15–210: Parallel and Sequential Data Structures and Algorithms

Practice Midterm II

- There are 8 pages in this examination, comprising 5 questions worth a total of 100 points.
- 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.
- Your answers for this exam must be written in blue or black ink.

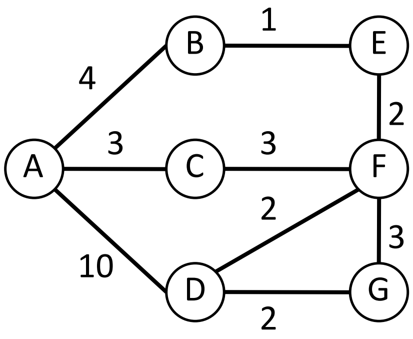
Full Name: ______________________________

Andrew ID: ________________ Section: ________________

Question	Points	Score
Graphs	15	
Short Answers	20	
Set Operations	20	
MST and Tree Contraction	25	
Treaps	20	
Total:	100	

Question 1: Graphs (15 points)

(a) (6 points) Consider the graph shown below. The edge weights appear next to the edges.



i. Show the order in which vertices are visited by Dijkstra's single source shortest path algorithm when the source vertex is A.

ii. Show an order in which vertices are visited by Prim's minimum spanning tree algorithm when the source vertex is A.

(b) (4 points) What is a key reason you would choose to use Bellman-Ford algorithm instead of Dijkstra's algorithm?

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

Question 2: Short Answers (20 points)

Please answer the following questions each with a few sentences, or a short snippet of code (either pseudocode or SML code). It has to fit in the given space. You will be graded on clarity as well as correctness.

(a) (7 points) Consider an undirected graph G with unique positive weights. Suppose it has a minimum spanning tree (MST) T. If we square all the edge weights and compute the MST again, do we still get the same tree structure? Explain briefly.

(b) (6 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 bandwith from any source s to any destinatin t. As you would expect, the bandwidth of a path is the minimum of the bandwith of the edges on that path—the minimum edge is the bottleneck.

Explain how to modify Disjstra 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.

(c) (7 points) Given a forest F (a collection of trees) with n vertices, how would you modify the tree contraction algorithm to return the number of trees (i.e. count the number of connected components) in F? Your algorithm should run in $O(n)$ work and $O(\log^2 n)$ span.

Question 3: Set Operations (20 points)
We can represent ordered sets of integers using binary search trees by the type

```
datatype bst = Leaf | Node of (bst * bst * int)
```

paired with the functions

```
split :  (bst * int) -> bst * bool * bst
join :  (bst * int option * bst) -> bst
```

where `split(T,k)` returns a pair of trees from `T` (less than and greater than `k`) and a flag indicating if `k` is in `T`.

Joe Twoten noticed the course staff kept writing almost the same code to implement set union, set intersection and set difference. He decided a more elegant solution would be to use higher order functions and just write the bulk of the code once. Typical of Joe, he only typed in part of the solution and left the rest up to you.

(a) (8 points) Consider a function

```
combine :  (bool * (int * bool -> int option)) -> (bst * bst) -> bst
```

where

- `combine (true,f) (Leaf,T2)` evaluates to `T2`
- `combine (false,f) (Leaf,T2)` evaluates to `Leaf`
- `combine (b,f) (T1,T2)` evaluates to a `bst` where every key `k` that appears in `T1` is replaced by the result of applying `f` to `k` and a boolean indicating whether `k` appears in `T2`. Every key that appears only in `T2` is handled as specified by `b` in the base case.

Most of the implementation of `combine` is provided below. Finish it by filling in the blanks.

```
fun combine (keep :  bool, f :  int * bool -> int option)
            (T1:  bst, T2:  bst) :  bst =
  case (T1, keep) of
      (Leaf, true) => T2

    | (Leaf, false) => ____________________
    | (Node(L1,R1,k1),  _) =>
      let

        val (L2, exists, R2) = split (_______________________)
      in

        join (__________________________________________,

              __________________________________________,

              __________________________________________)
      end
```

(b) We can use `combine` to implement the various set operations by making one call with carefully chosen arguments. For example,

```
val union = combine (true,(fn (k,_) => SOME k)
```

You may assume that `combine` works correctly, even if you did not implement it.

i. (3 points) Implement set intersection with exactly one call to `combine`.

```
val inter = combine ________________________________________________

                    ________________________________________________
```

ii. (3 points) Implement set difference with exactly one call to `combine`.

```
val diff = combine ________________________________________________

                   ________________________________________________
```

iii. (3 points) Implement symmetric set difference with exactly one call to `combine`. Recall that the symmetric difference of sets A and B is

$$A\Delta B := \{x : x \in A \oplus x \in B\}$$

where $\oplus$ is exclusive or.

```
val symdiff = combine ________________________________________________

                      ________________________________________________
```

(c) (3 points) We could also implement `symdiff` as

```
fun symdiff (A,B) = diff(union(A,B), inter(A,B))
```

Give one reason other than code reuse that it would be preferable to use the implementation written with `combine`.

Question 4: MST and Tree Contraction (25 points)

In this question, we will analyze another parallel algorithm of the Minimum Spanning Tree problem. In particular, consider the following algorithm on an **undirected** graph $G = (V, E)$ with **unique weights**. We'll assume throughout this problem that the edges are undirected, and labeled with a unique identifier (ℓ).

```
% returns the set of edges in the minimum spanning tree of $G$
fun 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 four 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) What technique would you suggest to efficiently contract the forest in parallel. What is a tight asymptotic bound of the work and span of your contract in terms of $n = |V|$ (explain briefly)? Are these bounds worst case or expected case?

(d) (4 points) Argue that each recursive call to *`MST`* removes, in the worst case, at least 1/2 the vertices; that is, $|V'| \leq |V|/2$.

(e) (4 points) What is the maximum number of edges that could remain after one step (i.e., the size of $|E'|$ in terms of $m = |E|$ and $n = |V|$)? Explain briefly.

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

Question 5: Treaps (20 points)

(a) (10 points) Assume that priorities are generated using a random hash function $h : \textsf{keys} \to [0,1]$. For keys $1, 2, 3, 4, 5$, assume the corresponding hash values are as follows.

key	1	2	3	4	5
h(key)	0.4	0.1	0.5	0.2	0.6

What would the Treap look like if we insert the keys $1, 4, 2, 5, 3$ in this order?

(b) (10 points) In our analysis of the expected depth of a key in a Treap, we made use of the following indicator random variable

$$A_i^j = \begin{cases} 1 & j \text{ is an ancestor of } i \\ 0 & \text{otherwise} \end{cases}$$

Write an expression for S_i—the size of a subtree rooted at key i—in terms of A_i^j.

Derive a closed-form expression for $\mathbf{E}[S_i]$ (you're allowed to use $\ln n, H_n, n!$ and the like in your expression).