Full Name: ____________________

Andrew ID: ____________ **Section:** ____________

15–210: Parallel and Sequential Data Structures and Algorithms

Practice Exam II (Solutions)

April 2016

- There are 16 pages in this examination, comprising 6 questions worth a total of 118 points. The last few pages are an appendix with costs of sequence, set and table operations.
- You have 80 minutes to complete this examination.
- Please answer all questions in the space provided with the question. Clearly indicate your answers.
- You may refer to your one double-sided $8\frac{1}{2} \times 11$in sheet of paper with notes, but to no other person or source, during the examination.

Circle the section YOU ATTEND

Sections		
A	9:30am - 10:20am	Edward/Angie
B	10:30am - 11:20am	Jake/Narain
C	12:30pm - 1:20pm	Sonya/Anisha
D	12:30pm - 1:20pm	Nick/Yongshan
E	1:30pm - 2:20pm	William/Bryan
F	1:30pm - 2:20pm	Sam S./Yutong
G	3:30pm - 4:20pm	Howard/Yongshan

Full Name: ______________________ **Andrew ID:** ______________________

Question	Points	Score
Short Answers	30	
Dijkstra and A*	15	
(Shortest Paths) Wormholes	10	
Strongly Connected Components	20	
(Dynamic Programming) Pipe Cutting	18	
MST and Tree Contraction	25	
Total:	118	

Question 1: Short Answers (30 points)

Please answer the following questions each with a few sentences, or a short snippet of code (either pseudocode or SML).

(a) (4 points) Consider an undirected graph G with unique positive weights. Suppose it has a minimum spanning tree T. If we square all the edge weights and compute the MST again, do we still get the same tree structure? Explain briefly.

> **Solution:** Yes we get the same tree. The minimum spanning tree only depends on the ordering among the edges. This is because the only thing we do with edges is compare them.

(b) (5 points) Lets say you are given a table that maps every student to the set of classes they take. Fill in the algorithm below that returns all classes, assuming there is at least one student in each class. Your algorithm must run in $O(m \log n)$ work and $O((\log m)(\log n))$ span, where n is the number of students and m is the sum of the number of classes taken across all students. Note, our solution is one line.

> **Solution:**
```
fun allClasses(T) = Table.reduce Set.union ∅ T
```

(c) (5 points) A new startup *FastRoute* wants to route information along a path in a communication network, represented as a graph. Each vertex represents a router and each edge a wire between routers. The wires are weighted by the maximum bandwidth they can support. *FastRoute* comes to you and asks you to develop an algorithm to find the path with maximum bandwidth from any source s to any destination t. As you would expect, the *bandwidth* of a path is the minimum of the bandwidths of the edges on that path; the minimum edge is the *bottleneck*.

Explain how to modify Dijkstra's algorithm to do this. In particular, how would you change the priority queue and the following relax step?

```
fun relax (Q, (u,v,w)) = PQ.insert (d(u) + w, v) Q
```

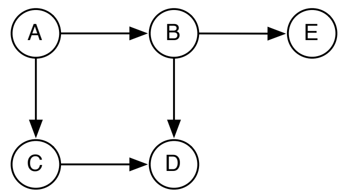
Justify your answer.

> **Solution:** We'll use a max priority queue instead of a min priority queue used in Dijkstra's. We will also modify the relax step to insert into the priority queue $\min(d(u), w)$ because the quality of a path is the minimum of the edge weights. These changes don't affect the correctness of Dijkstra's, so we could explore the vertices like in Dijkstra's.

(d) (5 points) Given a graph with integer edge weights between 1 and 5 (inclusive), you want to find the shortest *weighted* path between a pair of vertices. How would you reduce this problem to the shortest *unweighted* path problem, which can be solved using BFS?

Solution: Replace each edge with weight i with a simple path of i edges each with weight 1. Then solve with BFS.

