15–210: Parallel and Sequential Data Structures and Algorithms

Practice Midterm II (Solutions)

November 2012

- There are 9 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.

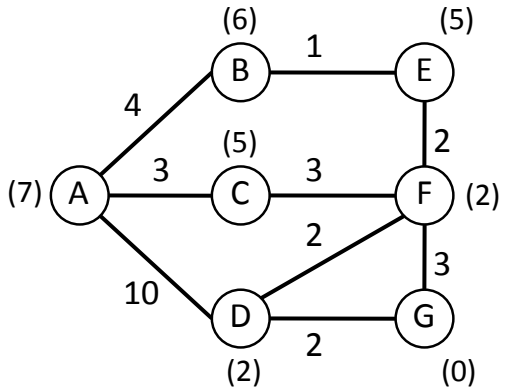
Full Name: Edsger W. Dijkstra

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, 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. Remember that Dijkstra doesn't take into account the heuristic distances.

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 goal vertex is G.

Solution: A C F G or A C F B G or A C F B E 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 torward the goal more directly skipping many more vertices than 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.

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 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) (6 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.

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) (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 destination 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?

1 **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 structure of Dijkstra's proof of correctness, so we could explore the vertices like in Dijkstra's.

(c) (8 points) **I Prefer Chalk.** There is a very unusual street in your neighborhood. This street forms a perfect circle, and there are $n \geq 3$ houses on this street. As the unusual mayor of this unusual neighborhood, you decide to hold an unusual lottery. Each house is assigned a random number $r \in_R [0, 1]$ (drawn uniformly at random). Any house that receives a larger number than **both** of its two neighbors wins a whiteboard marker and two pieces of chalk in celebration of education.

i. Compute the probability that a given house wins a prize. Justify your answer.

Solution: Oops, this is on homework 7.

ii. Let random variable X be the number of prize packages that you will be responsible for buying. What is the expected number of prizes given, $\mathbf{E}[X]$? Justify your answer.

Solution: Oops, this is on homework 7.

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 (  T2, k1  )
      in

        join (  combine (keep, f) (L1, L2)  ,

               f(k1,exists)  ,

               combine (keep, f) (R1, R2)  )
      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 (false,(fn (k, true) => SOME k
                          | (k, false) => NONE)
```

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

```
val diff = combine (false,(fn (k, true) => NONE
                          | (k, false) => SOME(k))
```

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 (true,(fn (k, true) => NONE
                          | (k, false) => SOME(k))
```

(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`.

Solution: Using combine would require passing over the trees just once instead of three times. It would therefore presumbably be cheaper by an constant amount.

Question 4: MST and Tree Contraction (25 points)

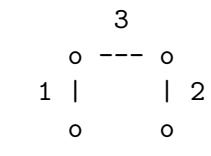
In class we covered Boruvka's basic algorithm for the Minimum Spanning Tree problem briefly but then described an optimized version that interleaved a star contract and finding minimum weight edges. In this questions you will analyze Boruvka's algorithm more carefully. Some parts are answered in the notes, but you should try the whole thing without looking at the notes.

We'll assume throughout this problem that the edges are undirected, and each is labeled with a unique identifier (ℓ). Consider the following code:

```
1  % returns the set of edges in the minimum spanning tree of G
2  fun MST(G = (V, E)) =
3    if |E| = 0 then {}
4    else let
5      val F = {min weight edge incident on v : v ∈ V}
6      val (V', P) =contract each tree in the forest (V, F) to a single vertex
7                          V' = remaining vertices
8                          P = mapping from each v ∈ V to its representative in V'
9      val E' = {(P_u, P_v, ℓ) : (u, v, ℓ) ∈ E | P_u ≠ P_v}
10   in
11     MST(G' = (V', E')) ∪ {ℓ : (u, v, ℓ) ∈ F}
12   end
```

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

Solution:



(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) 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?

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 1/2 the vertices; that is, $|V'| \leq |V|/2$.

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., the size of $|E'|$ in terms of $m = |E|$ and $n = |V|$)? 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 $m = |E|$ and $n = |V|$? 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.

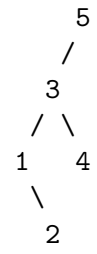
Question 5: Treaps (20 points)

(a) (10 points) Assume that priorities are generated using a random hash function $h : \mathsf{keys} \to [0, 1]$. 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?

Solution: Assuming a max heap (maximum priority at root):



(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.

Solution: $S_i = \sum_{j=1}^{n} A_j^i$

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).

Solution:

$$\begin{aligned} \mathbf{E}[S_i] &= \sum_{j=1}^{n} 1/(|j-i|+1) \\ &= H_i + H_{n-i+1} - 1 \\ &= O(\log n) \end{aligned}$$