Review Topics #3

Data Structures

You must justify all your answers to recieve full credit

Final Exam has 5 questions, but no programming. You can use computers, online resources, books and notes. Communication with others is forbidden during the exam. Questions may combine topics from various parts of the *Data Structures* course; roughly 40% of the needed theory is in Review Topics #1 and #2, the remaining 60% are in Review Topics 3.

1. **Use graph representations and graph properties**
   - 1.A. Given some parameters of a graph, estimate other parameters.
   - 1.B. Implement a function of the Graph ADT using a matrix or adjacency list.
   - 1.C. Given a directed or undirected graph, traverse its vertices in BFS order.
   - 1.D. Given a directed or undirected graph, traverse its vertices in DFS order.
   - 1.E. Given a graph, find paths or cycles using BFS or DFS traversals.
2. **Run and Analyze Graph Algorithms**
   - 2.A. Given a directed graph find its topological sorting or find a cycle.
   - 2.B. Given a directed acyclic graph, compute its transitive closure.
   - 2.C. Given a directed graph find its strongly connected components.
   - 2.D. Shortest paths with Dijkstra's or Bellman-Ford algorithm in a graph.
   - 2.E. Given an undirected graph, find its MST using Prim's or Kruskal's algorithm.
   - 2.F. Run Ford-Fulkerson or Edmonds-Karp algorithm to find the maximum flow.
3. **Use and Analyze Maps, Sets and Hashing**
   - 3.A. Find time complexity for a Map ADT function (hashing, BST, list implementation).
   - 3.B. Given a hash function, identify collisions or estimate probabilities of collisions.
   - 3.C. Build and analyze hash tables, if collisions are resolved by chaining.
   - 3.D. Build hash tables, if collisions are resolved by linear probing.
   - 3.E. Run and analyze rolling hash – polynomial and cyclic polynomial variants.
4. **Use and Analyze String Search Algorithms**
   - 4.A. Run and analyze naive string search algorithm.
   - 4.B. Run and analyze Rabin-Karp algorithm with a given rolling hash function.
   - 4.C. Run and analyze KMP algorithm, the prefix function used by this algorithm.
   - 4.D. Run NFA on a string, describe languages accepted.
   - 4.E. Convert NFAs to regular expressions and vice versa.
   - 4.F. Write regular expressions to match, search or substitute certain string patterns.
   - 4.G. Draw a "trie" (a tree showing common prefixes) from the given strings.
   - 4.H. Convert "trie" to an LCP array; convert an array to a Cartesian tree.
   - 4.I. Build a suffix tree, show how it can answer a given search query efficiently.

1 Use graph representations and graph properties

1.A. **Given some parameters of a graph, estimate other parameters.** For a directed or undirected graph some parameters are known – the numbers of vertices, edges, connected or strongly connected components. Find estimates for some other parameters.

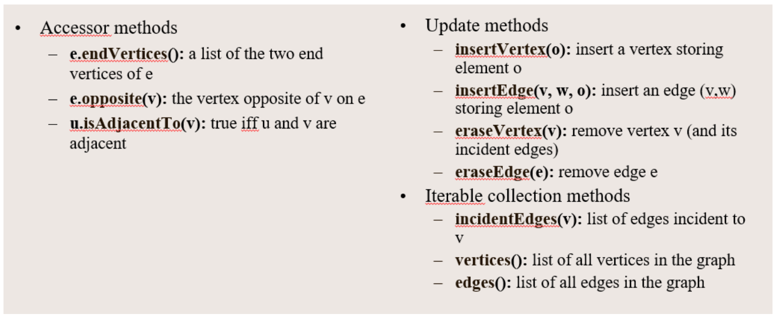
Question 1. Let an undirected graph G be such that it contains m edges and each vertex has degree equal 3 or 5. What are the possible values for $n = |V|$ – the number of nodes in this graph?

Question 2. An undirected graph with $n = 12$ vertices consists of three connected components. What is the largest number of edges in this graph?

Question 3. A directed graph has n nodes and each vertex has in-degree (the number of inbound edges) equal to 3 or to 5, and an out-degree (the number of outbound edges) equal to 5 or to 7. What is the possible total number of edges in this graph?

1.B. **Implement a function of the Graph ADT using a matrix or adjacency list.**

Question 4. Assume that a directed graph is stored as an adjacency matrix. For each of the given ADT functions find the worst-case complexity to implement it in terms of $n = |V|$, $m = |E|$, the degrees of the vertices $deg(v)$ and other graph parameters.



