Midterm var2

MIDTERM, 2022-04-29

Question 1: Consider the following algorithm which runs on a (zero-based) array `A[]`. The length of the array is n; the elements of array are non-negative integers smaller than 2^m (i.e. `unsigned` type using m bits each).

The initial call is COMPUTESOMETHING($\texttt{A}[], n$); after that the function calls itself recursively. This function calls GCD(x, y) or GCD(x, y, z) (the *greatest common divisor* of two or three numbers). This function uses Euclidean algorithm; you may assume that its time complexity is $O(\max(|x|, |y|))$ or $O(\max(|x|, |y|, |z|))$ respectively, where $|x|$ denotes the length of argument x (in bits).

COMPUTESOMETHING($\texttt{A}[], n$)
  if $n == 1$:
    return $A[0]$
  else if $n == 2$:
    return GCD($A[0], A[1]$)
  else if $n == 3$:
    return GCD($A[0], A[1], A[2]$)
  else if $n \equiv 0 \pmod 5$:
    return GCD($A[n-1], A[n-2], A[n-3]$) + COMPUTESOMETHING($\texttt{A}[], n-3$)
  else:
    return GCD($A[n-1], A[n-2]$) + COMPUTESOMETHING($\texttt{A}[], n-2$)

(A) Denote by $T(n, m)$ the time complexity of this algorithm on an array with n elements of size m. Express $T(n, m)$ in terms of $T(\ldots)$, where one or both arguments are smaller to reflect how the time complexity of the recursive call affects the time complexity of the overall algorithm.

(B) Find some $g(n, m)$ such that $T(n)$ is in $O(g(n))$ and justify your answer. If multiple asymptotic bounds are possible, select the smallest one among them.

Question 2: Assume that you know the degrees of some graph with 6 vertices; and we need to know what kind of graph can or cannot be constructed, if the degrees are like the given ones.

(A) Write a sequence of six numbers that are values in the interval $[1; 5]$ such that there is **no graph** with the given degrees. (Or explain, why such sequence does not exist.)

(B) Write a sequence of six numbers that are values in the interval $[1; 5]$ such that there exists a graph with such degrees, but there is **no bipartite graph** with the given vertices. (Or explain, why such sequence does not exist.)

(C) Write a sequence of six numbers that are values in the interval $[1; 5]$ such that there exists a graph with such degrees that there exists a bipartite graph with the given vertices, but this graph does not have a perfect matching. (Or explain, why such sequence does not exist.)

Question 3: Assume there is a queue implemented as a cyclical array. A queue is implemented as an array with $size = 15$ elements; it has two extra variables $front$ (pointer to the first element) and $length$ (the current number of elements in the queue). Enumeration of array elements starts with 0. The array is filled in a circular fashion. The command `enqueue(elt)` inserts a new element at $(front + length)$ mod $size$. The `enqueue(elt)` command also increments the `length`.

The command `dequeue()` does not change anything in the `array`, but increments `front` by 1 and decreases `length` by 1. Thus the queue becomes shorter by 1.

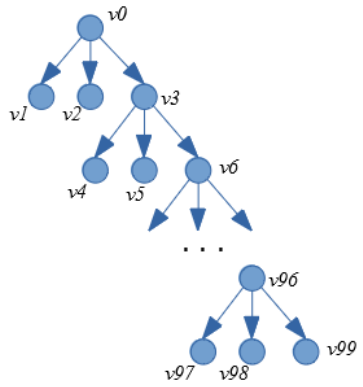
The initial state of the queue is the following:

```
size = 15
front = 0
length = 0
array[] = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The following list of size 25 contains all prime numbers from $[1; 100]$. After that we enqueue five elements, dequeue three elements (and repeat these actions five times).

```
list<int> L = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
↪ 67, 71, 73, 79, 83, 89, 97};
for (int i = 0; i < 5; i++) {
    queue.enqueue(L[5*i]);
    queue.enqueue(L[5*i+1]);
    queue.enqueue(L[5*i+2]);
    queue.enqueue(L[5*i+3]);
    queue.enqueue(L[5*i+4]);
    queue.dequeue();
    queue.dequeue();
    queue.dequeue();
}
```

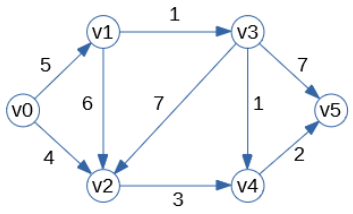
Question 4: We build a ternary tree as follows: Add node v_0. Then 33 times pick the rightmost leaf on the last level and add three children to it. You will end up with a tree with 100 nodes (see figure):



(A) Run the post-order traversal of this tree; find the four "middle" vertices a_{48}, a_{49}, a_{50}, a_{51}.

(B) Run the in-order traversal of this tree; find the four "middle" vertices b_{48}, b_{49}, b_{50}, b_{51}. (In-order traversal of a ternary tree – visit the leftmost subtree, then the parent, then the two remaining subtrees).

Question 5: Run the Bellman-Ford algorithm to find the minimum distance from the source v_0 to all the other vertices.



The pseudocode of Bellman-Ford algorithm is this:

BELLMANFORD(G, w, s):
    for each vertex $v \in V$: *(initialize vertices to run shortest paths)*
        $v.d = \infty$
        $v.p = \text{NULL}$
    $s.d = 0$ *(the distance from source vertex to itself is 0)*
    for $i = 1$ **to** $|V| - 1$ *(repeat $|V| - 1$ times)*
        for each edge $(u, v) \in E$
            if $v.d > u.d + w(u, v)$: *(relax an edge, if necessary)*
                $v.d = u.d + w(u, v)$
                $v.p = u$

As you run the algorithm, build a table (current distances from the source v_0 to all the other vertices) every time when some distances change (due to edge relaxing). The table looks something like this (but at each stage you specify actual edge that was relaxed – instead of (v_i, v_j); and also the actual distances).

Vertices	v_0	v_1	v_2	v_3	v_4	v_5
Initial distances	0	∞	∞	∞	∞	∞
Relax (v_i, v_j)	?	?	?	?	?	?
...	...					

Make sure that in the **for each** loop you visit all the edges in their lexicographical order. Show all the the relaxed edges in this table, but in case some edge does not result in changes of any distances, do not enter it into the table.