**CS180 — Algorithms and Complexity**
**Winter 2015**
**Wednesday, February 11, 4:00–5:50pm**
**D.S. Parker, Yuh-Jie Chen, Xiaoran Xu, Garrett Johnston**

**Midterm Examination — Partial Solutions**

If you have questions or concerns about grading of the midterm, please just write them clearly on the front page and give your exam back to us for regrading.

A **Master Theorem**: for $a > 1$, $b > 1$, $k \geq 0$, and assuming $T(1) = \Theta(1)$, the solution for the recurrence $T(n) = a\,T(n/b) + c\,n^k$ is

$$\begin{aligned} T(n) &= \Theta(\, n^\ell \,) && \text{if } \; k < \ell = \log_b a \\ T(n) &= \Theta(\, n^\ell \, \log n \,) && \text{if } \; k = \ell = \log_b a \\ T(n) &= \Theta(\, n^k \,) && \text{if } \; k > \ell = \log_b a. \end{aligned}$$

useful identities: $\displaystyle\sum_{k=1}^{N} k^p = \frac{1}{p+1} N^{p+1} + O(N^p)$ $\displaystyle\sum_{k=1}^{N-1} a^k = \frac{a^N - 1}{a-1}$ $\displaystyle\sum_{k=1}^{N-1} k\,a^k = \frac{N\,a^N}{a-1} + O(a^N)$

1. **The Master Theorem (25 points)**
Three platypuses meet in a bar and start to argue about the Master Theorem.

   (a) **Master Theorem? (8 points)**
   One of the platypuses says that, if we assume that $a > 1$, $b > 2$, $\ell = \log_b a$, and $c$ and $k$ are positive constants, then the recurrence $T(n) = a\,T(n/b) + c\,n^k$ has solution

$$T(n) = \begin{cases} \Theta(n^\ell) & \text{if } a > b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^k) & \text{if } a < b^k. \end{cases}$$

   The other two platypuses laugh and say this is wrong. The first one gets angry and asks you to help prove it. The two laughing platypuses ask you to give a counterexample. What is your answer?

   *This recurrence is equivalent to the Master Theorem on the cover page, obtained by exponentiating — e.g.:*

$$k < \ell = \log_b a \iff b^k < b^\ell = b^{\log_b a} \iff b^k < b^\ell = a \iff a > b^k.$$

   (b) **Laughing Theorem (8 points)**
   One of the laughing platypuses says that the solution of $T(n) = a\,T(n/b) + c\,\log_b n$ is

$$T(n) = \Theta(n \log n) \qquad \text{(assuming } n \text{ is a power of } b, \text{ and } T(1) = O(1))$$

   and asks you to prove this. The angry platypus says no, and asks for the correct solution. What is your answer?

   *The angry platypus is right; the solution should be $\Theta(n^\ell)$, as in the 'small' case of the Master Theorem. Consider the recurrence $T(n) = aT(n/b) + f(n)$ when $f(n) = c\,(\log n)$:*

   *As usual define $\ell$ and $N$ so $n^\ell = n^{\log_b a} = a^{\log_b n} = a^N$. (Also $b^\ell = a$.)*

$$\begin{aligned} T(n) &= a^N\,T(1) + \sum_{k=0}^{N-1} a^k\, f(n/b^k) \\ &= a^N\,T(1) + \sum_{k=0}^{N-1} a^k\, \log_b(\, b^{N-k}\,) \\ &= a^N\,T(1) + \sum_{k=0}^{N-1} a^k\,(N-k) \\ &= a^N T(1) + \frac{a}{(a-1)^2}\,\left(a^N - N(a-1) - 1\right) \\ &= n^\ell \left(1 + \frac{a}{(a-1)^2}\right) - \frac{a}{(a-1)^2}\,\left((a-1)\log_b n + 1\right) \\ &= \Theta(\, n^\ell \,) \quad \textit{for } a > 1,\ b > 1, \textit{ so that } \ell = \log_b(a) > 0 \textit{ and } \log_b n = o(n^\ell). \end{aligned}$$

   *The 'useful identities' on the cover page were intended to be useful for this derivation.*

(c) **Time Complexity (9 points)**
The platypuses start fighting over the asymptotic complexity $T(n)$ of the following algorithm $A$:

```
def A(x,y):
  if length(x) == 1:  return f(x,y);
  x1 = first_half(x); x2 = second_half(x);
  y1 = first_half(y); y2 = second_half(y);
  z1 = A( x1, y1 );
  z2 = A( x1, y2 );
  z3 = A( x2, y2 );
  return  f( z1, z2, z3 );
```

*Solution: This is the integer multiplication algorithm from Chapter 5 in [KT].*

$$\begin{aligned} \textit{recurrence: } T(n) &= 3\,T(n/2) \;+\; \Theta(n) \\ T(1) &= \Theta(1) \\ \textit{solution: } T(n) &= \Theta(n^{\log_2(3)}) \end{aligned}$$

2. **Shortest Paths (25 points)**

(a) **Graph with Distinct Edge Lengths (6 points)**
Kim complains that their graph is too small and spends $3M on a larger directed acyclic graph $G = (V, E, \ell)$. In this graph, all edges $e$ have distinct (unique) lengths $\ell(e) > 0$. She asks you whether Dijkstra's algorithm is guaranteed to yield a unique shortest-path tree from any source node $s$ in her new graph.

