Chapter 1

Quiz II (Version A)

- You have 90 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.
- You must open the exam in one and only one browser tab. Not doing so could lead to loss of work.

1.1 Recurrences

Recall that $f(n)$ is $\Theta(g(n))$ if $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.

Note. You do not have to show your work, but it might help you get partial credit.

Problem (16. Points). Give a closed-form solution in terms of Θ for the following recurrences. Also, state whether the recurrence is dominated at the root, the leaves, or equally at all levels of the recurrence tree.

Question. Give closed form for $f(n) = 5f(n/5) + \Theta(n)$
Solution. $\Theta(n \lg n)$.

Select one: –Balanced
–Leaves Dominated
✓ Root dominated

Question. Give closed form for $f(n) = 3f(n/2) + \Theta(n^2)$
Solution. $\Theta(n^2)$

Select one: According to brick method, the recurrence is
–Balanced
–Leaves Dominated
✓ Root dominated

Question. $f(n) = f(n/2) + \Theta(\lg n)$
Solution. $\Theta(\lg^2 n)$

Select one: According to brick method, the recurrence is
✓ Balanced
–Leaves Dominated
–Root dominated

Question. $f(n) = 5f(n/8) + \Theta(n^{2/3})$
Solution. $\Theta(n^{\lg_8 5})$ (roughly $\Theta(n^{0.77})$) leaves.

Select one: According to brick method, the recurrence is
–Balanced
✓ Leaves Dominated
–Root dominated

1.2 Short Answers

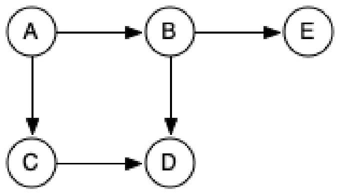
Problem (4. Points) Classes.
Lets say you are given a table that maps every student to the set of classes they take.

Question.
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.
Fill in the blanks:

```
fun allClasses(T : classSet studentTable) : classSet =
    ____ Table.reduce Set.union Set.empty T ____
```

Problem (5. Points) Shortest Weighted.
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.

Question. 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.
Problem (5. Points) Visit and Finish.
Recall the implementation of DFS shown in class using the `visit` and `finish` functions. Select the correct answer for each of the following statements, assuming DFS starts at A:



True or False? `visit D` could be called before `visit E`
Solution: True

True or False? `visit E` could be called before `visit D`: **Solution: True**

True or False? `visit D` could be called before `visit C` **Solution: True**

True or False? `finish A` could be called before `finish B` **Solution: False**

True or False? `finish D` could be called before `visit B` **Solution: True**
Problem (4. Points) Parallel Cycle Detection. **Question.** Describe in words how you would as part of star contraction efficiently detect that an undirected graph has a cycle. **No more than two sentences.**

Solution. If during star contraction we find any self edges not involved in contraction itself then there is a cycle. However, we need to be careful not to remove duplicate edges, and only to contract along a single duplicate edge.

Problem (6. Points) Star Contraction. Select **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.

Select any: ✓ a graph in which all vertices have degree at most 2

–a graph in which all vertices have degree at most 3

–a graph in which all vertices have degree $\sqrt{|V|}$

✓ a graph containing a single cycle (i.e. a forest with one additional edge)

✓ the complete graph (i.e. an edge between every pair of vertices)

–any graph (still circle others if relevant)

1.3 (Sets and Tables) Bingled

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:

$$\begin{array}{ll} \langle & (1, \text{“}\texttt{the big dog}\text{”}), \\ & (2, \text{“}\texttt{a big dog ate a hat}\text{”}), \\ & (3, \text{“}\texttt{i read a big book}\text{”}) \\ \rangle & \end{array}$$

the document index would be represented as

$$\begin{array}{lll} \texttt{idx} = & \{ & \text{“}\texttt{a}\text{”} \mapsto \{(2,0),(2,4),(3,2)\} \\ & & \text{“}\texttt{big}\text{”} \mapsto \{(1,1),(2,1),(3,3)\}, \\ & & \text{“}\texttt{dog}\text{”} \mapsto \{(1,2),(2,2)\}, \\ & & \ldots \\ & \} & \end{array}$$

In particular you want to support the following interface

```
signature INDEX = sig
  type word = string
  type docId = int
  type index = docIdIntSet wordTable

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

  val makeIndex : (docId * string) seq -> index
  val find : index -> word -> docList
  val adj : docList * docList -> docList
  val toSeq : docList -> docId seq
end
```

where, given an index `idx`,

```
toSeq (adj (find idx "210", find idx "rocks"))
```

would return a sequence of identifiers of documents where "210" appears immediately before "rocks", and

```
toSeq (adj (find idx "Umut", adj (find idx "loves", find idx "climbing")))
```

would return a sequence of identifiers of documents where the phrase "Umut loves climbing" appears.

Problem (8. Points).

Question. Show the pseudocode for generating 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. You can use the function

```
toWords~:~string -> string seq
```

that breaks a text string into a sequence of words.

