CSE 202 Final Exam
Spring, 2002
Due Thursday, June 13.

During final week, I will hold office hours on Monday 11-12 to answer possible questions about the problem set, and Thursday 11-12 to collect the solutions. If you need to contact me at any other time, just pass by my office, or send me email to schedule an appointment.

Collaboration policy: final exams should be solved individually, you are not allowed to collaborate with other students. Also, please avoid using reference material other than the textbook. In particular, using “learning tools” like other texts, the web, or material from other courses, possibly searching for the solution to the problems, is not allowed.

Algorithms may be described at "high level" without actual code. You may use any lower bound, algorithm or data structure from the text or in class, and their correctness and analysis, but be careful. For example, if you construct a graph with \(n^2\) nodes and \(n^3\) edges and then run Dijkstra’s algorithm on the resulting graph, the total run-time is \(O(n^3\log n)\), because Dijkstra’s algorithm is \(O(E\log V)\). If you want to use the correctness of Dijkstra’s algorithm on the graph, you must check that edge weights are positive, since otherwise the proof that Dijkstra’s algorithm gives shortest paths does not go through. If a problem has multiple size parameters, you should express the run-time as a function of all relevant parameters; e.g., saying maximum bi-partite matching takes time \(O(|V||E|)\) is more accurate than to say it takes time \(O(V^2)\), although both are correct.

For some problems, correctness may be trivial; for others the analysis may be trivial. If it is trivial, you do not need to go into detail, but you should at least give a one or two line explanation. (Conversely, if you only give a one or two line explanation, I will view this as implicitly claiming that it is trivial.)

**Problem 1**

Consider the following recursive program that takes as input an array \(A\) of integers, and returns an integer \(f(A)\) computed from the entries in \(A\):

- Program \(P(A)\): on input \(A[0, \ldots, n-1]\), if \(n = 1\) return \(A[0]\).
- Otherwise, let \(k = \lfloor n/32 \rfloor\), \(h = \lfloor n/2 \rfloor\), create an array \(B\) of size \(h\), initialize integer variable \(c = 0\) and go on.
- For \(i = 0, \ldots, 15\), do the following
  - Copy \(A[i * k + 1, \ldots, i * k + h]\) to \(B\)
  - Call \(P(B)\) and let \(r\) be the result
  - Add \(r\) to \(c\)
- return \(c\)

Write a recurrence relation expressing the running time of the algorithm as a function of the size \(n\) of the input array [3 points]. You should measure the running time as the number of arithmetic and copy operations, i.e., assume that integers can be copied and added up in constant time.

Give the solution to the recurrence relation and prove your answer correct (possibly invoking theorems and results in the textbook). [3 points].

Now consider a memoized version of the same program. Assume memoization is performed using hash tables, or other data structure that allows to perform lookup operations in constant time. In other words, you do not have to worry about how memoization is implemented: just assume that somebody modified the program for you, and whenever the program is called on a set of values \(A[\ldots]\) for which \(P(A)\) has already been calculated, \(P\) immediately returns the right answer. Analyse the running time of the memoized version of the program, providing upper bounds on the running time, and justify your answer. [4 points]. Is the memoized version faster or slower than the original program?
Problem 2

Consider the following properties for a directed graph $G$:

- $P_1(G)$: $G$ contains either a cycle or a Hamiltonian path (or both)
- $P_2(G)$: $G$ contains both a cycle and a Hamiltonian path.

Prove that one of the properties is decidable in polynomial time, while the other one is not decidable unless P=NP.

Remarks: Each part is worth 10 points. For the polynomial time decidable property, you should: (1) give an algorithm that on input a graph $G$ says if $P(G)$ is true or false [4 points]; (2) prove that the algorithm is correct [3 points]; (3) analyse the running time of the algorithm [3 points]. For the other property you should give a proof that if there is an algorithm that decides the property in polynomial time, then there is a polynomial time algorithm to solve any problem in NP [10 points]. You can assume the NP-hardness of any of the problems that are proved NP-hard in Chapter 34 of the textbook, namely, CIRCUIT-SAT, SAT, 3CNF-SAT, CLIQUE, SUBSET-SUM, VERTEX-COVER, HAM-CYCLE, TSP. Your proof may be either a map-reduction from any of these NP-complete problems to the language $L = \{\langle G \rangle : P(G)\}$, or more generally, an algorithm that decides $L$.

Problem 3

Consider a communication network (represented by a graph) where (bidirectional) communication links between nodes are perfectly reliable and secure, but nodes can be attacked by hackers, stopping their communication and routing capabilities. The network is considered broken if there are any two uncompromised nodes (i.e., two nodes that have not been successfully attacked) that cannot communicate with each other, i.e., any path from one node to the other has to go through a compromised node.

Give an algorithm that on input an undirected graph $G$ (representing the network) computes the maximum number of nodes failures that can be tolerated before the network gets broken. (Notice: it is enough for a single pair of node not to be able to securely communicate to consider the network broken, even if other pairs of nodes might still be able to communicate.)

In this problem you should: (1) give an algorithm that solves the problem [3 points]; (2) prove that the algorithm is correct [3 points]; (3) analyse the running time of the algorithm [4 points depending on the efficiency of your algorithm].

Problem 4

You are building a multiprocessor supercomputer in the backyard of your house, using spare parts of all obsolete computers you have been collecting year after year, upgrade after upgrade, when you were in college. You have the following parts available: $n$ processors $p_1, \ldots, p_n$ (each capable of performing $p_i$ operations per clock cycle) and $n$ clocks $c_1, \ldots, c_n$ (each with a corresponding frequency of $c_i$ ticks per second). Assume that processors can be arbitrarily overclocked, i.e., you can connect any processor $p_i$ to any clock $c_j$ to deliver a raw computing power of $p_i \cdot c_j$ operation per second, but (because of physical constraints) not clock can be used for more than one processor.

You want to determine what is the best way to match clocks to processors to get the highest possible raw computational capability, expressed by the sum of the computing power of each clock/processor pair.

Give an efficient as possible algorithm that on input the array of processor speeds, and clock rates, finds the best possible matching and output the corresponding value.

As usual you should: (1) give an algorithm that solves the problem [3 points]; prove the correctness of your solution [4 points]; and analyze the running time [3 points].