(e) (5 points) Recall the implementation of DFS shown in class using the `discover` and `finish` functions. Circle the correct answer for each of the following statements, assuming DFS starts at A:



`discover D` could be called before `discover E`:	True	False
`discover E` could be called before `discover D`:	True	False
`discover D` could be called before `discover C`:	True	False
`finish A` could be called before `finish B`:	True	False
`finish D` could be called before `discover B`:	True	False

Solution: True, True, True, False, True

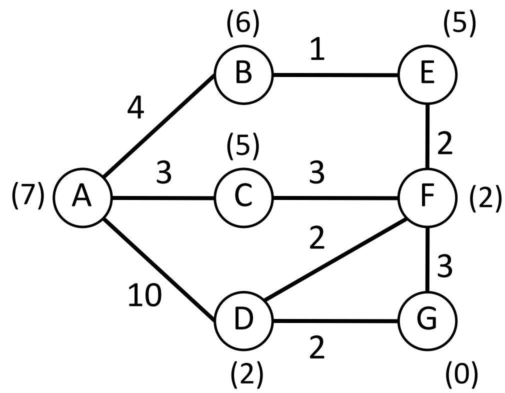
(f) (6 points) Circle *every* type of graph listed below for which star contraction will reduce the number of *edges* by a constant factor in expectation in every round until fully reduced (and hence imply $O(|E|)$ total work). You can assume redundant edges between vertices are removed.

(a) a graph in which all vertices have degree at most 2
(b) a graph in which all vertices have degree at most 3
(c) a graph in which all vertices have degree $\sqrt{|V|}$
(d) a graph containing a single cycle (i.e. a forest with one additional edge)
(e) the complete graph (i.e. an edge between every pair of vertices)
(f) any graph (still circle others if relevant)

Solution: a, d, e

Question 2: Dijkstra and A* (15 points)

(a) (6 points) Consider the graph shown below, where the edge weights appear next to the edges and the heuristic distances to vertex G are in parenthesis next to the vertices.



i. Show the order in which vertices are visited by Dijkstra when the source vertex is A.

Solution: A C B E F D G

ii. Show an order in which vertices are visited by A^* when the source vertex is A and the destination vertex is G.

Solution: A C F G

(b) (4 points) What is the key reason you would choose to use A^* instead of Dijkstra's algorithm?

Solution: You can use A^* if you want the shortest path to only a single goal vertex, and not all shortest paths. A^* can be much more efficient, as it tries to move toward the goal more directly, skipping many more vertices.

(c) (5 points) Show a 3-vertex example of a graph on which Dijkstra's algorithm always fails. Please clearly identify which vertex is the source.

Solution:

```
        A
       / \
x=4 /     \ y=-2   x+y < z < x guarantees failure
   /       \       x+y < z <= x may fail depending on the input order
  S ------- B
      z=3
```

Question 3: (Shortest Paths) Wormholes (10 points)

(a) (10 points) In your new job for a secret Government agency you have been told about the existence of wormholes (also known as Einstein-Rosen bridges) that connect various locations in the country. You have been tasked with designing an algorithm for finding the shortest path using a combination of roads and wormholes between a pair of locations. Traveling through a wormhole is instantaneous, for all practical purposes, but it turns out that on a given trip someone can only go through two wormholes otherwise they risk rearrangement of their atomic structure. The wormhole problem is therefore the weighted shortest path problem (assuming non-negative edge weights) with the additional constraint that

- Some edges are specially marked
- A path can take at most two of those edges

You still have your Dijkstra code from 210. You don't want to change your codeafter all you forgot how ML worksso you just want to preprocess your graph so that a call to your code $SP(s, t)$ returns the correct solution to the wormhole problem. Explain how to do this. **At most 5 sentences**.

Solution: Create three copies of the graph without the wormhole edges: copy 0, copy 1, and copy 2. Connect copy 0 with copy 1 with the wormhole edges, with weight 0. Likewise connect copy 1 with copy 2 with wormhole edges with weight 0. Now find the shortest path from $SP(s, d)$ by starting at s in copy 0 and finding the shortest path to d in any of the three copies.