*We know that Minimum Spanning Trees are unique if the edge costs are unique. This question was asking whether the same was true for Shortest Path Trees. The answer is no: even with unique edge costs, different trees can yield the same shortest-path distances to all nodes.*

*Problem: some students asked during the exam what 'unique' meant here, because (they assumed) Dijkstra's algorithm always yields a single result. This was clarified on the blackboard. However, some students couldn't read the blackboard because the lighting was poor — either too dim or too much glare. Because it was not possible to fix this, and clarify how to interpret 'unique', all students received credit for this problem.*

(b) **Graph with Negative Edge Lengths (6 points)**
Kanye has a life-changing experience and realizes *we all need negative edges*. He buys lots of directed graphs with negative edge lengths, but never buys graphs that have cycles with a negative total length. He asks you: 'if I use Dijkstra's algorithm on these graphs, is its resulting shortest-path tree guaranteed to be correct?'

*The point of this problem is that Dijkstra's algorithm can be fooled if there are negative edges: it is basically a greedy strategy, and it is not hard to construct examples where its selection of a node v at each step turns out to be suboptimal — i.e., there can be a better path to v that is longer but includes negative edges.*

*Problem: although negative edges were discussed in HW2 and negative cycles were mentioned briefly in class, some students did not understand what 'cycles with a negative total length' meant, and how this would affect the definition of shortest paths. Again some also could not read clarifications on the blackboard due to the lighting. Because it was not possible to clarify 'negative cycles', all students received credit for this problem.*

(c) **Air Travel (6 points)**
HW2 gives you the US airport graph $G = (V, E, \ell)$, and asks you to find a shortest path tree $T$ from LAX.
All edge lengths were given as distances in miles, but if we divide by some typical airspeed like 500mph,
we can convert the edge lengths into hours. So assume each edge length $\ell(e)$ is given in hours.
Kim complains that your shortest paths are not realistic, since air travel requires at least a one-hour layover in each airport, so you change the length $\ell(e)$ of every edge $e$ in the graph to $\ell'(e) = \ell(e) + 1$ hour.
Is the tree $T$ for $G$ guaranteed to still be a shortest-path tree from LAX for the changed graph $G' = (V, E, \ell')$?

*No — changing the edge lengths can change the shortest path tree.*

3. **Minimal Spanning Trees (25 points)**
Two highly-paid consultants, Kleinberg and Tardos, are arguing about spanning trees in an undirected graph $G$.
You are hired as an even more highly-paid consultant-consultant to resolve their dispute.

(a) **MSTs with Negative Costs (9 points)**
Assume that all edge costs $c$ are distinct, but the edge costs are permitted to be *negative*.
Kleinberg argues the MST can be determined just by using Kruskal's algorithm.
Tardos says this is ridiculous, Kruskal's algorithm won't work in this case. Which consultant is right?

*Kleinberg is right; the sign of edge costs doesn't affect correctness of algorithms like Kruskal's that worry only about the ordering of costs.*

(b) **Maximum Spanning Trees (9 points)**
The company changes its specifications so that all edge costs $c$ must satisfy $c > 1$, and it wants an algorithm to construct **Maximum** Spanning Trees. In other words, it wants an efficient algorithm that finds a spanning tree with the property that the sum of its edge costs is *maximum*.

Kleinberg says that this new Maximum Spanning Tree problem is hard, and will take exponential time to solve.
Tardos says the problem can be easily solved with minor changes to any Minimum Spanning Tree algorithm.
Which consultant is right?

*Tardos is right; simply multiplying costs by $-1$ will permit a MST algorithm to obtain a Maximum Spanning Tree.*

4. **Interview Questions (25 points)**

- $G$ is a directed acyclic graph if and only if $G$ has a node with no incoming edges.

