Full Name: _____

Andrew ID: _____ **Section:** _____

15–210: Parallel and Sequential Data Structures and Algorithms

Exam II (Solutions)

7 April 2017

- **Verify** There are 17 pages in this examination, comprising 7 questions worth a total of 100 points. The last 2 pages are an appendix with costs of sequence, set and table operations.
- **Write** the name (e.g., "J. Snow") of the persons sitting to your left and to your right below your andrew id (in left to right order).
- **Time:** You have 80 minutes to complete this examination.
- **Goes without saying:** Please answer all questions in the space provided with the question. Clearly indicate your answers.
- **Beware:** 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.
- **Primitives:** In your algorithms you can use any of the primitives that we have covered in the lecture, unless otherwise states. A reasonably comprehensive list is provided at the end and sometimes in the body of the question itself.
- **Code:** When writing your algorithms, you can use ML syntax or the pseudocode notation used in the notes or in class. In the questions, we use pseudocode.
- **Good luck!**

Sections

A	9:30am - 10:20am	Andra/Charles
B	10:30am - 11:20am	Aashir/Anatol
C	12:30pm - 1:20pm	Oliver
D	12:30pm - 1:20pm	Rohan/Serena
E	1:30pm - 2:20pm	John/Christina
F	1:30pm - 4:20pm	Vivek/Teddy
G	3:30pm - 5:20pm	Ashwin/Sunny

Question	Points	Score
BSTs and Treaps	22	
A Sparse Problem	10	
(BFS) Multisource Shortest Paths	12	
DFS	18	
Shortest Paths	13	
Contraction Action, what's Your Reaction?	15	
Parallel Bridges	10	
Total:	100	

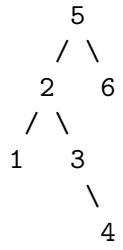
Question 1: BSTs and Treaps (22 points)

(a) (6 points) Suppose we have the keys $1, 2, 3, 4, 5, 6$ with priorities p shown below:

key	1	2	3	4	5	6
p(key)	2	5	4	1	6	3

Draw the **max**-treap (requires that priority at a node is greater than the priority of its two children) associated with these keys and priorities.

Solution: Recall that Treaps are unique with a given set of keys and priorities. The only possible solution is:



(b) (6 points) What is the probability that a treap with n nodes has depth n? You can assume that the priorities are picked uniformly at random, and that there are no duplicates.

Solution:

$$\frac{2^{n-1}}{n!}$$

(c) (10 points) In lecture you saw an implementation of the `union` operation on sets represented as binary search trees. This problem involves the set `intersect` operation.

To put this code in context, we're using a binary search tree of the following type to represent these sets.

datatype `BST = Leaf | Node of (BST * BST * key)`

Also, recall the following type signatures of `split` and `joinMid`:

`split`(T, k) `: BST` $\times$ `key` $\rightarrow$ `BST` $\times$ `bool` $\times$ `BST`

`joinMid`(L, m, R) `: BST` $\times$ `key` $\times$ `BST` $\rightarrow$ `BST`

Fill in the blanks in the `SML` code below that implements `intersect`.

```
intersect (T1, T2) =
   case T1 of
      Leaf => _____________________________
    | Node (L1, R1, k1) =>
        let (L2, x2, R2) = split(T2, k1)
            (L, R) = par( _____________________________,
                          _____________________________ )
        in
           case x2 of
              NONE => _____________________________
            | SOME(a) => _____________________________
        end
```

Solution:

```
fun intersect (T1,T2) =
   case T1 of
      Leaf => Leaf
    | Node(L1,R1,k1) =>
        let val (L2,x2,R2) = split(T2,k1);
            val (L,R) = par(intersect(L1,L2), intersect(R1,R2))
        in
           case x2 of
              NONE => join(L,NONE,R)
            | SOME(_) => join(L,x2,R)
        end
```

Question 2: A Sparse Problem (10 points)

A vector is basically a sequence whose values are real numbers, for example:

$$\langle\, 2.1, 5.3, 1.2, -2.55, 11.0 \,\rangle$$

Recall that the dot-product of two vectors is an elementwise multiplication followed by a sum. With sequences this can be implemented as:

$$\texttt{dotProduct}(A, B) \;=\; \texttt{reduce}\ +\ 0\ \langle\, A[i] \times B[i] : 0 \leq i < |A| \,\rangle$$