Question 4: Strongly Connected Components (20 points)

In this question, you will write 2 functions on directed graphs. We assume that graphs are represented as:

```
type graph = vertexSet vertexTable
```

with key comparisons taking $O(1)$ work.

(a) (10 points) Given a directed graph $G = (V, E)$, its transpose G^T is another directed graph on the same vertices, with every edge flipped. More formally, $G^T = (V, E')$, where

$$E' = \{(b, a) \mid (a, b) \in E\}.$$

Here is a skeleton of an SML definition for `transpose` that computes the transpose of a graph. Fill in the blanks to complete the implementation. Your implementation must have $O(|E| \log |V|)$ work and $O(\log^2 |V|)$ span.

```
fun transpose (G : graph) : graph =
  let
    val S = vertexTable.toSeq(G)      (* returns (vertex*vertexSet) seq *)

    fun flip(u,nbrs) = Seq.map  (fn v => (v,u))  (vertexSet.toSeq nbrs)

    val ET = Seq.flatten(Seq.map flip S)

    val T = vertexTable. collect  ET
  in
    vertexTable.map   vertexSet.fromSeq   T
  end
```

(b) (10 points) A *strongly connected component* of a directed graph $G = (V, E)$ is a subset S of V such that every vertex $u \in S$ can reach every other vertex $v \in S$ (i.e., there is a directed path from u to v), and such that no other vertex in V can be added to S without violating this condition. Every vertex belongs to exactly one strongly connected component in a graph.

Implement the function:

```
val scc : graph * vertex -> vertexSet
```

such that `scc(G,v)` returns the strongly connected component containing v. You may assume the existence of a function:

```
val reachable : graph * vertex -> vertexSet
```

such that `reachable(G,v)` returns all the vertices reachable from v in G. Not including the cost of `reachable`, your algorithm must have $O(|E| \log |V|)$ work and $O(\log^2 |V|)$ span. You might find `transpose` useful and can assume the given time bounds.

```
fun scc (G : graph, v : vertex) : vertexSet =

    vertexSet.intersection(reachable(G,v),

                           reachable(transpose G,v))
```

Question 5: (Dynamic Programming) Pipe Cutting (18 points)

Sammy, the proprietor of your friendly neighborhood hardware store (as if they still existed) will cut a pipe at a cost proportional to its length. You have a pipe and have marked on it n places it needs to be cut. You want to figure out in what order to have your pipe cut so as to minimize your expenses. This can be solved with dynamic programming. Let A be a sequence of fragment lengths, in the order they appear along the pipe, and $w(A) = \sum_{a \in A} a$ (i.e. the sum of their lengths).

Note that a greedy method based on picking either the cut nearest the middle of the pipe, or the middle of the possible cuts does not work. Consider, for example cuts at locations .4, .55 and .7 along the pipe (i.e. $A = \langle\, 0.4, 0.15, 0.15, 0.3 \,\rangle$). In this case the best first cut is .4.

(a) (6 points) Give a recursive solution to the problem. It should not be more than 3 or 4 lines of pseudocode.

> **Solution:**
>
> **fun** `pipecut`$(A) =$
> **if** $|A| \leq 1$ **then** 0
> **else**
> $w(A) + \min_{k \in \{0,\ldots,|A|-2\}} \{\texttt{pipecut}(A \langle\, 0, \ldots, k \,\rangle) + \texttt{pipecut}(A \langle\, k+1, \ldots, |A|-1 \,\rangle)\}$

(b) (4 points) How many distinct calls are there?

> **Solution:** There are no more than $n(n+1)/2$ distinct contiguous subsequences, and hence at most that many distinct arguments to `pipecut`.

(c) (4 points) What is the total work assuming sharing on the DAG.

> **Solution:** Each call does $O(n)$ work, so the total work is $O(n^3)$.

(d) (4 points) What is the total span.

