15–210: Parallel and Sequential Data Structures and Algorithms

Practice Exam II

April 2016

- There are 17 pages in this examination, comprising 6 questions worth a total of 116 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\(\frac{1}{2}\)\(\times\)11in sheet of paper with notes, but to no other person or source, during the examination.

Circle the section YOU ATTEND

<table>
<thead>
<tr>
<th>Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>Question</td>
</tr>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Short Answers</td>
</tr>
<tr>
<td>(Sets and Tables) Bingled</td>
</tr>
<tr>
<td>Dijkstra and A*</td>
</tr>
<tr>
<td>(Shortest Paths) Wormholes</td>
</tr>
<tr>
<td>Strongly Connected Components</td>
</tr>
<tr>
<td>MST and Tree Contraction</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
</tr>
</tbody>
</table>
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.

(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.

(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.
(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?

(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:

![Graph Diagram]

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

(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)
Question 2: (Sets and Tables) Bingled  (16 points)

After forming your company Bingle to index the web allowing word searches based on logical combination of terms (e.g. “big” and “small”), you discover that there are already a couple companies out there that do it....and lo-and-behold, they even have similar names. You therefore decide to extend yours with additional features. In particular you want to support phrase queries: e.g. find all documents where “fun algorithms” appears.

You decide the right way to represent the index is as a table of sets where the keys of the table are strings (i.e. the words) and the elements of the sets are pairs of values consisting of a document identifier and an integer location in the document where the string appears. So, for example the following collection of three documents with integer document identifiers:

\[
\langle (1, \text{“the big dog"}), (2, \text{“a big dog ate a hat"}), (3, \text{“i read a big book”}) \rangle
\]

the document index would be represented as

\[
\text{idx} = \{ \text{“a”} \mapsto \{(2,0),(2,4),(3,2)\}, \\
\text{“big”} \mapsto \{(1,1),(2,1),(3,3)\}, \\
\text{“dog”} \mapsto \{(1,2),(2,2)\}, \\
\ldots \}
\]

In particular you want to support the following interface

\[
\text{INDEX} = \text{sig} \begin{array}{ll}
\text{type} & \text{word} = \text{string} \\
\text{type} & \text{docId} = \text{int} \\
\text{type} & \text{index} = \text{docIdIntSet wordTable} \\
\end{array}
\]

(* represents all documents and all locations where a phrase appears *)

\[
\text{type} \ \text{docList}
\]

\[
\text{val} \ \text{makeIndex} : (\text{docId} * \text{string}) \ \text{seq} \rightarrow \text{index}
\]

\[
\text{val} \ \text{find} : \text{index} \rightarrow \text{word} \rightarrow \text{docList}
\]

\[
\text{val} \ \text{adj} : \text{docList} * \text{docList} \rightarrow \text{docList}
\]

\[
\text{val} \ \text{toSeq} : \text{docList} \rightarrow \text{docId seq}
\]

end

where, given an index I, \text{toSeq} (\text{adj (find I "210", find I "rocks"))} would return a sequence of identifiers of documents where “210” appears immediately before “rocks”, and

\[
\text{toSeq} (\text{adj (find I "Phil", adj (find I "loves", find I "cats"))})
\]

would return a sequence of identifiers of documents where the phrase “Phil love cats” appears.
(a) (8 points) Show SML code to generate the index from the sequence of documents. It
should not be more than 8 lines of code and assuming all words have length less than
some constant, must run in $O(n \log n)$ work and $O(\log^2 n)$ span, where $n$ is the total
number of words across all documents. A function to break a string into words has been
given to you.

```sml
type index = docIdIntSet wordTable

fun makeIndex (docs : (docId * string) seq) : index =
    let
        fun toWords str = Seq.tokens (fn c ⇒ not (Char.isAlphaNum c)) str
```

(b) (8 points) Define the `docList` type and implement the function `adj` as defined above. You
might find the function `setmap` useful. The solution should only be a few lines of code.

```sml
fun setmap f s = Set.fromSeq (Seq.map f (Set.toSeq s))

type docList =

fun adj ( , ) : docList =
```
Question 3: 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.

