Term Project

CS5633
Term Project

1. Select a problem from the following list or choose another problem (get email approval in any case). Identify three distinct algorithms for solving the problem. Use the textbook, other reference books, and research papers for this purpose. Submit the statement of the problem and sketches of the three algorithms in no more than 500 words.
2. Code one of the algorithms. Pay attention to the data structures. Also, ensure that your coding is free of any unnecessary overheads, time- or space-wise. Experiment thoroughly for best performance and document all the methods tried out.

   Run your program on random inputs of varying lengths. Report your program and its run-time performance in 2-3 pages, typed double-spaced. Your report should describe the algorithm, the data structures, the time behavior, and an analytical worst-case time and space complexity analysis. Show the run-time data of your program using a plot with input length on x-axis, and time required on y-axis. Submit this report with a soft copy of your program (commented adequately). The report should be succinct and well-written. If you employ LLM, state how it was employed, and pros and cons.
3. Repeat 2 for your second algorithm.
4. Repeat 2 for your third algorithm.
5. Write a final report comparing and contrasting the three algorithms. Which algorithm would you recommend (and under what circumstances)? Substantiate drawing upon your previous reports. Also submit a plot containing data for all three algorithms. Limit the final report to two pages, with other two reports in the appendices.

Due Dates:

Submit at the beginning of the class:

- Part I: March 5
- Part II: March 26
- Part III: April 16
- Part IV,V: May 5

List of Project Topics

1. maximum flow in a weighted graph
2. vertex cover
3. triangle packing
4. Quad tree, R tree, and variant data structures for Geographic Information System (GIS)
5. intersection algorithms of large GIS polygons
6. approximate matching algorithms of multidimensional weighted data (e.g., matching homes for buyers's choices)
7. ranking web searches for display in mobile devices
8. graph node coloring
9. graph edge coloring
10. clique
11. graph isomorphism
12. steiner tree
13. satisfiability problem

14. Longest Common Subsequence
15. splay tree, heap, and skewed heap
16. traveling salesperson problem
17. graph coloring problem
18. VLSI layout algorithms

Some Reference Books for CSc 4520/6520

Algorithms Mainly dealing with algorithms, but often include complexity analysis

Basse, Computer
Algorithms Even, Graph
Algorithms
Greene and Knuth, Mathematics for the Analysis of
Algorithms Horowitz and Sahni, Fundamentals of
Computer Algorithms Klienberg and Tardos,
Algorithms Design
Knuth, The Art of Computer Programming, 3
volumes Minieke, Optimization Algorithms for
Networks and Graphs Nijenhufs and Wilf,
Combinatorial Algorithms
Reingold, Nievergelt and Deo, Combinatorial
Algorithms Skeina, Algorithm Design Manual
Wells, Elements of Combinatorial Computing

NP-Completeness: Emphasizing Complexity Theory

Aho, Hopcroft and Ullman, The design and analysis of computer
algorithms Davis, Computability and Unsolvability
Garey and Johnson, Computers and Interactability: A guide to the theory of NP-Completeness (the most well-known introductory text; easy to read)
Hartmanis, Feasible computation and provable complexity properties (very
difficult) Hennie, Introduction to computability
Machtey and Young, An introduction to the general theory of
algorithms Savage, The complexity of computing

Parallel Processing:

Grama et al. "Introduction to Parallel Computing", 2nd Edition,
Quinn, M. J., Parallel Computing: Theory and Practice, McGraw-Hill, NY, 1994. (A good introductory text) J'aJ'a, J., An Introduction to Parallel Algorithms, Addison-Wesley Pub Co, Reading, MA, 1992.
F. T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann, CA 1992.
Prasad, et al. https://tcpp.cs.gsu.edu/curriculum/?q=CDER_Book_Project – **Free Chapter Downloads**
J. H. Rief, Synthesis of Parallel Algorithms, Morgan Kaufman, San Mateo, CA, 1993.

Sample Test 1

NAME: Questions to be graded:——

Design and Analysis of Algorithms
Test 1

Notes

1. Answer any four. You have 75 minutes.
2. Show your work, as partial credit will be given. Write neatly and to the point. State algorithms at an abstract level.
3. Use both sides of the answer sheet, start each answer on a new page, and arrange your answers in increasing order of their question numbers. Write your name and the four question numbers that you want graded at top right of the first page. If you do not, then your first four answers will be graded in case you have attempted more.

Questions

1. (15 points) Prove that $3n^2 + 5n + \log_2 n = O(n^2)$ by definition 2 and by definition 3.

