Algorithmics II (H) Exam April / May 2022 – Solutions

1. Create a point set P where each rectangle is represented by its lower-left hand and upper-right hand corner coordinates (x_1,y_1) and (x_2,y_2). Ensure that each "rectangle" point is marked L or R according to whether it is a lower-left hand or upper-right hand corner. Also ensure that each "rectangle" point marked L stores the y-coordinate y_2 of its corresponding upper-right hand corner alongside it. For each rectangle point p marked R, store a pointer to the corresponding point marked L from the same rectangle as p (call this p's *partner*).

Also in P, represent each vertical line segment with endpoints (x,y_1) to (x,y_2), where $y_1 \leq y_2$, by (x,y_1), storing the integer y_2 alongside it.

Sort all the stored points in increasing order of x-coordinate, breaking ties so that rectangle points marked L come before vertical line points, which come before rectangle points marked R.

Maintain an AVL tree T of "candidate" points, ordered on their y-coordinate, which is initially empty.

Now scan through the list of points sorted on x-coordinate. For each rectangle point p encountered marked L, add p to T, whilst for each rectangle point p encountered marked R, remove p's partner from T. For each vertical point p encountered, let $y_1..y_2$ be its range and range search T, constructing a list S of all candidates with y-coordinate in the range $y_1..y_2$.

Let p be the candidate in S with the largest y-coordinate, let y_3 be the y-coordinate of p and let y_4 be the y-coordinate of the corresponding upper-right hand corner point of p. If $y_4 \leq y_2$ then $y_1 \leq y_3 \leq y_4 \leq y_2$, so add $|S|$ to the count of rectangles that fully intersect vertical line segments, otherwise add $|S|$-1 to the count.

2. **(a)**

$$D_0=\begin{pmatrix}0 & \infty & 4 & 5\\ 2 & 0 & 7 & 8\\ \infty & \infty & 0 & -1\\ \infty & \infty & \infty & 0\end{pmatrix}\quad D_1=\begin{pmatrix}0 & \infty & 4 & 5\\ 2 & 0 & 6 & 7\\ \infty & \infty & 0 & -1\\ \infty & \infty & \infty & 0\end{pmatrix}=D_2\quad D_3=\begin{pmatrix}0 & \infty & 4 & 3\\ 2 & 0 & 6 & 5\\ \infty & \infty & 0 & -1\\ \infty & \infty & \infty & 0\end{pmatrix}=D_4$$

The length of a shortest path from v_i to v_j is given by $D_4(i,j)$.

(b) Possible pseudocode for the process is as follows:

```
if (Πn(i,j)==null)
   output "no path from vi to vj exists";
else
{  S=⟨j⟩;
   while (i≠j)
   {  S=⟨Πn(i,j)⟩++S;
      j=Πn(i,j);
   }
   output S;
}
```

(c)

$$\Pi_0=\begin{pmatrix}- & - & 1 & 1\\ 2 & - & 2 & 2\\ - & - & - & 3\\ - & - & - & -\end{pmatrix}\quad \Pi_1=\begin{pmatrix}- & - & 1 & 1\\ 2 & - & 1 & 1\\ - & - & - & 3\\ - & - & - & -\end{pmatrix}=\Pi_2\quad \Pi_3=\begin{pmatrix}- & - & 1 & 3\\ 2 & - & 1 & 3\\ - & - & - & 3\\ - & - & - & -\end{pmatrix}=\Pi_4$$

"–" refers to **null**. for each of Π_0, Π_1 and Π_3.

3. **(a)** This ensures that no suffix of $S\$$ is a prefix of another suffix of $S\$$, and hence every suffix of S is represented by a unique leaf node in the suffix tree for $S\$$.

(b) **(i)** During a traversal of T, we can identify all the branch nodes of T having maximum string depth d. At any such branch node b of T, all of b's children must be leaf nodes. Scan through the suffix numbers of these leaf nodes, letting k_1 and k_2 be the minimum and maximum suffix numbers encountered. If $k_1+d \leq k_2$ then we have found a longest repeated substring with non-overlapping embeddings, namely $S(k_1..k_1+d-1)$ and $S(k_2..k_2+d-1)$. Otherwise, b has no instances of longest repeated substrings with non-overlapping embeddings, so consider the next branch node with maximum string depth until all are exhausted, in which case no longest repeated substring with non-overlapping embeddings exists.

(b) **(ii)** Clearly the branch nodes with maximum string depth can be identified in $O(n)$ time. The total work done at all branch nodes with maximum string depth taken together is linear in the number of leaf nodes of S in the worst case, and therefore $O(n)$ overall.

(c) At every leaf node v, set $\min_v$ and $\max_v$ to be the suffix number of v. At every branch node v, set $\min_v$ to be min $\{\min_w : w \text{ is a child of } v\}$ and set $\max_v$ to be max $\{\max_w : w \text{ is a child of } v\}$. Then traverse T, searching for a branch node v whose string depth d is maximum such that $\min_v + d \leq \max_v$. If such a v exists, output the path label of v, otherwise output "none exists".

4. **(a)** **(i)** The dynamic programming table is as follows:

i	w_i	p_i \ j	0	1	2	3	4	5	6
0			0	0	0	0	0	0	0
1	3	10	0	0	0	10	10	10	10
2	1	6	0	6	6	10	16	16	16
3	5	18	0	6	6	10	16	18	24
4	2	9	0	6	9	15	16	19	25

(ii) The knapsack capacity might be large (e.g., $C=2^n$) in which case the complexity of the algorithm is no better than $O(n.2^n)$, which is not polynomial in the size of the problem instance.

(b) Consider, for example, the following items: 5, 3, 3, 3, 2, 2 and bin capacity 9. FFD will use 3 bins: bin 1 will contains items of sizes 5, 3; bin 2 will contain items of sizes 3, 3, 2; whilst bin 3 will contain an item of size 2. An optimal packing uses 2 bins: bin 1 will contain items of sizes 5, 2, 2; whilst bin 2 will contain items of sizes 3, 3, 3.