IIT Kharagpur: End-semester Examination: CS31005: Algorithms II.

Department of Computer Science and Engineering

LTP 3-1-0: Credits 4: Time 3 hours: Maximum marks: 100:

There is limited choice. Write neatly with rigour and precision.

(Dated: Thursday, November 25, 2010: 2-5 pm: F-142)

1. Design a linear time algorithm for computing the *convex hull* of a polygon that is *monotone* along the Y-axis (vertical) direction. [10 marks]

2. Design an efficient polynomial time algorithm for *triangulating* a set of n points in the plane. If you cannot achieve linear running time then can you achieve an $O(n \log n)$ bound on the time complexity? [7+3 marks]

3. Show that the *first fit* heuristic for the *bin packing problem* yields a constant *ratio approximation* factor. Analyze the time complexity of this *approximation* algorithm. [8+2 marks]

4. Consider an undirected bipartite graph $G(V, E)$. Formulate an integer linear program $P1$ whose solution would yield the maximum cardinality matching for the bipartite graph G. Also, construct the dual integer linear program $P2$ of $P1$, and interpret the property of the graph that the dual program $P2$ minimizes. Are the optimal objective function values of relaxations of the integer linear programs $P1$ and $P2$ identical? Why? Are the optimal objective function values of $P1$ and $P2$ identical? [6+3+3+3 marks]

5. Suppose we sample r points independently and randomly from a given set of n distinct points on a line. It is known that all $r-1$ consecutive pairs of sampled points are assured to be separated by $O(\frac{n}{r} \log r)$ points with probability at least $\frac{1}{2}$. Using these facts, design a Las Vegas algorithm that runs in expected polynomial time creating a balanced $(r+1)$-ary search tree of the n points. What is the expected time required for creating such a search tree? How is this tree used for answering queries? [5+3+2 marks]

6. Suppose you wish to compute the product of two $(n-1)$ degree polynomials A and B, each with n coefficients.

   (a) Define the *Discrete Fourier Transforms (DFTs)* $DFT(A)$ and $DFT(B)$ for the coefficient vectors of A and B.

(b) In the $O(n \log n)$ FFT algorithm for computing the DFT of an $(n-1)$-degree polynomial A, how do we use the two DFTs ($DFT(A_{odd})$ and $DFT(A_{even})$), as returned by the two recursive calls, to compute $DFT(A)$? Show how the *halving* lemma and the *cancellation* lemma are used to establish the correctness of this process. Show that the time complexity in this step is $O(n)$.

[5+(4+3+3) marks]

7. State the preprocessing step for the *planar point location* method of Kirkpatrick. Once this data structure is built, explain the way a search query is executed. State the time complexities for preprocessing and query. [7+4+4 marks]

8. State the definition of a *valid flow* function in a network $G(V, E, c)$, where G is a directed graph and c is a valid *capacity* function defined over all edges of E. Given a valid flow function f for the network G, define the *residual network* G_f of G for flow f. Show that a minimum cut C in a flow network $G(V, E, c)$, separating the source vertex s and the sink vertex t, is obtained when there is no augmenting path left in the residual network from s to t. State the *max-flow-min-cut* theorem. [3+3+6+3 marks]

9. State Edmonds' and Karp's algorithm for computing the maximum flow in a network $G(V, E, c)$, where all augmentations are done along shortest paths in the residual networks. This network has $|V|$ vertices and $|E|$ edges. Show that augmenting flow in a residual network along a shortest path from the source to sink does not decrease the shortest path length (from source to sink) in the subsequent residual network. Assuming that any edge can become a *bottleneck* edge on an augmenting path at most $O(|V|)$ times, derive the time complexity of Edmonds' and Karp's algorithm in terns of $|V|$ and $|E|$. [3+6+6 marks]

10. We know that the usual dynamic programming algorithm for the Knapsack problem does not run in polynomial time for certain inputs. Characterize such inputs where the running time is not polynomial in the length of the input. Mention one way in which can we scale down the *profits* associated with the items so that we may get polynomial running time even though we may get approximate solutions. Would it make any sense to scale profits by a constant independent of the number n of items? Should we scale by a factor that involves the number n of items? Why? [5+4+6 marks]

==========