> **Solution:** Each call has span $O(\log n)$, and the depth of the DAG is $O(n)$ so the total span is $O(n \log n)$.

Question 6: MST and Tree Contraction (25 points)

In *SegmentLab*, you implemented Borůvka's algorithm that interleaved star contractions and finding minimum weight edges. In this question you will analyze Borůvka's algorithm more carefully.

We'll assume throughout this problem that the edges are undirected, and each edge is labeled with a unique identifier (ℓ). The weights of the edges do not need to be unique, and $m = |E|$ and $n = |V|$.

% *returns the set of edges in the minimum spanning tree of* G
function `MST`$(G = (V, E)) =$
  if $|E| = 0$ **then** $\{\}$
  else let
    val $F = \{$*min weight edge incident on* $v : v \in V\}$
    val $(V', P) =$*contract each tree in the forest* (V, F) *to a single vertex*
        $V' =$ *remaining vertices*
        $P =$ *mapping from each* $v \in V$ *to its representative in* V'
    val $E' = \{(P_u, P_v, \ell) : (u, v, \ell) \in E \mid P_u \neq P_v\}$
  in
    `MST`$(G' = (V', E')) \cup \{\ell : (u, v, \ell) \in F\}$
  end

(a) (4 points) Show an example graph with 4 vertices in which F will not include all the edges of the MST.

Solution:

```
      3
   o --- o
 1 |     | 2
   o     o
```

(b) (4 points) Prove that the set of edges F must be a forest (i.e. F has no cycle).

Solution: Answer 1: The MST does not have a cycle (it is a tree) and F is a subset of F so it can't have a cycle.

Answer 2: AFSOC that there is a cycle. Consider the maximum weight edge on the cycle. Neither of its endpoints will choose it since they both have lighter edges. Contradiction.

(c) (4 points) Suggest a technique to efficiently contract the forest in parallel. What is a tight asymptotic bound for the work and span of your contract, in terms of n? Explain briefly. Are these bounds worst case or expected case?

> **Solution:** Use star contraction as described in class. Since in contraction a tree will always stay a tree, the number of edges must go down with the number of vertices. Therefore total work will be $O(n)$ and span will be $O(\log^2 n)$ in expectation.

(d) (4 points) Argue that each recursive call to `MST` removes, in the worst case, at least *half* of the vertices; that is, $|V'| \leq \frac{|V|}{2}$.

> **Solution:** Every vertex will join at least one other vertex. Since edges have two directions, at least $n/2$ of them must be selected, which will remove at least $n/2$ vertices ($n = |V|$).

(e) (4 points) What is the maximum number of edges that could remain after one step (i.e. what is $|E'|$)? Explain briefly.

> **Solution:** $m - n/2$ since at least $n/2$ edges are removed, as described in previous answer.

(f) (5 points) What is the expected work and span of the overall algorithm in terms of m and n? Explain briefly. You can assume that calculating F takes $O(m)$ work and $O(\log n)$ span.

> **Solution:** Since vertices go down by at least a factor of $1/2$ on each round, there will be at most $\log n$ rounds. The cost of each round is dominated by calculating F, $O(m)$ work and $O(\log n)$ span and the contraction of forests $O(n)$ work and $O(\log^2 n)$ span. Multiplying the max of each of these by $\log n$ gives $O(m \log n)$ work and $O(\log^3 n)$ span.

