· U C ·

UNIVERSIDADE DE COIMBRA
Faculty of Science and Technology
Department of Informatics Engineering

Laboratório de Programação Avançada

Retake Exam – July 5 2017

Name: ________________________ Student ID: ________________

8 grade points in total, 3 hours, closed books.

1. Give the time complexity of the following algorithm that computes the minimum absolute difference between two integers in a list and justify your answer using the Master Theorem. Assume that $S = (S[1], \ldots, S[n])$ is a non-ordered list of n distinct integers. In addition, assume that all arithmetic operations as well as function *abs* (absolute value) take constant amount of time, and that the following functions take linear amount of time with respect to the number of elements in S: $\min(S)$ (minimum value), $\max(S)$ (maximum value), $median(S)$ (index of the median value in S), $extract_{\leq}(S,k)$ e $extract_{>}(S,k)$ (it returns the elements in S that are smaller that or equal to $S[k]$, and larger than $S[k]$, respectively). (1 g.p.)

Function $ClosestPair(S)$
  $m = |S|$
  if $m = 1$ **then**
    $d = \infty$
  else if $m = 2$ **then**
    $d = abs(S[2] - S[1])$
  else
    $k = median(S)$
    $S_1 = extract_{\leq}(S,k)$
    $S_2 = extract_{>}(S,k)$
    $d_1 = ClosestPair(S_1)$
    $d_2 = ClosestPair(S_2)$
    $d = \min(d_1, d_2, \min(S_2) - \max(S_1))$
  return d

Master Theorem (general version):
Let $a \geq 1$, $b > 1$, $d \geq 0$.

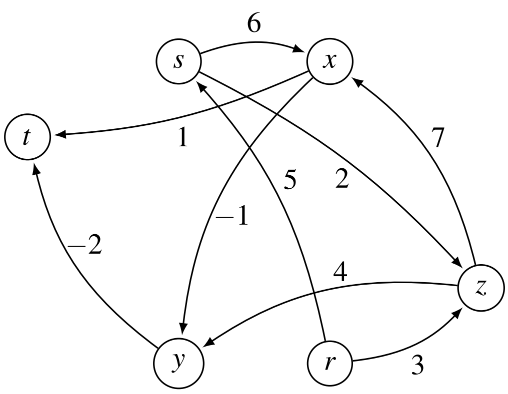
$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases} \Rightarrow$$

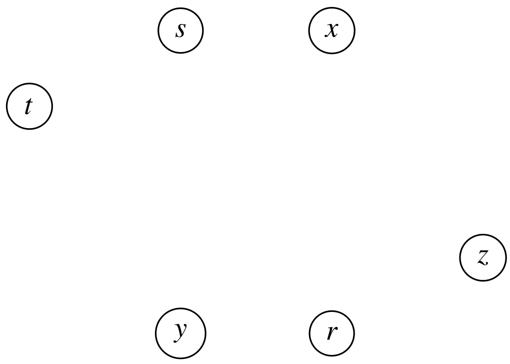
$$T(n) = \begin{cases} \Theta(n^c) & \text{if } \log_b a < c \\ \Theta(n^c \log n) & \text{if } \log_b a = c \\ \Theta(n^{\log_b a}) & \text{if } \log_b a > c \end{cases}$$

2. Consider the problem of finding a shortest path from vertex s to vertex t in a graph $G = (V, A)$, where V is the set of vertices, A is the set of edges and $w(u, v)$ denotes the distance between a vertex u and a vertex v, $(u, v) \in A$. If the graph is acyclic, then it is possible to find the value of the shortest path between s and t in time $O(|V| + |E|)$ with the following approach.

   1. Let $d(s) = 0$ and let $d(v) = \infty$, for all vertices $v \in V \setminus \{s\}$
   2. Sort the vertices in V with respect to their topological ordering
   3. For each vertex $u \in V$, following the topological ordering
      3.1 For each vertex v such that $(u, v) \in A$
         3.1.1 If $d(v) > d(u) + w(u, v)$, then $d(v) = d(u) + w(u, v)$
   4. Returns $d(t)$

   a) Find the shortest path between vertex s and vertex t in the following graph (left), based on the algorithm above. For the step 2, show the ordering of the vertices with respect to the topological ordering in the box below. In step 3, draw the arcs that belong to the shortest path as well as the final values of d at each vertex in the graph to your right. (1.5 g.p.)