A sparse vector is one in which the value is 0 at most indices. Sparse vectors are common in practice and can be represented more efficiently as a mapping from the non-zero indices to their values. For example the vector

$$\langle\, 0.0,\ 2.1,\ 0.0,\ 1.2,\ 0.0,\ 0.0,\ 0.0,\ 11.0,\ 0.0,\ 0.0 \,\rangle$$

can be represented as

$$\{1 \mapsto 2.1, 3 \mapsto 1.2, 7 \mapsto 11.0\}$$

Your job is to implement `dotProduct(A,B)` based on this sparse representation. You can use any of the functions on tables, but the implementation must take $O(m \log(1 + n/m))$ work and $O(\log n)$ span, where $m = \min(|A|, |B|)$ and $n = \max(|A|, |B|)$, and $|A|$ and $|B|$ indicates the number of non-zero elements (i.e. the size of the mapping). Please note that the signature for the Table interface is at the end of the exam. Our solution is one line.

```
dotProduct (A, B) =  reduce + 0 (intersection * A B)
```

Question 3: (BFS) Multisource Shortest Paths (12 points)

You are starting a company that supplies maps indicating the proximity of every intersection in Pittsburgh to the nearest coffee shop. This would be useful if you're buying coffee in a hurry. You should assume that all road segments are the same length, but some are one-way streets. This problem can be defined as follows:

> **Definition** (The multisource unweighted shortest pathlength (MUSP) problem): Given a *directed* graph $G = (V, E)$ and a set of sources $U \subseteq V$ determine for every vertex in $v \in V$ its shortest path length **from** any vertex in U, denoted as $\delta(U, v)$.

Please answer the following:

(a) (6 points) Describe an algorithm that solves this problem in $O(m \lg n)$ work and $O(d \log^2 n)$ span, where $n = |V|$, $m = |E|$, and $d = \max_{v \in V} \delta(U, v)$. A couple sentences should suffice—you may justify the cost by relating it to the cost of some algorithm we covered in class.

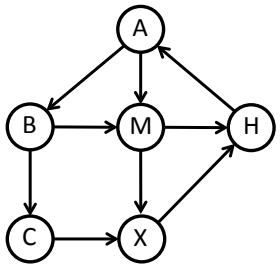
> **Solution:** Use breadth first search with U as the initial frontier. The bounds are the same as given in lecture for BFS.

(b) (6 points) Suppose now you would like to indicate the shortest path length from every intersection to the nearest coffee shop. That is to say, you want for every vertex in V its shortest path length **to** any vertex in U. How would you modify your algorithm above so that it has the same cost bounds, except that $d = \max_{v \in V} \delta(v, U)$? Give a short justification.

> **Solution:** Reverse the edges in G and do the same as above. Reversing can be done with a flatten and collect, which are within the cost bounds given above.

Question 4: DFS (18 points)

(a) A *DFS order* is a sequence of vertices of a graph in the order in which they are first visited by *depth-first* search (DFS). Consider the following graph:



One DFS order of this graph is $\langle A, B, C, X, H, M \rangle$.

i. (6 points) List ALL other possible DFS orders for this graph, given source vertex A.

Solution: A B M X H C
A B M H X C
A M H X B C
A M X H B C

ii. (6 points) **For the order we gave you**, list the *forward*, the *back*, and the *cross* edges.

Solution: Forward: (A, M)
Cross: $(M, X), (M, H)$
Back: (H, A)

(b) (6 points) Suppose you perform DFS on an undirected graph G where G is a clique (there is an edge between every two vertices). In the resulting DFS tree, how many nodes will have degree greater than 2?

Question 5: Shortest Paths (13 points)

(a) (2 points) **TRUE** or **FALSE**: If you add a positive constant to the weight on every edge in a weighted graph, it does not change the shortest path tree.

Solution: FALSE

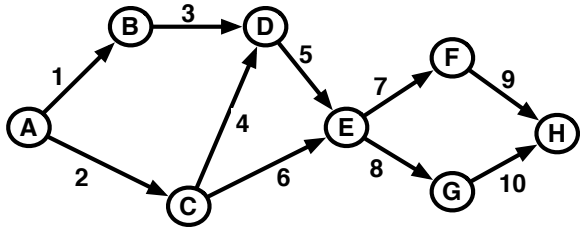
(b) (2 points) **TRUE** or **FALSE**: If a graph has negative edge weights, Dijkstra's algorithm for shortest path may loop forever.

Solution: FALSE

(c) (2 points) **TRUE** or **FALSE**: We can use the Bellman-Ford algorithm to detect negative cycles in a graph.