Appendix: Library Functions

```
signature SEQUENCE =
sig
  type 'a t
  type 'a seq = 'a t
  type 'a ord = 'a * 'a -> order
  datatype 'a listview = NIL | CONS of 'a * 'a seq
  datatype 'a treeview = EMPTY | ONE of 'a | PAIR of 'a seq * 'a seq

  exception Range
  exception Size

  val nth : 'a seq -> int -> 'a
  val length : 'a seq -> int
  val toList : 'a seq -> 'a list
  val toString : ('a -> string) -> 'a seq -> string
  val equal : ('a * 'a -> bool) -> 'a seq * 'a seq -> bool

  val empty : unit -> 'a seq
  val singleton : 'a -> 'a seq
  val tabulate : (int -> 'a) -> int -> 'a seq
  val fromList : 'a list -> 'a seq

  val rev : 'a seq -> 'a seq
  val append : 'a seq * 'a seq -> 'a seq
  val flatten : 'a seq seq -> 'a seq

  val filter : ('a -> bool) -> 'a seq -> 'a seq
  val map : ('a -> 'b) -> 'a seq -> 'b seq
  val zip : 'a seq * 'b seq -> ('a * 'b) seq
  val zipWith : ('a * 'b -> 'c) -> 'a seq * 'b seq -> 'c seq

  val enum : 'a seq -> (int * 'a) seq
  val filterIdx : (int * 'a -> bool) -> 'a seq -> 'a seq
  val mapIdx : (int * 'a -> 'b) -> 'a seq -> 'b seq
  val update : 'a seq * (int * 'a) -> 'a seq
  val inject : 'a seq * (int * 'a) seq -> 'a seq

  val subseq : 'a seq -> int * int -> 'a seq
  val take : 'a seq -> int -> 'a seq
  val drop : 'a seq -> int -> 'a seq
  val splitHead : 'a seq -> 'a listview
  val splitMid : 'a seq -> 'a treeview



  val iterate : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b
  val iteratePrefixes : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq * 'b
  val iteratePrefixesIncl : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq
  val reduce : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a
  val scan : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq * 'a
  val scanIncl : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq

  val sort : 'a ord -> 'a seq -> 'a seq
  val merge : 'a ord -> 'a seq * 'a seq -> 'a seq
  val collect : 'a ord -> ('a * 'b) seq -> ('a * 'b seq) seq
```

```
  val collate : 'a ord -> 'a seq ord
  val argmax : 'a ord -> 'a seq -> int

  val $ : 'a -> 'a seq
  val % : 'a list -> 'a seq
end
```

ArraySequence	Work	Span

`empty ()`
`singleton a`
`length s`
`nth s i`
`subseq s (i, len)`
	$O(1)$	$O(1)$

`tabulate f n`
if $f(i)$ has W_i work and S_i span
`map f s`
if $f(s[i])$ has W_i work and S_i span, and $\|s\| = n$
`zipWith f (s, t)`
if $f(s[i], t[i])$ has W_i work and S_i span, and $\min(\|s\|, \|t\|) = n$
	$O\left(\sum_{i=0}^{n-1} W_i\right)$	$O\left(\max_{i=0}^{n-1} S_i\right)$

`reduce f b s`
if `f` does constant work and $\|s\| = n$
`scan f b s`
if `f` does constant work and $\|s\| = n$
`filter p s`
if `p` does constant work and $\|s\| = n$
	$O(n)$	$O(\lg n)$

