Midterm

MIDTERM, 2022-04-06

Question 1: Consider the following algorithm which can run on an array $\texttt{A}[]$; we also specify the leftmost and the rightmost element of the array in every call of COMPUTESOMETHING$(\ldots)$. The initial call is COMPUTESOMETHING$(\texttt{A}[], 0, n-1)$; after that the function calls itself recursively. It may also call $f(\ell)$, which runs in time $O(\ell)$.

COMPUTESOMETHING$(\texttt{A}[], \ell, r)$
  total = 0
  if $\ell - r \leq 2$:
    total = *total* + $f(\ell)$
  else:
    $m_1 = \lfloor (2\ell + r)/3 \rfloor$
    $m_2 = \lfloor (\ell + 2r)/3 \rfloor$
    total += COMPUTESOMETHING$(\texttt{A}[], \ell, m_1)$
    total += COMPUTESOMETHING$(\texttt{A}[], m_1, m_2)$
    total += COMPUTESOMETHING$(\texttt{A}[], m_2, r)$
  return *total*

(A) Denote by $T(n)$ the time complexity of this algorithm on an array with $n-1$ elements. Write the recurrence for the time complexity of this algorithm. Namely, express $T(n)$ in terms of $T(\ldots)$ for some smaller arguments.

(B) Apply Master Theorem to find some $g(n)$ such that $T(n)$ is in $\Theta(g(n))$.

Question 2: Some binary tree T has exactly 100 *internal nodes*. Other nodes in this tree are *leaves* – they also contain information payload (keys, values, etc.), but they do not have non-empty children (both child pointers are NULL).

(A) Can tree T be a full binary tree? Can tree T be a complete binary tree? Can tree T be a perfect binary tree?

To remind the tree terminology: In a full binary tree every node has either two children or no children at all. Complete trees are used to implement heaps (nodes are filled in level by level). In perfect binary trees all leaves have the same depth.

(B) What is the largest and the smallest value for n – the total number of nodes in the tree T? Explain your estimates.

(C) What is the largest and the smallest value for the NULL pointers in this tree (such pointers do not count as internal nodes or leaves).

(D) What is the largest and the smallest value for h – the height of T? Explain your estimates.

Question 3: An array of 10 elements are inserted into a minimum heap in the order specified here:

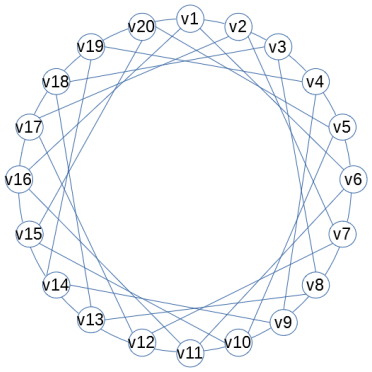
$$1, 7, 9, 4, 5, 3, 6, 8, 2, 10.$$

Assume that we do not use any fast heap-building algorithms; we just insert new elements and let them sift up.

(A) Show the final state of the tree after all the nodes are inserted in this way. Draw this heap as a complete binary tree. (You can also show intermediate results to show your approach.)

(B) What is the total number of comparisons ($a <^{?} b$) that is used during this heap building process.

Question 4: Consider the following regular 20-gon as a graph. It has 20 vertices $V_1, \ldots, V_{20}$, it has exactly 40 undirected edges – all sides of the 20-gon, and also the diagonals that connect vertices having distance exactly 5. Namely, (V_i, V_j) exists in the graph iff $(i - j) \equiv \pm 1 \pmod{20}$ or $(i - j) \equiv \pm 5 \pmod{20}$.

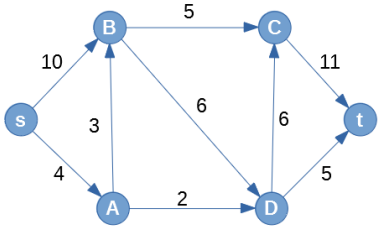


(A) Draw the BFS tree that is created if this graph is traversed in the BFS order, and vertex V_1 is the root. Make sure to show the labels of all vertices. Assume that the children for each internal node in the BFS tree are visited in the order of increasing numbers (namely, if a parent node V_i discovers two neighbors V_j and V_k where $k > j$, then V_k is a sibling drawn to the right of V_j).

(B) What is the number of internal nodes in this BFS tree? What is the number of leaves? What is the height of the BFS tree (the number of edges leading from its root to the deepest leaf)?

(C) Consider a graph – regular 100-gon with edges that are all its sides and also those diagonals that connect vertices with distance exactly 5. What is the height of the BFS tree created from this graph?

Question 5: Run the Edmonds-Karp maximum flow algorithm on the graph provided.



(A) Run Edmonds-Karp algorithm on the graph shown above. For every phase highlight the the augmenting path (or simply list its vertices), find the *residual flow* of this augmenting graph. Next to it draw a copy of the flow graph where every age is labeled by two numbers `f/c` – the actual flow `f` and also the capacity `c` of the edge. Thus, every phase shows two oriented graphs: First: The current residual graph (initially – it is simply the given graph with all flows equal to 0). Second: The original graph with edge capacities and new flows.

(B) Finally, redraw the original graph with all the maximum flows (use the same two-number labels for edges `f/c`). Show the min-cut which prevents any further augmenting paths (either highlight with another color, or simply list the partition of graph's vertices into two disjoint sets that describe the cut).