Solution: TRUE

(d) (7 points) Given the following graph:



Show the order that the vertices are visited in by Dijkstra's algorithm, the distance computed from the source A, and the contents of the priority queue after the vertex is visited (order in priority queue does not matter).

A	1	0	(2,C),(1,B)
B	2	1	(2,C),(4,D)
C	3	2	(4,D),(6,D),(8,E)
D	4	4	(6,D),(9,E),(8,E)
E	5	8	(9,E),(15,F),(16,G)
F	6	15	(24,H),(16,G)
G	7	16	(24,H),(26,H)
H	8	24	(26,H)

Question 6: Contraction Action, what's Your Reaction? (15 points)

For each graph property below, state whether or not *edge contraction* preserves this property. That is, if a graph has the property before performing an edge contraction, will the graph necessarily have the property after the contraction? You can assume that duplicate edges are removed.

Answer **Yes** or **No**. Also, if your answer is **No**, then give a counterexample. That is, give a graph G and highlight an edge e in G such that G has the given property initially but not after contracting e.

(a) (3 points) All nodes in the graph have degree ≤ 2.

Solution: Yes

(b) (3 points) All nodes in the graph have degree ≤ 3.

Solution: No:

```
\_/  contract middle edge
/ \
```

(c) (3 points) The graph is connected.

Solution: Yes

(d) (3 points) The graph is a tree (single nodes count as trees)

Solution: Yes

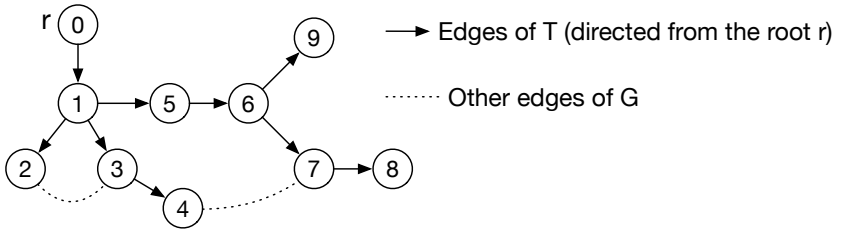
(e) (3 points) The graph has more edges than vertices.

Solution: No. A complete graph on 4 vertices (6 edges) will contract to a complete graph on 3 vertices (3 edges)

Question 7: Parallel Bridges (10 points)

In BridgeLab you found bridges of a graph using DFS, sequentially. Recall that a bridge in a undirected connected graph is any edge that does not belong to any cycles. It is also possible to do it in parallel in linear work using the following approach—each step can be parallelized.

1. Find a spanning tree T of G.
2. Select one vertex r to be the root of the tree T
3. Give an preorder numbering (**preNum**) to the nodes in the T starting at the root r. Recall that a pre-order numbering first numbers the root, then recursively numbers each of the subtrees. Here is an example:



4. For every vertex, determine the minimum and maximum pre-order number of its neighbors in G, not including its parent in T. In the example, 3s min is 2 and max is 4.
5. For each vertex, determine
   - **minSubNgh**: the minimum neighbor in its subtree (the minimum in the subtree of the minimum values calculated in the previous step),
   - **maxSubNgh**: the maximum neighbor in its subtree (the maximum in the subtree of the maximum values calculated in the previous step),
   - **maxPreNum**: the maximum pre-order number of its subtree

   For the example graph:
   minSubNgh[3] = 2, **maxSubNgh**[3] = 7, **maxPreNum**[3] = 4,
   minSubNgh[5] = 4, **maxSubNgh**[5] = 9, **maxPreNum**[5] = 9.

(a) (4 points) Argue in one or two sentences that if an edge (u, v) is a bridge then it must be in the spanning tree found in step 1.

> **Solution:** If there is is no cycle involving (u, v) then the only way to connect u and v is with the edge between them, and therefore it must be in the spanning tree.

(b) (6 points) Given an edge $(u, v) \in T$, with u the parent, write an equation for determining if it is a bridge based on the values of **preNum**[u], **preNum**[v], **minSubNgh**[u], **minSubNgh**[v], **maxSubNgh**[u], **maxSubNgh**[v], **maxPreNum**[u], and **maxPreNum**[v].

> **Solution:** (u, v) is a bridge if **minSubNgh**[v] $\geq$ **preNum**[v] and **maxSubNgh**[v] $\leq$ **maxPreNum**[v].

Scratch Work:

Scratch Work:

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`)`
	$O\big(m \lg(\frac{n+m}{m})\big)$	$O\big(\lg(n+m)\big)$