OR

A 3-way search (instead of a binary search) compares the key with the values at locations $n/3$ and $2n/3$ of a n-size array, and then recursively searches in one of the three intervals. Set up a recurrence relation for the worst-case time complexity of 3-way search, and solve it.

2. (15 points) State the tree (insertion) sort, odd-even transposition sort, and quicksort algorithms in 3-4 sentences each, and estimate, with suitable but brief justification, their (i) average case time complexities and (iii) their worstcase space complexities. Use known properties of binary search trees in case of tree sort and intuitive arguments about the other two to justify average case scenarios.
3. (15 points) State a situation with suitable but brief justification, if any, when you would prefer (i) linear insertion sort over binary insertion sort, (ii) heapsort over quicksort, (iii) quicksort over mergesort, (iv) tournament sort over sorting by ranking, and (v) odd-even exchange sort over bubble sort.
4. (15 points) Distinguish between time complexity and problem complexity. Briefly state the milestones (in terms of time complexity) we arrived at in the process of developing binary search algorithm starting from the sequential search and proceeding via jump search. Why did we not look further than the binary search algorithm? What was the role of decision trees in this process?
5. Set up a recurrence relation for the depth of the stack for quicksort assuming that the sizes of the left and the right partitions are compared and quicksort is recursively called on the smaller partition only. There is no recursive call on the larger partition because of tail recursion. By solving this recurrence relation, show that the depth of the stack is bounded by O(log n) on a list of n integers.
6. (15 + bonus 5 points) Prove that a lower bound on problem complexity of comparison-based sorting algorithms is $\Omega(n \log n)$.

Sample Test 2

Design and Analysis of Algorithms
Test 2 Take
Home

Notes

1. Answer any four questions. Your solutions are due at the beginning of class Wednesday. Plan your time wisely. *Do not overwork.* Be clever. Read all the problems; do easy problems first, then work on the harder problems. If you get stuck, take time outs – you may get new insights or a fresh start. You have enough time to do this exam, so be neat and precise. To guard against silly mistakes, do your solutions on scratch paper, and then neatly copy them over for final submission.

2. For each algorithm that you design, describe it stepwise in English, and give pseudocodes only if clarity demands it, and calculate its worst-case time and space complexities. More efficient your (correct) algorithm is, the better your score will be. Also, the more concise and simple your solution is, the better your grade will be. Use algorithms and theorems from the lectures and the text to simplify your solutions. If you are solving a simpler problem than the one given, state your simplifying assumptions at the beginning of your solution. For graph problems, assume n and m, respectively, to be the number of nodes and edges.

3. Policy on academic honesty: You may use your notes, your text, and other books. You may not communicate with any person, except me, about any aspect of the exam until after the hand-in deadline.

4. Bugs: If you are in doubt about the interpretation of a question, clarify it with me. Email: sprasad@gsu.edu. Good luck!

Questions

1. Give a modified algorithm for Counting Sort which does not use an output array, and is able to genearte sorted output in the input array itself. You may use a count array and an additional constant number of variables. Additionally, the Counting Sort need not remain stable.

2. Here is a divide-and-conquer based algorithm which employs Kruskal's algorithm as a subroutine to find minimum spanning forest (MSF):

   Given a undirected weighted graph $G = (V, E)$, divide the set of edges E into two equally-sized disjoint subsets E_1 and E_2 (such that $E = E_1 \cup E_2$) thus obtaining two subgraphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. Recursively, find the MSF $F_1 = (V, T_1)$ of G_1 and $F_2 = (V, T_2)$ of G_2. Then "merge" the two forests into one $(V, T_1 \cup T_2)$ and run the kruskal's algorithm on it to find an MSF for the original graph G. A subgraph is recursively split into two until the number of edges in it becomes less than or equal to n in which case Kruskal's algorithm is employed to find a MSF for the subgraph.

   Argue in a few sentences that this algorithm produces an MSF of graph G. Notice that algorithm is based on successively eliminating edges from the graph. You simply need to prove that those edges which are discarded could not belong to the MSF of G. That is, if an edge in a subgraph G' of G does not belong to the MSF of G', then that edge does not belong to the MSF of G either. For the purpose of this proof, assume that all weights are distinct, and therefore, G has a unique MSF.

   What is the time complexity of this algorithm. Do you see any advantage of this algorithm over

basic Kruskal's algorithm if you wanted to employ a parallel/multicore system.

