CMPSCI 311: Introduction to Algorithms **Spring 2018**

Homework 2

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Instructions. You make work in groups, but you must individually write your solutions yourself. List your collaborators on your submission.

If you are asked to design an algorithm as part of a homework problem, please provide: (a) the pseudocode for the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submission instructions. This assignment is due by 8:00pm on 10/7/2016 in Moodle. Please submit a pdf file. You may submit a scanned handwritten document, but a typed submission is preferred.

1. **Graph short answer.**

(a) Let us work towards a contradiction. Suppose that G is a graph and that every vertex has degree at least $n/2$, but that G is not connected. Since G is disconnected, there must exists two vertices $u, v \in V$ for which there is no path from u, v. Let S be the set of neighbors of u and let T be the neighbors of v. By assumptions $|S| \geq n/2$ and similarly $|T| \geq n/2$. Since there are n vertices, since $u \notin S$ and since $u \notin T$ (otherwise we have a path immediately), this means that $|S \cap T| > 0$ which implies that there exists a path from u, v, establishing a contradiction.

(b) If there does not exist a vertex v such that deleting v destroys all $s - t$ paths, then there must exists at least two $s - t$ paths that share no vertices. By assumption all $s - t$ paths have length strictly greater than $n/2$, but this immediately establishes a contradiction, since these two $s - t$ paths comprise strictly more than n vertices, while the graph is on just n vertices.

(c) Consider the following graph on vertices s, t, u_1, u_2 where $w(s, t) = 4$, $w(s, u_1) = w(u_1, u_2) = w(u_2, t) = 1$. There are two $s - t$ paths, one follows the single edge and has weight 4. The other follows $s - u_1 - u_2 - t$ and has weight 3 so this is the shorter path. But in G_2 all the edge weights are increased by one, so the first path now has length 5, while the latter has length 6.

2. **Directed Graphs**

Perform a topological sort using the algorithm we saw in class. Suppose we find that $v_1, \ldots, v_n$ is a topological ordering. To test if there exists a directed path that visits all edges (also called called a Hamiltonian path), check if every consecutive pair in the topological sort is a valid edge. Formally check if $(v_i, v_{i+1}) \in E$ for all $i \in [n-1]$.
If the graph has a Hamiltonion path, there is a unique topological ordering, since the existence of an edge $v_i \to v_{i+1}$ implies that v_i must come before v_{i+1} in the ordering. Clearly, $v_1, v_2, ..v_n$ is the only valid topological sort that satisfies this condition.
On the other hand, if no Hamiltonian path exists, then there never exists a topological sort with each consecutive pair being a valid edge. Thus this algorithm detects the Hamiltonian path IFF it exists.
We can identify a topological ordering in $O(|V| + |E|)$, and runtime of checking edges is $O(|V|)$.

3. **K&T Ch4.Ex3.**

We prove that after using k trucks, the greedy algorithm has shipped more boxes than any other possible algorithm. More formally, if the greedy algorithm fits boxes $b_1, \ldots, b_i$ into the first k trucks and some other strategy fits boxes $b_1, \ldots, b_j$ into the first k trucks, then we must have $j \leq i$.
To prove the claim, we use induction. When $k = 1$ the claim holds and the argument is straightforward: the greedy algorithm packs as many boxes as possible into the first truck. Now for $k > 1$ we know that greedy packs i' boxes into the first $k - 1$ trucks, and let j' denote the number of boxes that the other strategy packs into the first $k - 1$ trucks. By induction we know that $j' \leq i'$. Now on the kth truck, the alternate solution packs $b_{j'+1}, \ldots, b_j$ onto the truck. And since $j' \leq i'$, this implies that greedy can pack $b_{i'+1}, \ldots, b_j$ onto the kth truck, which shows that $j \leq i$.
Thus for every k the greedy algorithm packs the maximum number of boxes onto the first k trucks, and hence it is optimal.

4. **K&T Ch4.Ex3.**

(a) **Algorithm**: Use a greedy algorithm that invokes status checks as late as possible, right when the most immediate unchecked process is about to expire. Keep track of currently unchecked processes in a queue Q, and keep track of processes that have been successfully checked in a set S. In other words, sort the jobs in increasing order of finish time. Then do the following:

Algorithm 1 Algorithm for covering all processes

1: Sort jobs in increasing order of finish time.
2: Sort jobs in increasing order of start time.
3: merge the two lists into one list A of size $2n$.
4: **for** $i = 1, \ldots, 2n$ **do**
5: If $A[i]$ is a starting time, mark the job as "running."
6: If $A[i]$ is a finish time, and the job is not "checked", invoke `statusCheck` and mark all "running" jobs as "checked."

Supposing there are n processes, this algorithm requires $O(n \log n)$ to initially sort start and finish times, and $O(n)$ to go through the processes to invoke all the status checks.

Proof by induction: Let the set of invocation times produced by this algorithm $\{t_1, t_2, ...t_m\}$ be called T (so we are using m to denote the number of status checks invoked by the greedy algorithm). We prove that T is the smallest such set by induction. Consider some alternative set of times T' produced by another algorithm:

Base case: Up until the first finish time t_1, T' must have at least one invocation, otherwise we know that the job that ends at t_1 will go unchecked.

Inductive hypothesis: Up until time t_k, T' must have at least k invocations.

Induction step: Consider time t_{k+1}. We know there is a sensitive process not covered by t_k which ends at t_{k+1}. Therefore T' must invoke a status check between t_k and t_{k+1} to cover this process. With k guaranteed invocations up to time t_k, then there are $k + 1$ guaranteed invocations up to time $t_k + 1$ in the set T'.

(b) Let N be the number of invocations of `statusCheck`. This questions essentially asks to prove whether or not it is always valid to let $m = k^*$. We will show that this is true.

Note that the reasoning of the security consultants already proves that $m \geq k^*$, since we can find a set of k^* sensitive processes with no two running at the same time. What remains is to show that $m \leq k^*$

Consider the set of processes that dictate the invocation of status checks according to our algorithm in part a). By definition, these processes are disjoint since each is only covered by one status check. The size of this disjoint set is less than or equal to k^* by definition of k^*. Since N is equal to the size of this disjoint set, then $m \leq k^*$ always.

5. **(0 points).**