

CSE 101 Homework 1

Order, correctness proofs, time analysis, and speeding up algorithms with restructuring, preprocessing and data structures.

Due Jan. 27

100 points total = 10 %

Solve each problem. For algorithm problems, if the problem only specifies that you need to give a proof of correctness, then no time analysis is required. If it specifies that you need to give an efficient implementation, then you do not need to give a correctness proof for the basic strategy (just explain why your version actually carries out the strategy). If it says to do both, or doesn't specify what parts you need, you need to give both a proof of correctness and time analysis.

Order (10 points) Is it always the case that $f(2n) \in O(f(n))$? If so, give a proof; if not, give a counter-example.

Levelling a DAG. (20 points) The next two problems concern the following computational problem and algorithm. The input is a directed, acyclic graph (DAG) $G = (V, E)$. A *levelling* of a DAG assigns each $u \in V$ a positive integer value $level(u)$ meeting the following constraint: If $(u, v) \in E$, then $level(u) < level(v)$. (One application that might be helpful is to think of nodes as jobs, and an edge (u, v) represents that the output of job u is an input to job v . A levelling assigns each job a time so that all jobs at that time can be done concurrently.) The following algorithm strategy computes a levelling of the DAG that minimizes the maximum assigned level.

Level(G=(V,E): DAG): array of integers indexed by V

1. Initialize $level(x)$ to NIL for all $x \in V$.
2. Repeat $n = |V|$ times:
 3. Find a node x so that there are no edges to x from unassigned nodes, i.e., no edges (y, x) where $level(y) = NIL$.
 4. If there are no edges $(y, x) \in E$ THEN $level(x) \leftarrow 1$.
 5. ELSE $level(x) \leftarrow 1 + \max_{y, (y,x) \in E} level(y)$.
6. Return the array $level$.

[Part A: Correctness proofs– 10 points] The following is a proof that the above algorithm finds the levelling that uses the smallest number of levels. The proof is missing some phrases, denoted by Roman numerals. Fill in the missing phrase to complete the proof. (You can just turn in a list of the missing phrases, with the corresponding numerals.)

We need to show that the array by the above algorithm $level$ is a levelling and that no other levelling has a smaller maximum level assigned.

To prove that $level$ is a levelling of G , we need to show that for any u, v with $(u, v) \in E$, $level(v) \leq 1 + level(u)$. Consider the iteration in which $level(v)$ was defined. Since $(u, v) \in E$, the IF condition in line II was $level(u) \geq level(v)$ so we assigned $level(v)$ the value $level(u) + 1$. Now $level(u) \leq \max_{y, (y, x) \in E} level(y) + 1$ since $(u, v) \in E$, so $level(v) \leq 1 + \max_{y, (y, x) \in E} level(y) + 1$, which is what we needed to prove.

Now we need to show that no other levelling L has a smaller maximum level assigned. Let v_t be the vertex assigned a level in the t 'th iteration of our algorithm. We will prove by strong induction that for all t , $level(v_t) \leq L(v_t)$.

Assume the claim holds for all $1 \leq i < t$, i.e., that $level(v_i) \leq L(v_i)$ for all nodes v_i previously assigned a level. We need to show that the claim also is true at t , i.e., that $level(v_t) \leq L(v_t)$.

There are two cases, depending on the branch in line III . In the first case, there are no edges of the form (v_t, v_i) . In this case, we assign $level(v_t)$ value $level(v_t)$ in line IV . Since $L(v_t)$ is a levelling from the definition of levelling, we have $level(v_t) = 1 \leq L(v_t)$ as needed.

In the other case, there are edges of the form (y_0, v_t) in E , and we assign $level(v_t)$ value $level(y_0) + 1$, in line V . Pick y_0 so that $(y_0, v_t) \in E$ and $level(y_0) = \max_{y, (y, v_t) \in E} level(y)$. Then since at time t , there are no edges of the form (v_t, v_i) , $y_0 = v_i$ for some $i < t$. Thus, we can apply the induction hypothesis to y_0 to see that $level(y_0) \leq L(y_0)$. Since L is a levelling, and $(y_0, v_t) \in E$, $L(v_t) > L(y_0)$ and since both are integers, $L(v_t) \geq L(y_0) + 1$. Thus, we have $level(v_t) = level(y_0) + 1 \leq L(y_0) + 1 \leq L(v_t)$, which is what we needed to prove.

Therefore, by strong induction on t , we have $level(v) \leq L(v)$ for all $v \in V$. It follows that $\max_v level(v) \leq \max_v L(v)$, so our algorithm uses the minimum possible number of levels.

Part B: Data structures and efficient versions of algorithms, 10 points.

For the above algorithm (levelling a DAG) give an efficient implementation, specifying pre-processing and data structures. Give a time analysis. Assume that the directed, acyclic graph G is given in adjacency list format, with the list at x being nodes y where $(x, y) \in E$. Let $n = |V|, m = |E|$, and give your time analysis in terms of both n and m .

Merge lists (20 points) You are given an array of k sorted, non-empty lists, $L[1..k]$, where each $L[I] = a[I, 1] < a[I, 2] < \dots < a[I, n_I]$. Let $n = \sum_{I=1}^k n_I$ be the total sizes of all the lists (so in particular, $n \geq k$). Give an $O(n \log k)$ time algorithm that returns a sorted list containing exactly the elements in the union of the k lists.

Maximum sum of four symmetric entries, 10 points Give an efficient algorithm for the following problem: Given an $n \times n$, (where $n \geq 2$) matrix of integers $A[I, J]$, find four entries of the form $A[I_1, J_1]$, $A[I_2, J_1]$, $A[I_1, J_2]$ and $A[I_2, J_2]$ where $1 \leq I_1 < I_2 \leq n$ and $1 \leq J_1 < J_2 \leq n$ whose sum is maximum. Try to be strictly faster than $O(n^4)$. (3 pts correct algorithm and correctness proof, 7 pts. efficiency and time analysis).

Best Party Time-20 points You have n friends, of whom you want to invite at most $1 \leq k \leq n$ to a two-hour party. Each friend F_i told you the earliest time a_i that they can arrive, and the latest time d_i that they would need to leave. You have also rated each friend with a “fun factor” f_i saying how much fun they add to the party. You want to pick a time t to start the party and invite up to k friends i that can attend the whole party ($a_i \leq t < t + 2 < d_i$). Given this constraint, you wish to maximize the sum of the fun factors of invited guests. Give an efficient algorithm to solve this problem. My best algorithm is $O(n \log n)$.

Implementation-20 points Implement bubble-sort and heap-sort. You can use heaps from a standard library to implement heap-sort. Plot their performance on random arrays of n integers with values between 1 and n , for $n = 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}$. Plot their performance on a log-log scale. Is heap-sort always better than bubble-sort? Why or why not?