`flatten s`	$O\left(\sum_{i=0}^{n-1} \left(1 + \|s[i]\|\right)\right)$	$O(\lg \|s\|)$
`sort cmp s` if `cmp` does constant work and $\|s\| = n$	$O(n \lg n)$	$O(\lg^2 n)$
`merge cmp (s, t)` if `cmp` does constant work, $\|s\| = n$, and $\|t\| = m$	$O(m + n)$	$O(\lg(m + n))$
`append (s,t)` if $\|s\| = n$, and $\|t\| = m$	$O(m + n)$	$O(1)$

```
signature TABLE =
sig
  structure Key : EQKEY
  structure Seq : SEQUENCE

  type 'a t
  type 'a table = 'a t

  structure Set : SET where Key = Key and Seq = Seq

  val size : 'a table -> int
  val domain : 'a table -> Set.t
  val range : 'a table -> 'a Seq.t
  val toString : ('a -> string) -> 'a table -> string
  val toSeq : 'a table -> (Key.t * 'a) Seq.t

  val find : 'a table -> Key.t -> 'a option
  val insert : 'a table * (Key.t * 'a) -> 'a table
  val insertWith : ('a * 'a -> 'a) -> 'a table * (Key.t * 'a) -> 'a table
  val delete : 'a table * Key.t -> 'a table

  val empty : unit -> 'a table
  val singleton : Key.t * 'a -> 'a table
  val tabulate : (Key.t -> 'a) -> Set.t -> 'a table
  val collect : (Key.t * 'a) Seq.t -> 'a Seq.t table
  val fromSeq : (Key.t * 'a) Seq.t -> 'a table

  val map : ('a -> 'b) -> 'a table -> 'b table
  val mapKey : (Key.t * 'a -> 'b) -> 'a table -> 'b table
  val filter : ('a -> bool) -> 'a table -> 'a table
  val filterKey : (Key.t * 'a -> bool) -> 'a table -> 'a table

  val reduce : ('a * 'a -> 'a) -> 'a -> 'a table -> 'a
  val iterate : ('b * 'a -> 'b) -> 'b -> 'a table -> 'b
  val iteratePrefixes : ('b * 'a -> 'b) -> 'b -> 'a table -> ('b table * 'b)

  val union : ('a * 'a -> 'a) -> ('a table * 'a table) -> 'a table
  val intersection : ('a * 'b -> 'c) -> ('a table * 'b table) -> 'c table
  val difference : 'a table * 'b table -> 'a table

  val restrict : 'a table * Set.t -> 'a table
  val subtract : 'a table * Set.t -> 'a table

  val $ : (Key.t * 'a) -> 'a table
end
```

```
signature SET =
sig
  structure Key : EQKEY
  structure Seq : SEQUENCE

  type t
  type set = t

  val size : set -> int
  val toString : set -> string
  val toSeq : set -> Key.t Seq.t

  val empty : unit -> set
  val singleton : Key.t -> set
  val fromSeq : Key.t Seq.t -> set

  val find : set -> Key.t -> bool
  val insert : set * Key.t -> set
  val delete : set * Key.t -> set

  val filter : (Key.t -> bool) -> set -> set

  val reduceKey : (Key.t * Key.t -> Key.t) -> Key.t -> set -> Key.t
  val iterateKey : ('a * Key.t -> 'a) -> 'a -> set -> 'a

  val union : set * set -> set
  val intersection : set * set -> set
  val difference : set * set -> set

  val $ : Key.t -> set
end
```

MkTreapTable	Work	Span
`size` T	$O(1)$	$O(1)$
`filter` f T `map` f T	$\sum_{(k\mapsto v)\in T} W(f(v))$	$\lg \lvert T\rvert + \max_{(k\mapsto v)\in T} S(f(v))$
`tabulate` f X	$\sum_{k\in X} W(f(k))$	$\max_{k\in X} S(f(k))$
`reduce` f b T if f does constant work	$O(\lvert T\rvert)$	$O(\lg \lvert T\rvert)$

`insertWith` f `(`T`,(`k`,`v`))`
if f does constant work
`find` T k
`delete (`T`,`k`)`
	$O(\lg \lvert T\rvert)$	$O(\lg \lvert T\rvert)$

`domain` T `range` T `toSeq` T	$O(\lvert T\rvert)$	$O(\lg \lvert T\rvert)$
`collect` S `fromSeq` S	$O(\lvert S\rvert \lg \lvert S\rvert)$	$O(\lg^2 \lvert S\rvert)$

For each argument pair (A, B) below, let $n = \max(|A|, |B|)$ and $m = \min(|A|, |B|)$.

MkTreapTable	Work	Span

`union` f `(`X`,`Y`)`
`intersection` f `(`X`,`Y`)`
`difference (`X`,`Y`)`
`restrict (`T`,`X`)`
`subtract (`T`,`X`)`
	$O\big(m \lg(\frac{n+m}{m})\big)$	$O\big(\lg(n+m)\big)$