CS21003 Algorithms I, Autumn 2011–12

End-Semester Test

Maximum marks: 60 | Date: 22-November-2011 | Duration: Three hours

---

Roll no: ________________ **Name:** ____

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. Supply brief answers to the following parts. **(10)**

(a) What is the maximum possible height of a binary search tree with n nodes?

Solution $n - 1$.

(b) Name an optimal sorting algorithm.

Solution Merge sort or heap sort.

(c) What is the height-balancing condition used in AVL trees?

Solution $|h(R) - h(L)| \leqslant 1$ for every node in an AVL tree, where L and R are the left and right subtrees of the node.

(d) The running time of a divide-and-conquer algorithm satisfies the recurrence $T(n) = 5T(n/2) + \Theta(n^2)$. What is $T(n)$ in the big-Theta notation of a function of n?

Solution $\Theta(n^{\log_2 5})$.

(e) Let a min-heap consist of n distinct keys. Argue why the maximum key must be found in a leaf node.

Solution A non-leaf node u has at least one child v, and the key stored in v is larger than the key stored in u.

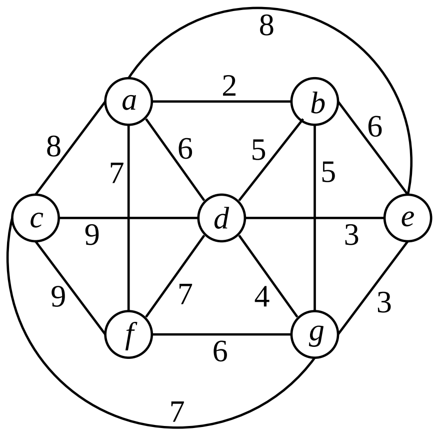
(f) What is the longest proper border of the string $ababbababa$?

Solution aba.

(g) Let G be a connected undirected graph on n vertices with each edge having a cost of 1. What is the cost of a minimum spanning tree of G?

Solution $n - 1$.

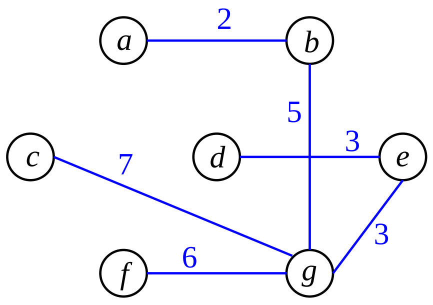
2. In Part (a) of the following figure, a weighted undirected graph is shown. Your task is to construct a minimum spanning tree (MST) of this graph using *Kruskal's algorithm*. Draw the MST of the graph in Part (b). Also write in Part (c) the sequence of edges chosen by Kruskal's algorithm and if an edge is not included in the MST, why it is discarded. (Example: (x, y) added, (u, v) discarded because it creates the cycle $uxywv$.) **(7)**



(a)

(c) Sequence of edges chosen

(a,b) added
(d,e) added
(g,e) added
(d,g) discarded (cycle: dge)
(b,g) added
(b,d) discarded (cycle: bdg)
(a,d) discarded (cycle: adegb)
(b,e) discarded (cycle: bge)
(f,g) added
(a,f) discarded (cycle: afgb)
(c,g) added



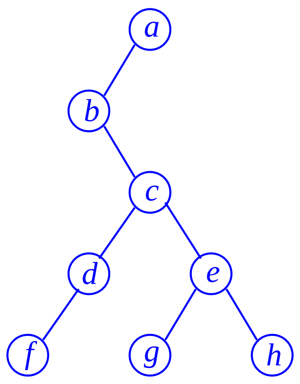
(b)

Roll no: ____________________ **Name:** ____

3. The vertex set of a binary tree T on eight nodes is $\{a, b, c, d, e, f, g, h\}$. The inorder listing of the vertices of T is $bfdcgeha$, and the postorder listing of the vertices of T is $fdghecba$. Reconstruct the tree T. Explain the relevant steps. **(7)**

Solution The last element in the postorder listing is the root, so T has root a. Since nothing follows a in the inorder listing, the right subtree of a is empty. We need to construct the left subtree of T from the inorder listing $bfdcgeh$ and postorder listing $fdghecb$. The root is b. Since nothing precedes b in the inorder listing, the left subtree of b is empty. So we recursively construct the right subtree of b from the inorder and postorder listings $fdcgeh$ and $fdghec$. The root is c. In order to construct its left subtree, we consider the listings fd and fd—this has root d and left child f. For the right subtree of c, we consider the listings geh and ghe—this is the tree with root e, left child g and right child h.

The tree T is, therefore, reconstructed as follows.



4. [*Single-destination-shortest-path problem*] Let $G = (V, E)$ be a directed graph with a positive cost associated with each edge $e \in E$, and let $v \in V$. Your task is to find shortest u, v paths for all $u \in V$. Describe an efficient algorithm to solve this problem. What is the running time of your algorithm? **(7)**

Solution Construct the graph $G' = (V, E')$ on the same vertex set V as G and with $(x, y) \in E'$ if and only if $(y, x) \in E$. The cost of (x, y) in G' is the same as the cost of (y, x) in G. Run Dijkstra's algorithm on G'.