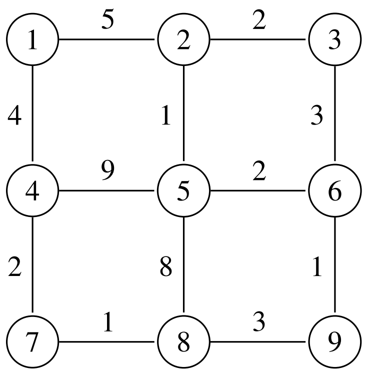




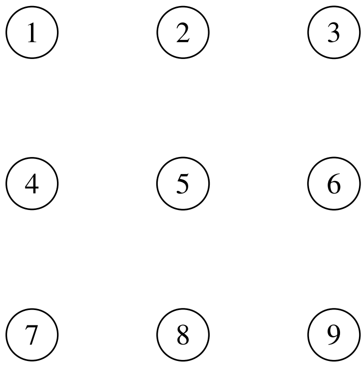
Topological ordering of the vertices:

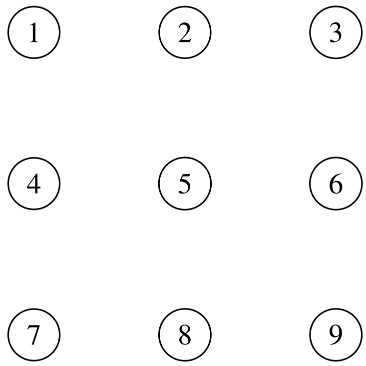
   b) Show that step 3 is in fact a dynamic programming approach by using the argument of optimal substructure. (1 g.p.)

3. Consider the following graph.

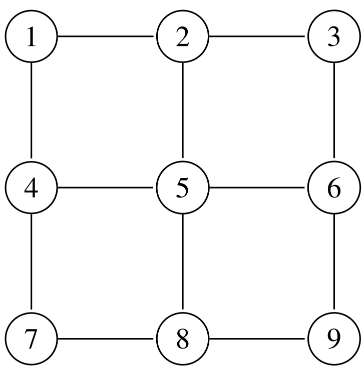


Draw its minimum spanning tree (left) as well as the graph of the union-find data structure (right), without path compression, using Kruskal algorithm. Always connect the root of the tree with the smallest height to the root of the tree with largest height and, in case of a tie, choose as root the node with the smallest label. (1.5 pontos)





4. Given a directed graph $G = (V, A)$, where V is the vertex set, A is the arc set, an *arborescence* rooted in a vertex $r \in V$ is a subgraph of G for which there exists a path between r and every vertex $v \in V \setminus \{r\}$. However, not all directed graphs contain an arborescence. Build an example of a directed graph that does not contain any arborescence by providing the direction of the edges in the graph below. Briefly explain the reasoning of your example in the box below. (1 g.p.)



5. A *cut* of a string consists of splitting that string into two non-empty substrings. Given a string $s = s_1 \dots s_n$, the goal is to compute the least number of cuts in s such that each resulting substring is a palindrome (a palindrome is a string that can be read from left to right or from right to left). For example, the least number of cuts for the string "ananas" is one: "anana", "s". It is possible to compute the least number of cuts with the following recurrence:

$$C(s,i,j) = \begin{cases} 0 & \text{if } s_i \dots s_j \text{ is a palindrome} \\ \min\limits_{i \le k < j} \{C(s,i,k) + 1 + C(s,k+1,j)\} & \text{otherwise} \end{cases}$$

a) Based on this recorrence, give the pseudo-code of a top-down dynamic programming algorithm that solves the problem. Assume that there exists a function $Palindrome(s')$ that returns *True* if the (sub)string s' is a palindrome or *False* otherwise. (1 g.p.)

b) Give the pseudo-code of the bottom-up version and discuss its time complexity (1 g.p.).