NAME and UID: Junkyun Kim 704 443

CS180 — Algorithms and Complexity
Winter 2015
D.S. Parker, Yuh-Jie Chen, Xiaoran Xu, Garrett Johnston

Midterm Examination
OPEN BOOK, OPEN NOTES
Wednesday, February 11, 4:00–5:50pm

Do not cheat.

Problem	Points
1	19 /25
2	25 /25
3	25 /25
4	16 /25
Total	/100

In any answer on this exam, you can make reference to any definition or result from the [KT] text by giving a section number, page number, gray box number, verbal summary, etc.

There is NO need to provide a complete reproduction or proof of these results. Short answers are good.

Similarly, you can use any definition or result from the course notes, slides, homework assignments, etc. Just be clear in your references to these sources.

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

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

A **Minimum Spanning Tree** in an undirected graph with edge costs $G = (V, E, c)$ is a spanning tree T for G that has minimal total cost $\sum_{e \in T} c(e)$.

A **Shortest Path** from a source node s to t in a directed or undirected graph $G = (V, E, \ell)$ with edge lengths ℓ is a path P from s to t with minimal total length $\sum_{e \in P} \ell(e)$.
In this exam, a **Shortest Path Tree** is a directed tree of edges outward from s selected by Dijkstra's algorithm.

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

1. **The Master Theorem (25 points)** – 6

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 $T(n) = aT(n/b) + cn^k$ 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?

- [x] the first platypus is right, the recurrence is correct.
- [ ] the laughing platypuses are right, the recurrence is incorrect.

Short proof, or counterexample:

If you take $\log_b$ of if statements, the statement becomes

$$T(n) = \begin{cases} \Theta(n^l) & \text{if } \log_b a > k \\ \Theta(n^k \log n) & \text{if } \log_b a = k \\ \Theta(n^k) & \text{if } \log_b a = k \end{cases}$$

And $\log_b a = l$. After switching $\log_b a$ to l, this is equivalent to master's theorem defined in front page of the exam. Also, if $l = k$, $n^k = n^l$, so n^k can substitute n^l.

(b) **Laughing Theorem (8 points)**

One of the laughing platypuses says that the solution of $T(n) = aT(n/b) + c\log_b n$ is

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

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

- [ ] the laughing platypus is right, the solution is correct.
- [x] the angry platypus is right, the solution is incorrect.

Short proof, or corrected solution:

$T(n) = \Theta(n^l)$, because n^l is greater than $c\log_b n$.