Fill in the blanks:

```
type index = docIdIntSet wordTable

fun makeIndex (docs : (docId  * string) seq) : index =
  let
       ____ fun tagWords (id,doc) =
____
       ____ let val words = toWords doc
____
       ____ in
____
       ____ Seq.tabulate (fn i = (nth i words, (id, i)) (length words)
____
       ____ end
____

       ____ val allPairs = Seq.flatten (Seq.map tagWords docs)
____

       ____ val wordTable = Table.collect allPairs
____
       ____
____
       ____
____
       ____
____
       ____
____
in
     ____ Table.map Set.fromSeq wordTable
____
end
```

Problem (8. Points).

Show code for a function `adj(docList1, docList2)` that given two docLists returns a docList in which those two words are adjacent. For example for the index generated from the documents above,

```
toSeq(adj(find idx "a", find idx "big"))
```

would return $\langle 2, 3 \rangle$. For full credit `adj` must be an associative operator.

Question. 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.

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

Fill in the blanks:

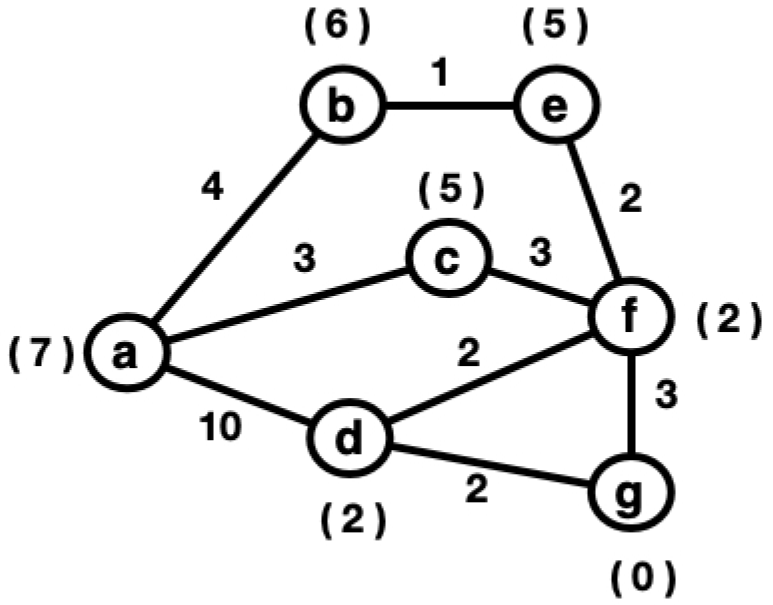
```
type docList = (docIdIntSet) * int

fun adj (____        (d1,l1)        ____ , ____       (d2,l2)        ____ )
: docList =
  let
      ____ val d2' = setmap (fn (d,i) = (d, i-l1)) d2
____
  in
      ____ (Set.intersection (d1,d2'), l1+l2)
____
  end
```

1.4 (Shortest Paths) Dijkstra and A*

Problem (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.



Question. Show the order in which vertices are visited by Dijkstra when the source vertex is A. **Solution.** A C B E F D G

Question. 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
Problem (4. Points).

Question. 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.
Problem (5. Points). **Question.** 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
```

1.5 (Graphs) Strongly Connected Components

In this question, you will write 2 functions on directed graphs. We assume that key comparisons take $O(1)$ work and that graphs are represented as:

```
type graph = vertexSet vertexTable
```

Problem (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.

Fill in the blanks:

```
function transpose (G : graph) : graph =
let
  val S = vertexTable.toSeq(G)

  function flip(u,nbrs) = Seq.map ____      (fn v.(v,u))      ____ (vertexSet.toSeq nb

  val ET = Seq.flatten(Seq.map flip S
  val T = vertexTable.____  collect             ____  ET
in
  vertexTable.map ____ vertexSet.fromSeq T              ____
end
```

Problem (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.

Question. 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.

Fill in the blanks:

```
fun scc ($G$ : graph, v : vertex) : vertexSet =
    ____  vertexSet.intersection(reachable(G,v),}              ____
    ____ reachable(transpose(G,v)))}                           ____
```

1.6 (MST) MST and Tree Contraction

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 *)

MST (G = (V,E)) =
  if |E| = 0 then {}
  else let
    F = {min weight edge incident on v : v in V}
    (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'
    E' = { (P_u, P_v, l) : (u, v,  l) in E  |  P_u <> P_v }
  in
    MST (G' = (V',E')) union {l : (u,v,l) in F}
  end
```

Problem. **Question (4. Points).** Show an example graph with 4 vertices in which `F` will not include all the edges of the MST. Specify the graph as a set of vertices and edges, e.g., $V = \{0, 1, 2\}$ and $E = \{\{0, 1\}, \{0, 2\}\}$.

Solution.

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

Question (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.

Problem (4. Points). **Question.** 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.
Problem (4. Points). **Question.** 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 ($n = |V|$).
Problem (4. Points). **Question.** 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.
Problem (5. Points). **Question.** 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.