  *This is false for directed graphs $G$: if $G$ is a DAG, then it has a node with no incoming edges; however if $G$ has a node with no incoming edges, it is not necessarily a DAG.*

  *Note: 'if and ony if' is NOT equivalent to 'if'. The statement '$A$ if and only if $B$' means: $(B \Rightarrow A)$ & $(A \Rightarrow B)$.*

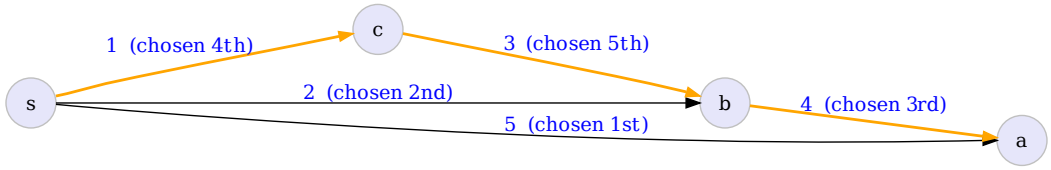
- **Longest Path Problem (4 points)**
  In the longest path problem, we're given a weighted directed graph $G = (V, E, \ell)$, and
  a source $s \in V$, and we're asked to find the longest path from $s$ to every vertex in $G$.
  In general, it's not known whether there is an efficient algorithm to solve the Longest Path problem.
  If we restrict $G$ to be acyclic, however, this problem can be solved in polynomial time.
  Give an efficient algorithm for finding the Longest Paths from $s$ in a weighted directed acyclic graph $G$.
  (Hint: no incoming edges)

  *A good algorithm for finding a longest path in a DAG is to combine topological sorting with an algorithm for tracking path lengths. At every node $v$ we keep track of its maximum path distance from the source node $s$.*

  *The Hint suggested repeated removal of nodes with no incoming edges (as in topological sorting). More specifically: at each step one can select one node $v$ with no incoming edges, and use $v$'s maximum path distance to update the distance of each of $v$'s children. For each child, if their incoming edge from $v$ (i.e., a path to them through $v$) increases their distance from $s$, then their maximum path distance is updated. At the end, the maximum of all node distances is the longest path distance.*

  *This approach resembles the Bellman-Ford algorithm, except that each node is visited only once (in topsort order). Since $G$ is a DAG, updates propagate only in one direction.*

  *Dijkstra's algorithm — with 'min' replaced by 'max' — is* not *sufficient. At each step it chooses the node $v$ that is as far as possible from $s$ using an edge from the current set $S$. This edge choice is* greedy*: it does not consider the possibility that longer paths outside $S$ to $v$ might exist, but simply commits to this path and adds $v$ to $S$. By including additional long edges outside $S$ we can construct a graph in which this choice of edge (with maximum current distance) is suboptimal; for example the greedy maximum-first ordering of nodes chosen by Dijkstra's algorithm shown in this graph misses the longest path $s \to c \to b \to a$.*



*Another approach that is similar is to use BFS from the start node, and incrementally update node distances. However, consider a graph in which all edge lengths are 1, so that the longest path distances to nodes from $s$ directly reflect their topological ordering. The BFS layers are* not *equivalent to a topological ordering, so whatever updating process is used in a BFS solution must be more complex.*

*DFS might work somehow, but there is a similar problem: DFS visits nodes depth-first, without a mechanism for topological ordering, so it is not clear how the search will yield a longest path. A backtracking algorithm that generates all possible paths will find the longest path; however the problem asked for an* efficient algorithm, *and backtracking can generate an exponential number of paths. It might be possible to construct a DFS-based algorithm, but like BFS, it must be able to yield a topological ordering. If some version of DFS can do this, it must be more complex than the basic version.*

- **Hamiltonian Path (4 points)**
  A Hamiltonian Path is a path that visits all nodes in a graph. Explain how, given a *directed acyclic graph* $G$, it is possible to determine in time $O(V + E)$ whether $G$ has a Hamiltonian Path.

  *This problem is a special case of the previous problem. When all edges lengths are 1, a Hamiltonian Path is a longest path of length $n - 1$. Some of your alert classmates solved this problem by immediately reducing it to Longest Path in a DAG.*

  *Another solution, which is very elegant, is to apply the Hint of the previous problem. Since $G$ is a DAG, it must have a node with no incoming edges. At any point in time, if there is more than one node like this, $G$ cannot have a Hamiltonian Path. So we can easily extract a Hamiltonian Path simply by repeated removing the unique node with no incoming edges (failing if it is not unique), and then repeating this process with its children.*

  *Notice this process is equivalent to topological sorting — so there must be a unique topological ordering — and this topological ordering determines the Hamiltonian Path.*

  *This process is similar to BFS in a way, but BFS by itself will not generally yield a topological sort ordering. Also, neither Dijkstra nor DFS will obviously do this either. With Dijkstra, it needs to be clarified how the algorithm can identify the best node to visit next: multiple nodes could have the same shortest-path distance. Similarly, with DFS the selection of nodes requires some ordering (like topological sort ordering) consistent with the Hamiltonian Path. For example, the kind of graph below can confuse BFS and DFS — it has a 2-level BFS tree, and many paths to consider besides the Hamiltonian Path. So a variant of these algorithms for solving Hamiltonian Path must be able to deal with graphs like this one, having many different paths, even though the topological ordering at the core of the graph is extremely simple.*