–6 (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( x2, y2 );
  z3 = A( x1+x2, y1+y2 );
  return  z1 + z2 + z3 + f(x,y);
```

recurrence: $T(n)$	=	~~6~~ 3 $T(\frac{n}{2}) + n$
$T(1)$	=	$\Theta($ ~~n~~ 1 $)$
solution: $T(n)$	=	$\Theta($ ~~(n)~~ $n^{\log_2 6}$ 3 $)$

Assume that the inputs x and y are vectors of size n, and the input lengths n are always a power ~~of 2.~~

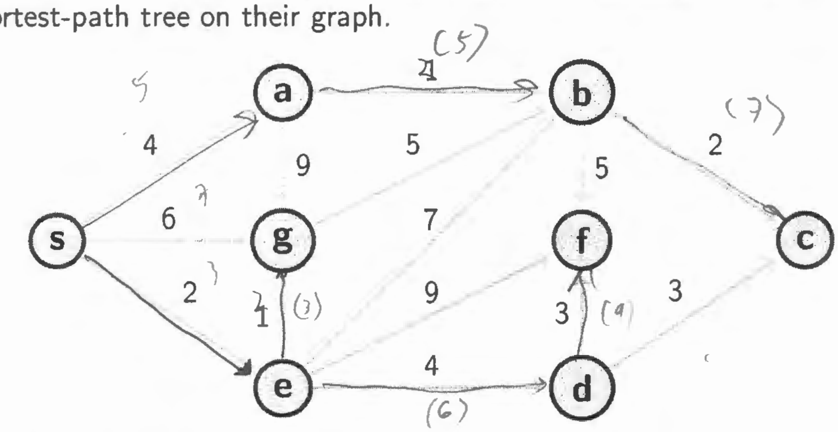
The functions `first_half` and `second_half` each take an vector of size n as input, and yield an output vector of size $n/2$. The function `f` takes time $\Theta(n)$ to compute if its arguments have length n.

Please stop the fight by providing the recurrence for $T(n)$ and its solution in the box above.

…test Paths (25 points)

(a) Applying Dijkstra's algorithm (7 points)

Kim Kardashian and Kanye West buy the directed weighted graph G below for $2M. They are trying to apply Dijkstra's algorithm to the graph, starting at vertex s. They want to know what is the order in which vertices get added to the shortest-path tree. Please tell them the order by filling in the box, and draw the resulting shortest-path tree on their graph.



order of addition to shortest-path tree:

1	2	3	4	5	6	7	8
s	e	g	a	b	d	c	f

(b) 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.

Your answer is: ☑ Yes ☒ No. OK

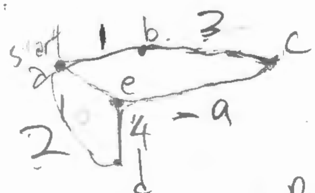
Proof: By the definition of Dijkstra's algorithm, as long as all edges e have distinct lengths greater than 0, no matter what source nodes you start from, you will always add a unique edge in a way that path from s to any other edge v is minimal. Because no edges have same length, every edge will be added in the same order every time a Dijkstra's algorithm is run on any starting node s, resulting a unique shortest-path tree every time.

(c) 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?'

Your answer is: ☐ Yes ☑ No.

Proof:



Counterexample. examine graph to the left. This graph does not contain a cycle w/ negative total length, but dijkstra's algorithm will add edge (a,b), then (a,d), then (b,c), then (c,e). But correct shortest path is (a,b), (b,c), (c,e), (e,d), which does not match Dijkstra's algorithm.

(d) 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.

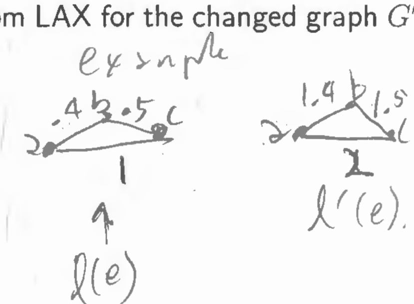
Kim complains that your shortest paths are not realistic, since air travel requires at least a one-hour layover in each airport, so you change the length $\ell(e)$ of every edge e in the graph to $\ell'(e) = \ell(e) + 1$ hour.

Is the tree T for G guaranteed to still be a shortest-path tree from LAX for the changed graph $G' = (V, E, \ell')$?

Your answer is: ☐ Yes ☑ No.

Proof: If you have flights that less than 1 hour, adding 1 hour makes big enough impact to shortest path. In $\ell(e)$, shortest path is (a, b) then (b, c), while in $\ell'(e)$, shortest path is (a, b), (a, c).

example



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 ☐ Tardos
Proof: Because Kruskal's algorithm first sorts edges in order of least to greatest, even if negative edges are present, if you add most negative edges first and then less negative edges before doing the normal Kruskal's algorithm, you can still obtain MST with just Kruskal's algorithm even with negative edge costs are present.

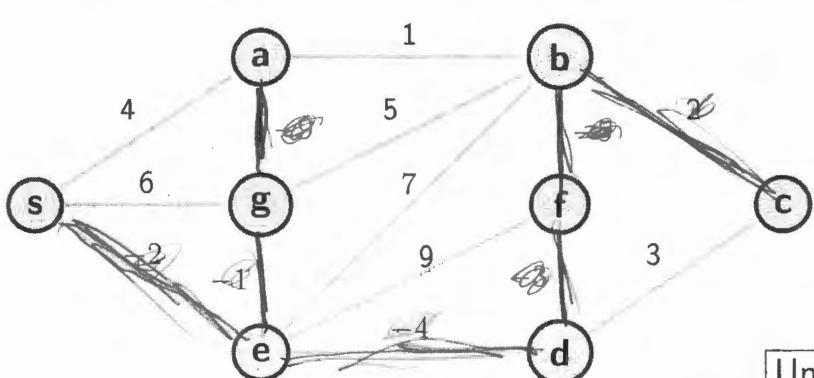
(b) **Maximum Spanning Trees (9 points)**
The company changes its specifications so that all edge costs c must satisfy $c \geq 1$, and it wants an algorithm to construct **Maximum Spanning Trees.** In other words, it wants an efficient algorithm that finds a spanning tree with the property that the sum 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? ☐ Kleinberg ☑ Tardos
Proof: Instead of adding edges from the smallest length to biggest length, if you simply add edges from the biggest length to smallest length, and then do the same Kruskal's algorithm, you can still obtain maximum spanning tree with same time complexity as MST's time complexity.

(c) **Overpaid Consultants (7 points)**
Kleinberg and Tardos cannot figure out a MST for the following graph.
Please draw a MST for them, and tell them whether or not it is unique.



Unique MST? ☑ Yes ☐ No

Review Questions (25 points) — 9

(a) **Changing Minimum Spanning Trees (3 points)**
For an undirected graph G with distinct (unique) edge costs, which of the following statements are true?

−1

- **True** ☑ **False** ☒ The MST could change if we change the cost of each edge e from $c(e)$ to $c(e)+2$.
- **True** ☐ **False** ☑ The MST could change if we change the cost of each edge e from $c(e)$ to $2\,c(e)$.
- **True** ☐ **False** ☑ The MST could change if we change the cost of each edge e from $c(e)$ to $c(e)^2$.

−4 (b) **DAGs (6 points)**

- **True** ☒ **False** ☑ A directed graph is a directed acyclic graph if and only if it can be topologically sorted.
- **True** ☑ **False** ☒ G is a directed acyclic graph if and only if G has a node with no incoming edges.
- **True** ☐ **False** ☑ A 'DFS tree' T starting from node s in an undirected graph G is sometimes a directed aycli graph that is not a tree.

−4 (c) **Undirected Graphs (8 points)**

- **True** ☐ **False** ☑ Suppose $G=(V,E)$ is an undirected, connected graph in which all vertices have even degree. Then G is bipartite.
- **True** ☑ **False** ☒ G is a bipartite graph if and only if G has no triangles (i.e., no three nodes are a clique).
- **True** ☑ **False** ☒ Suppose a weighted undirected graph G has a cycle C, and there is an edge e that is the unique least-cost edge in C. Then e is in every MST for G.
- **True** ☑ **False** ☐ If all edge lengths in an undirected graph G are a constant $c>0$, then for every source vertex s in G, a shortest path tree from s is the same as a BFS tree starting from s.

(d) **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)

From node s, run a breadth first search but do not mark any node as visited or unvisited. For each node, store the length of each path from s for each specific layer in an array, pathlength[k], where k = layer length. Do this until the BFS is completed. Then, for each vertex in G look at the stored array and the maximum value is the longest path from s to the node.

(e) **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.

If you run a DFS on a graph G and if every node starting from root node only has 1 children, then G has a Hamiltonian path.