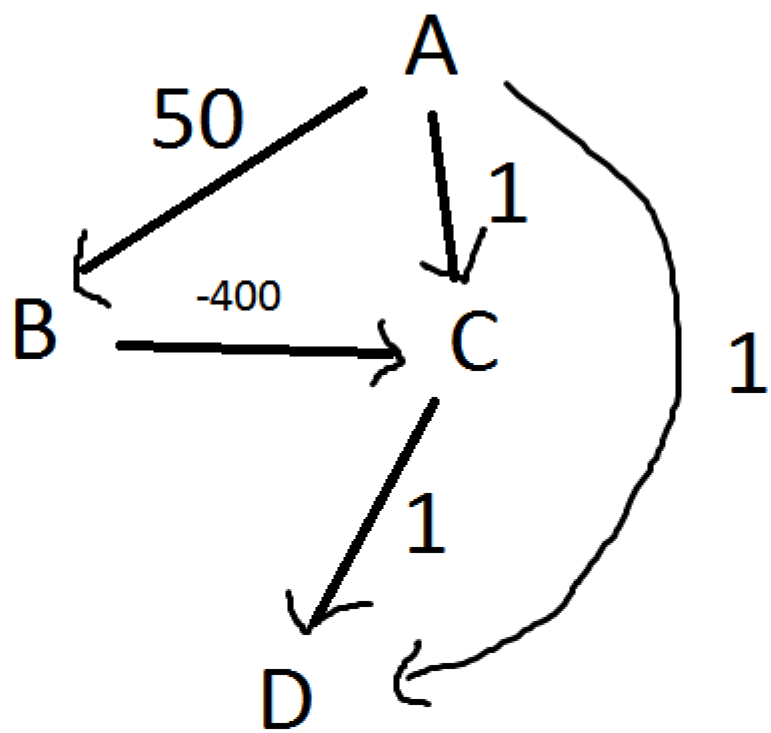
Input: Strings T[0…n] of P[0…n]
Output: Starting index of substring of T matching P

```
F ← compute failure function of Pattern P
i ← 0
j ← 0
While i < length[T] do
        If j ← m – 1
                Return i – m+1
                i ← i + 1
                j ← j + 1
        else if (j > 0)
                j ← f(j - 1)
        else
                i ← i + 1
end while
```

7) Algorithm

Put all edges in a heap, put each vertex in a set by itself
While (not found a MST yet) do begin
        Delete max edge, {u,v} from heap;
        If (u and v are not in the same set)
                Mark {u,v} as tree edge;
                Union sets containing u and v
        End if
End while

8)



First, we set A to 0, and all others to infinity. Expanding node A, we set C to 1, B to 50, and D to 1. We expand C and D which give no effect. Then we expand B. This changes C to -350. However, D is still set to 1, despite the shortest path to it being -349. Thus the algorithm fails to accurately compute.

9)

All-Pair-shortest-path is still correct in this case.

Let $d_k(i,j)$ = shortest path from i to j involving o k edges.

$d_1(i,j)$ = is original weight matrix.

Compute d_{k+1}'s from d_k's by seeing if adding edge helps:

$$d_{k+1}(i,j) = \text{Min}\ \{ d_k(i,m) + d_1(m,j) \}$$

Assuming that there are no negative cycles, $d_{V-1}(i,j)$ is the solution.

10)

We use Dijkstra's algorithm the shortest paths to all the vertices, and then use the distances to calculate the shortest cycle.

Apply Dijkstra's algorithm

Min weight = infinity

For each $u \in V[G]$

    For each $w \in Adj[u]$

        If (w = v && min-weight < d[u] + weight(u,w)

            Min-weight = d[u] + weight(u,w)

Return min-weight