CS 1510 Midterm 2
Fall 2018

Instructions

- If you are unsure how to proceed at some point, you will receive more partial credit for admitting this, than for proceeding in an incorrect direction.
- Algorithm descriptions that are predominantly pseudo-code, and that do not contain sufficient explanation, will receive no credit.
- Algorithm descriptions that use an array data structure must contain a statement of the intended meaning of the array entries. For example "The intended meaning of $A[i, j]$ is the length of the longest increasing subsequence of $x_i, \ldots, x_i$ that ends at x_j. " Algorithm descriptions that do not include such a statement, will receive no credit.

1. (10 points) Briefly answer both of the following two questions:

   (a) Wikipedia says that there are approximately 10^{86} elementary particles of matter in the visible universe, most of which are neutrinos. Approximately how many bits of memory would it take to store the number 10^{86}, if it was stored in standard binary notation? You can approximate 10 by 2^3. Show your work.

   (b) Does every problem (input/output relation) have an algorithm that solves that problem? If so, explain why. If not, give an example of an problem (input/output relation) that can not be computed by an algorithm.

2. Answer at most one of the following three questions on dynamic programming:

Option C: (10 points) We consider the standard knapsack problem. So the input is n coins $c_1, \ldots, c_n$ with associated positive integer values $v_1, \ldots, v_n$, associated positive integer weights $w_1, \ldots, w_n$, and a positive integer weight limit L.

(a) Give recursive pseudo-code to compute the maximum value one can attain from a subset of the coins that has aggregate weight at most L.

(b) Assume for the moment that restrict our attention to inputs where $L = n$, and the value of each coin is 1. Will your pseudo-code from the previous subproblem run in polynomial time or exponential time on such inputs? Justify your answer.

(c) Give iterative, array-based pseudo-code to compute the maximum value one can attain from a subset of the coins that has aggregate weight at most L. The running time of your code should be $O(n \cdot L)$. Don't forget to explain how to extract the maximum value from your array.

(d) Explain how to compute the actual subset of coins that obtains the maximum value subject to the constraint that the aggregate weight is at most L. Do not give pseudo-code. Give an English explanation.

Option B: (20 points) The input to this problem is a set of n gems. For each gem g the input specifies its value v_g in dollars and whether g is either a ruby, an emerald, or a diamond. Let the sum of the values of the gems be L. The problem is to determine if it is possible to partition of the gems into two parts P and Q, such that the gems in P have the same value as the gems in Q, and the number of diamonds in P is equal to the number of diamonds in Q. The number of rubies and emeralds in each of P and Q do not matter. Note that a partition means that every gem must be in exactly one of P or Q. Give a dynamic programming algorithm that runs in time $O(n^2 L)$.

Option A: (30 points) Assume that you are given a collection $g_1, \ldots, g_n$ of gems. Each gem's type is either diamond, ruby or emerald. You know the type of each gem. You know that the value of each gem is an integer between 1 and L, inclusive. However, you do not know the specific value of any gem. You also do not know the specific value of L.

You are provided with an assayer. A assayer functions in the following manner. You can give the assayer any two disjoint sub-collections, say S_1 and S_2, of the gems. Let $v(S_1)$ and $v(S_2)$ be the cumulative values of the gems in S_1 and S_2, respectively. The assayer then determines for you whether $v(S_1) < v(S_2)$, $v(S_1) = v(S_2)$, or $v(S_1) > v(S_2)$.

The problem is to determine if one can partition the gems into two disjoint sub-collections A and B such that:

- $v(A) = v(B)$, and
- the number of diamonds in A is equal to the number of rubies in B.

Note that every gem has to be in exactly one of A or B. Note that the distribution of emeralds does not matter.

Give an algorithm for this problem that makes at most $O(n^4\ L)$ uses of the assayer.

3. Answer at most one of the following three questions on proving NP-hardness using reductions. For each question you are asked to prove that the stated problem is NP-hard using reductions, the fact that it is NP-hard to decide if a graph H has a clique of size ℓ, and the fact that it NP-hard to determine if a graph H has an independent set of size ℓ. Make sure to explain how one proves a problem NP-hard.

   Option C: (10 points) Prove that the following problem is NP-hard. The input is an undirected graph G and an integer k. The problem is to determine if G contains a clique of size k **AND** an independent set of size k.

   Option B: (20 points) Prove that the following problem is NP-hard. The input is an undirected graph G and an integer k. The problem is to determine if G contains a clique of size k **OR** an independent set of size k.

   Option A: (30 points) Prove that the following problem is NP-hard. The input is an undirected graph G. Let n be the number of vertices in G. The problem is to determine if G contains a clique of size at least $3n/4$ **OR** an independent set of size at least $3n/4$.

4. Answer at most one of the following three questions.

Option C: (10 points)

(a) Assume that you have n numbers $x_1, \ldots, x_n$ stored in a doubly link list. Assume that each of n processors has a pointer to a unique record in the linked list. The is no relationship between the processor ID and the record pointer. Give an EREW algorithm that transfers these numbers to a one-dimensional array A of size n, such that x_i ends up in $A[i]$. You algorithm should run in time $O(\log n)$. You can assume that each record has $O(1)$ additional uninitialized fields that your algorithm may use.

(b) Define the efficiency of a parallel algorithm. Determine the efficiency of the algorithm for the previous subproblem. Assume that the sequential time used to compute the efficiency is the time used by this parallel algorithm on one processor. Show your work.

(c) Define the folding principle in terms of the time used by a parallel algorithm. Make sure to be careful in your choice of $<, >, =, \geq, \leq$ in your statement of the folding principle. What does the folding principle say about the time used this algorithm when it uses $n^{2/3}$ processors?

(d) Define the folding principle in terms of the efficiency of a parallel algorithm. Make sure to be careful in your choice of $<, >, =, \geq, \leq$ in your statement of the folding principle. does the folding principle say about the efficiency of this algorithm when it uses $n^{1/3}$ processors?

Option B: (20 points)

(a) Give a CREW algorithm to compute the length of the shortest path between every pair of vertices in an edge-weighted directed graph G. You can assume each edge weight is positive. Your algorithm should run in time $O(\log^2 n)$ using n^3 processors. Assume G is represented as an adjacency matrix. So your output is an array SP where $SP[i, j]$ is the length of the shortest directed path from vertex i to vertex j.

(b) Explain how to convert this into an EREW algorithm, that runs in time $O(\log^2 n)$ using n^3 processors.

Option A: (30 points)

(a) Give a CREW algorithm to compute the the shortest path between every pair of vertices in an edge-weighted directed graph G. You can assume each edge weight is positive. Your algorithm should run in time $O(\log^2 n)$ using n^3 processors. Assume G is represented as an adjacency matrix. So your output is an array SP where $SP[i, j, k]$ is the identify of kth vertex on a shortest directed path from vertex i to vertex j.

(b) Give a CREW algorithm to compute the longest common subsequence of two sequences of integers $x_1, \ldots, x_m$, and $y_1, \ldots, y_n$ in time $O(\log^2 n)$ using at most a polynomial number of processors. Explain how many processors you need to implement your algorithm. Note that your algorithm should return the actual longest common subsequence, not just its length.