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

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

(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.
Question 4: (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 code after all you forgot how ML works so 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.
Question 5: Strongly Connected Components  (20 points)

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

```ml
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.

```ml
fun transpose (G : graph) : graph =
  let
    val S = vertexTable.toSeq(G) (* returns (vertex*vertexSet) seq *)
    fun flip(u,nbrs) = Seq.map (______________________________) (vertexSet.toSeq nbrs)
    val ET = Seq.flatten(Seq.map flip S)
    val T = vertexTable._____________ ET
  in
    vertexTable.map ____________________________ 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:

\[
\text{fun scc \( \): graph * vertex \rightarrow vertexSet}
\]

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

\[
\text{val reachable \( \): graph * vertex \rightarrow vertexSet}
\]

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

\[
\text{fun scc (G : graph, v : vertex) : vertexSet =}
\]

\[
\]

\[
\]
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 $n = |V|$.

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

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

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

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

(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.
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

<table>
<thead>
<tr>
<th>ArraySequence</th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty ()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>singleton a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>length s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nth s i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subseq s (i, len)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tabulate f n</td>
<td>$O \left( \sum_{i=0}^{n-1} W_i \right)$</td>
<td>$O \left( \max_{i=0}^{n-1} S_i \right)$</td>
</tr>
<tr>
<td>map f s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zipWith f (s, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reduce f b s</td>
<td>$O(n)$</td>
<td>$O(\lg n)$</td>
</tr>
<tr>
<td>scan f b s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filter p s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flatten s</td>
<td>$O \left( \sum_{i=0}^{n-1} (1 +</td>
<td>s[i]</td>
</tr>
<tr>
<td>sort cmp s</td>
<td>$O(n \lg n)$</td>
<td>$O(\lg^2 n)$</td>
</tr>
<tr>
<td>merge cmp (s, t)</td>
<td>$O(m + n)$</td>
<td>$O(\lg(m + n))$</td>
</tr>
<tr>
<td>append (s, t)</td>
<td>$O(m + n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th><strong>MkTreapTable</strong></th>
<th><strong>Work</strong></th>
<th><strong>Span</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>size $T$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>filter $f T$</td>
<td>$\sum_{(k\mapsto v) \in T} W(f(v)) \lg</td>
<td>T</td>
</tr>
<tr>
<td>map $f T$</td>
<td>$\sum_{k \in X} W(f(k)) \max_{k \in X} S(f(k))$</td>
<td></td>
</tr>
<tr>
<td>tabulate $f X$</td>
<td>$\sum_{k \in X} W(f(k)) \max_{k \in X} S(f(k))$</td>
<td></td>
</tr>
<tr>
<td>reduce $f b T$</td>
<td>$O(</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>if $f$ does constant work</td>
<td>*</td>
</tr>
<tr>
<td>insertWith $f$</td>
<td>$O(\lg</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>$(T, (k, v))$</td>
<td></td>
</tr>
<tr>
<td>find $T k$</td>
<td>$O(\lg</td>
<td>T</td>
</tr>
<tr>
<td>delete $(T, k)$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>domain $T$</td>
<td>$O(</td>
<td>T</td>
</tr>
<tr>
<td>range $T$</td>
<td>$O(</td>
<td>T</td>
</tr>
<tr>
<td>toSeq $T$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>collect $S$</td>
<td>$O(</td>
<td>S</td>
</tr>
<tr>
<td>fromSeq $S$</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th><strong>MkTreapTable</strong></th>
<th><strong>Work</strong></th>
<th><strong>Span</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>union $f (X, Y)$</td>
<td>$O(m \lg \left( \frac{O(\lg(n+m))}{m} \right))$</td>
<td></td>
</tr>
<tr>
<td>intersection $f (X, Y)$</td>
<td>$O(m \lg \left( \frac{O(\lg(n+m))}{m} \right))$</td>
<td></td>
</tr>
<tr>
<td>difference $(X, Y)$</td>
<td>$O(m \lg \left( \frac{O(\lg(n+m))}{m} \right))$</td>
<td></td>
</tr>
<tr>
<td>restrict $(T, X)$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>subtract $(T, X)$</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>