Constructing G' from G takes $O(|E| + |V|)$ time. Since $|E'| = |E|$, Dijkstra's algorithm on G' runs in $O(|E| \log |V|)$ time. So the running time of the above algorithm is $O(|E| \log |V|)$.

5. [*Hamiltonian paths in DAGs*] Let $G = (V, E)$ be a directed graph with n vertices. A *Hamiltonian path* in G is a path in G of length $n - 1$ (that is, a path on which all vertices of G appear). In general, it is difficult to determine whether a graph G contains a Hamiltonian path. Given that G is a directed *acyclic* graph, propose an $O(|V| + |E|)$-time algorithm to determine whether G contains a Hamiltonian path. Prove the correctness of your algorithm. (**Hint:** Sort G topologically.) **(7)**

Solution Let $u_0, u_1, \ldots, u_{n-1}$ be a topological sorting of the vertices of G. Return *true* if and only if $(u_i, u_{i+1}) \in E$ for all $i = 0, 1, 2, \ldots, n - 2$.

In order to prove the correctness of this algorithm, first suppose that $u_0, u_1, \ldots, u_{n-1}$ is a Hamiltonian path in G. Then the topological sorting of G must give the sequence $u_0, u_1, \ldots, u_{n-1}$, and the algorithm outputs *true*. On the other hand, if the algorithm outputs *true*, then $u_0, u_1, \ldots, u_{n-1}$ is evidently a Hamiltonian path in G.

6. Let $A = (a_0, a_1, a_2, \ldots, a_{n-1})$ be an unsorted array of n floating-point numbers. Propose an $O(n)$-time algorithm to compute the (floating-point) number x (not necessarily an element of A) for which $\max\limits_{0 \leqslant i \leqslant n-1} |a_i - x|$ is as small as possible. Prove the correctness of your algorithm. **(7)**

Solution Compute the minimum m and the maximum M in the array A in $O(n)$ time. Output $x = (m + M)/2$.

For proving the correctness of this algorithm, let $m = a_s$ and $M = a_t$. If $x > (m + M)/2$, then $|a_s - x| > |a_s - (m + M)/2|$. On the other hand, if $x < (m + M)/2$, then $|a_t - x| > |a_t - (m + M)/2|$.

7. You are given n dates in the *dd-mm-yyyy* format. Describe an $O(n)$-time algorithm to sort the dates chronologically (from earlier dates to later dates). **(7)**

Solution The fields *dd*, *mm* and *yyyy* assume only constant numbers of values (31, 12 and $10,000$, respectively). First, sort the list by counting sort with respect to *dd*. Then, sort the intermediate list by stable counting sort with respect to *mm*. Finally, sort the second intermediate list by stable counting sort with respect to *yyyy*. (We can replace the last sort by four stable counting sorts with respect to the four digits of *yyyy*, starting at the least significant digit and ending at the most significant digit.) Each counting sort used above can be finished in $O(n)$ time.

The dates *dd-mm-yyyy* may also be considered as strings. Denote the string positions from left to right by $d_1 d_2 m_1 m_2 y_1 y_2 y_3 y_4$. Eight counting sorts on the eight string positions can sort the dates. The sequence of positions for these counting sorts should be $d_2, d_1, m_2, m_1, y_4, y_3, y_2, y_1$. All these counting sorts (except the first) need to be stable.

8. [*Multiple string matching*] Let $S, T_1, T_2, \ldots, T_k$ be strings of lengths $|S| = n$ and $|T_j| = m$ for all $j = 1, 2, \ldots, k$. Assume that $n \geqslant m$. Your task is to locate all the positions i in S at which one of the patterns $T_1, T_2, \ldots, T_k$ matches (that is, for which we have $S[i \ldots i + m - 1] = T_j$ for some $j \in \{1, 2, \ldots, k\}$). Describe an algorithm to solve this problem in $O(n + mk)$ *expected* time. (**Remark:** The input size for this problem is $\Theta(n + mk)$, so we are seeking for a linear-time algorithm. Running k instances of KMP leads to an $O(nk)$-time algorithm, where nk may be asymptotically larger than $n + mk$.) **(8)**

Solution We modify the Rabin-Karp algorithm. First, compute the hashes of the patterns: $H_j = h(T_j)$ for $j = 1, 2, \ldots, k$. Using the Rabin-Karp hash function, these k hashes can be computed in a total of $O(mk)$ time. Insert the pairs (j, H_j) in a hash table with chaining. The insertion is made with respect to the second element in the pair. Subsequently, for $i = 0, 1, 2, \ldots, n - m$, compute the hash $H = h(S[i \ldots i + m - 1])$. If $i = 0$, H is computed by directly applying h on $S[0 \ldots m - 1]$. If $i \geqslant 1$, then H is updated from the previous hash value $h(S[i - 1 \ldots i + m - 2])$ in $O(1)$ time (as is done in the original Rabin-Karp algorithm). We then search for H in the hash table. For each match (j, H) found in the hash table, we check whether $S[i \ldots i + m - 1] = T_j$ by character-by-character matching. For a randomly chosen hash function, each list in the hash table is expected to be of $O(1)$ length. So we do not expect too many values of j for which this character-by-character matching needs to be done.