Algorithmics II (H) Exam April / May 2024 – Solutions

1. **(a)** Assume that each triangle $t \in S$ is represented by three points, $t.p1$, $t.p2$ and $t.p3$. From these points, we can instantiate the horizontal, vertical and diagonal line segment objects that make up the sides of t. For example, if $t.p1.x = t.p2.x$ then the vertical line segment of t has endpoints $(t.p1.x, t.p1.y)$ and $(t.p2.x, t.p2.y)$. Ensure that each line segment has a label indicating which triangle it comes from.

Two triangles in S intersect if and only if there is an intersection between two line segments from different triangles that belong to the following types:

- Horizontal and vertical
- Horizontal and diagonal (SW-NE or NW-SE)
- Vertical and diagonal (SW-NE or NW-SE)

To determine whether there is an intersection between a horizontal line segment and a vertical line segment from two different triangles, use the standard line sweep algorithm from Lecture 4. For the case involving a vertical line segment and a SW-NE line segment, use the process outlined in the solution to Question 14 from the Geometric Algorithms tutorial sheet. The other cases involving diagonal line segments can be dealt with in a similar way. If an intersection of line segments from two different triangles is encountered, output “yes” and halt. If we reach the end of the process, output “no”.

(b) Use of the line sweep algorithm for intersections between line segments of different types has $O(N \log N + p)$ complexity, where N is the total number of line segments considered and p is the number of intersections. $N \leq 2n$, since each execution of the line sweep algorithm involves two sides of each triangle. Also $p \leq n+1$, since at most one intersection from two line segments belonging to the same triangle will be encountered before one may be encountered between two different triangles, in which case the algorithm terminates immediately. Finally, this algorithm is used at most 5 times.

2. **(a)** Various solutions are possible here, though of course the final flow value will be the same. One possible sequence of augmentations is as follows:

- Augment by 1 along path $s \rightarrow v_1 \rightarrow v_4 \rightarrow t$; flow now has value 6;
- Augment by 1 along path $s \rightarrow v_3 \rightarrow v_6 \rightarrow v_7 \rightarrow t$; flow now has value 7;
- Augment by 2 along path $s \rightarrow v_3 \rightarrow v_5 \rightarrow v_8 \rightarrow v_6 \rightarrow v_7 \rightarrow t$; flow now has value 9;
- Augment by 1 along path $s \rightarrow v_3 \rightarrow v_5 \rightarrow v_8 \leftarrow v_6 \rightarrow v_7 \rightarrow t$; flow now has value 10.

Note 1: the last augmentation involves a 'backward' edge, and this backward edge will have to be used at some point to achieve a maximum flow.

Note 2: this maximum flow is not (quite) unique. The flow along edges (s,v_3) and (v_3,v_5) can be reduced by 1, with a corresponding increase in the flow along edges (s,v_2) and (v_2,v_5).

(b) One cut is (s, v_1), (v_2, v_4), (v_3, v_6), (v_5, v_4), (v_5, v_8) with capacity 2 + 2 + 2 + 1 + 3 = 10. Another possible cut is (v_7, t), (v_8, t), (v_1, v_4), (v_5, v_4), (v_2, v_4) with capacity 5 + 1 + 1 + 1 + 2 = 10.

(c) Clearly f is a valid flow in H', so $\text{val}(f') \geq \text{val}(f)$. By the Max Flow-Min Cut theorem, $\text{val}(f') = \text{cap}(C')$ and $\text{val}(f) = \text{cap}(C)$, where C' and C are minimum cuts in H' and H respectively. But C is a cut in H' whose capacity in H' is at most 1 more than its capacity in H. Hence $\text{val}(f') = \text{cap}(C') \leq \text{cap}(C) + 1 = \text{val}(f) + 1$.

3. **(a)** The dynamic programming table after the recursion terminates is as follows:

		0	1	2	3	4	5	6
			C	A	T	G	C	G
0		-	*	-	*	*	-	-
1	A	*	-	*	*	*	-	-
2	C	-	*	*	*	*	*	-
3	T	*	*	*	*	*	*	-
4	G	*	*	*	*	*	*	-
5	G	-	-	-	-	*	-	*
6	C	-	-	-	-	-	*	*

The entries that are *not* computed are thus:
(0,0), (0,2), (0,5), (0,6)
(1,1), (1,5), (1,6)
(2,0), (2,6)
(3,6)
(4,6)
(5,0),…,(5,3), (5,5)
(6,0),…,(6,4)

Note: the above table was *not* required for the solution, only the set of (i, j) pairs. The table is just included here for illustration purposes.

(b) The following list indicates the values of x corresponding to each position of T:

pos	*char*	x
1	b	{0, 1, 3, 4, 5, 6, 7, 10, 13}
2	c	{0, 1, 3, 8}
3	d	{0, 1, 3}
4	a	{0, 1, 2, 3, 5, 6, 7, 10, 13}
5	b	{0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13}
6	c	{0, 1, 3, 8}
7	a	{0, 1, 2, 3, 5, 6, 7, 9, 10, 12, 13}
8	d	{0, 1, 3, 14}

4. (a) $\varepsilon=2$ and $n=4$. Thus $\delta=\varepsilon/2n=1/4$ and thus $1-\delta=3/4$. (Not required for solution.) The contents of the set `s` after each iteration of the main loop are as follows:

`s` = {0, 3} after the first iteration
`s` = {0, 3, 7, 10} after the second iteration
`s` = {0, 3, 7, 10, 13*, 16, 20*} after the third iteration
`s` = {0, 3, 7, 10, 15, 16*, 18*} after the fourth iteration

Elements marked * are trimmed from the set.

(b) The ptas outputs 15 and the optimal measure is 20.

(c) Add the following lines of code to the ptas:

```
k = max s[n];
j = n;
while (j > 0)
{  if (k does not belong to s[j-1])
   {  output x_j;
      k -= x_j;
   }
   j--;
}
```