3. Write an efficient algorithm that will find an optimal order for multiplying n matrices $A_1XA_2X \ldots A_n$ where dimen- sion of each matrix is *iXi*, *iXj*, *jXi*, or *jXj* for some fixed positive integers *iandj*. Analyze your algorithm.
4. Prove or disprove that if the weights on the edges of a connected graph are distinct, then there is a unique shortest path tree.
5. A *bipartite* graph is a graph whose vertices can be partitioned into two subsets such that there is no edge between any two vertices in the same subset. Develop an efficient algorithm to determine if a graph is bipartite. Analyze its time complexity in detail.
6. Here is a proposed shortest path algorithm to handle the case of graphs with negative edge weights. The idea is to add a large positive constant to the weight of every edge, thereby making all the weights positive, and then run Dijkstra's algorithm to find the shortest path in the new graph. So for example, if the negative edge weight is -42, one could add 50 to all edges. If this works, explain why. If it does not work, provide counter examples.

Sample Test 3

NAME:
Questions to be graded:

Design and Analysis of Algorithms
Test 3

Notes

1. Answer any six problems. Start each answer in a different page. Time =100 minutes.
2. Do not spend too much time on any problem. Read them all through first and attack them in the order that allows you to make the most progress.
3. Show your work, as partial credit will be given. Write neatly and to the point. State algorithms at an abstract level. If appropriate, you may use algorithms from the lectures and the text as subroutines.

Questions

1. Define the class NP, a polynomial transformation, and the class NP-Complete. Consider a known NP-complete problem π. What is currently known about the best algorithm for π? What would be the implication of finding a deterministic polynomial time algorithm for π?
2. Prove that if the weights are distinct, then there is a unique minimum spanning tree.
3. Work out Krushkal's Smallest-Edge-First algorithm on the following weighted undirected graph to find a minimum spanning tree (MST) starting at node *A*. Show the partial forest and the status of the working data structures (employing C-Find and W-Union) as you choose edges. Union two trees of same weight such that root of the first tree becomes the overall root.
4. Workout MST based heuristics for TSP problem on the following completely connected weighted graph.
5. Answer true/false and give one-sentence justification.
   (a) With current knowledge, an NP-complete problem will require exponential time even on a parallel computer with polynomial number of processors.
   (b) To determine whether a graph is bipartite is NP-Complete.
   (c) A dynamic programming solution is bottom up, where as its memoized version is top-down.

(d) Using Cooks theorem, to show that Hamiltonian Cycle problem is NP-complete, one must show a polynomial reduction of Satisfiability to Hamiltonian Cycle.

(e) Kruskal's minimum spanning tree algorithm can be categorized as a dynamic programming algorithm because of the optimality of the substructures and the fact that one builds larger solution from smaller ones.

(f) If a problem can be shown to be in class NP, it means that the problem cannot be in class P.

(g) An independent set in a graph is a set of nodes with no edges among those nodes. Therefore, if a graph is properly colored using 8 colors, then it has 8 independent sets.

(h) If one finds a polynomial algorithm for Satisfiability problem, then all known problems will be in class P.

6. Prove that $\lceil |V|/2 \rceil$ iterations of Bellman Ford all-to-all shortest path algorithm are sufficient with the following changes. The nodes of a graph G are indexed from 1 through n, thus partitioning G into G_f and G_b such that G_f (respectively, G_b) contains only those edges which have their source node indexed lower (higher) than their destination node. Each iteration of Bellman Ford algorithm is carried out in two phases: Phase 1 (respectively, Phase 2) relaxes the edges of G_f (G_b) in the order of increasing (decreasing) source index. *Hint:* How are the edges in a shortest path arranged with respect to the two graphs G_f and G_b?

7. Given a graph $G = (V, E)$, and *independent set* is a subset of nodes $V' \subseteq V$ such that no two nodes in V' has an edge between them. (i) State the decision problem associated with independent set problem (use standard format: Instance followed by Question), and (ii) show that it is in class NP (give Guess, Checking algorithm, and Analysis showing that time to guess and check is bounded by a polynomial in length of the instance).

8. Show that clique problem can be polynomially reduced to the independent set problem (IS) as defined in the previous question. That is, (i) show a transformation of a generic instance of Clique to an instance of IS, and prove that (ii) the transformation is answer-preserving and that (iii) it can be carried out in polynomial time. For (i), simply state the transformation in 1-2 sentences. For (ii), use 2-3 sentences to argue why a k-size clique in the original graph must mean k'-size independent set in the transformed graph, and vice versa. Here, k' is assumed to be the integer parameter in the instance of IS obtained from that of Clique's, and your transformation in (i) will specify the relationship between k and k'. For (iii), simply give the time complexity of your transformation and the order of the length of the Clique's instance. Then, argue that the former is bounded by a polynomial in the length of the Clique's instance.