- Accessor methods
  - **e.endVertices():** a list of the two end vertices of e
  - **e.opposite(v):** the vertex opposite of v on e
  - **u.isAdjacentTo(v):** true iff u and v are adjacent
- Update methods
  - **insertVertex(o):** insert a vertex storing element o
  - **insertEdge(v, w, o):** insert an edge (v,w) storing element o
  - **eraseVertex(v):** remove vertex v (and its incident edges)
  - **eraseEdge(e):** remove edge e
- Iterable collection methods
  - **incidentEdges(v):** list of edges incident to v
  - **vertices():** list of all vertices in the graph
  - **edges():** list of all edges in the graph

Question 5. The *square* of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E')$ such that $u, v \in E'$ if and only if G contains a path with at most two edges between u and v. (Namely, G^2 contains all the edges of the original graph G plus also new edges (u, v) whenever there exists a path $u \to w \to v$ for some $w \in G$.)

(A) Describe a "brute force" procedure to find the adjacency matrix of G^2 given the adjacency matrix of G. Find its worst-case time complexity as a function of $n = |V|$.

(B) Describe a "brute force" procedure to find the adjacency list representation of G^2 given the adjacency list representation of G. Find its worst-case time complexity.

1.C. **Given a directed or undirected graph, traverse its vertices in BFS order.**

BFS traversal is a procedure that adds more information to the vertices and edges of the graph. Namely, each vertex changes colors from white (undiscovered) to gray (discovered and inserted into the queue of BFS) to black (fully processed with all its neighbors added to the queue). Each vertex also gets attributes "p" – its pointer to the parent and "d" – its distance to the BFS root.

```
$Q = \emptyset$
ENQUEUE($Q, s$)
while $Q \neq \emptyset$
    $u$ = DEQUEUE($Q$)
    for each $v \in G.Adj[u]$
        if $v.color$ == WHITE
            $v.color$ = GRAY
            $v.d = u.d + 1$
            $v.\pi = u$
            ENQUEUE($Q, v$)
    $u.color$ = BLACK
```

Question 6. Assume that the graph $G(V, E)$ is given by an adjacency matrix of size $n \times n$, where $n = |V|$ is the number of vertices. The graph also has $m = |E|$ edges. Assume that we run the BFS traversal to create the BFS tree in the graph G. Express its time complexity in terms of n and m.

Question 7. The *square* of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E')$ such that $u, v \in E'$ if and only if G contains a path with at most two edges between u and v.

Assume that somebody runs the BFS traversal two times – on the original graph G and also on its square G^2. During that process two BFS trees are obtained – name them T and T' respectively. There are also two queues used during the BFS search – name them Q and Q' respectively. Consider the following parameters:

(A) $maxlen(Q)$ and $maxlen(Q')$ are the maximum lengths of queues Q and Q' respectively during the execution of BFS.
(B) $maxchildren(T)$ and $maxchildren(T')$ denote the maximum number of children of any node in the trees T and T' respectively.
(C) $height(T)$ and $height(T')$ denote the height of the trees T and T' respectively.

Write the estimates for $maxlen(Q')$, $maxchildren(T')$, $height(T')$, if we know the parameters $maxlen(Q)$, $maxchildren(T)$, and $height(T)$.

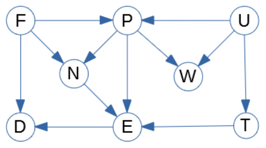
1.D. **Given a directed or undirected graph, traverse its vertices in DFS order.** Traversal algorithms include coloring vertices (as white – unvisited, gray – being processed and black – exited), classifying edges (discovery edges, fordward edges, back edges and cross edges) creating time labels for discovery and finish times.

1.E. **Given a graph, find paths or cycles using BFS or DFS traversals.** Use BFS to find the shortest path from some vertex v to other vertices in the graph.

2 Run and Analyze Graph Algorithms

2.A. **Given a directed graph find its topological sorting or find a cycle.** Run the algorithm for topological sorting on a graph or estimate its worst-case time complexity in terms of vertice and edge counts (n and m respectively).

Question 1. Assume that the topological sorting of the vertices in the graph is `U, T, F, P, W, N, E, D`. Assume that this order is obtained by reversing the exit times of some DFS traversal. Draw the 8 vertices and show only "discovery edges" belonging to those DFS traversal trees that created this topological sorting order. You will get one or more DFS trees (specify the order in which these trees are visited).



Node. DFS for topological sorting always works, since a node in DFS traversal is never exited before a node it points to is exited.

2.B. **Given a directed graph find its strongly connected components.**

2.C. **Shortest paths with Dijkstra's or Bellman-Ford algorithm in a graph.** Shortest paths are found either by a faster Dijkstra's algorithm (but only positive edges), or Bellman-Ford algorithm (both positive and negative edges).

2.D. **Given a directed acyclic graph, compute its transitive closure.** Transitive closure (also mentioned in Discrete structures) can be done using Floyd-Warshall's algorithm. It is also used to find all pairs shortest paths.

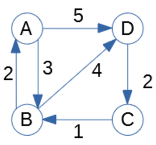
Theory. Let $G(V, E)$ be a directed graph (possibly with weights on edges, but you can also define all edges with the same weight 1). Define the following $n \times n$ matrix of weights $W = (w_{ij})$:

$$w_{ij} = \begin{cases} 0, & \text{if } i = j \\ \text{weight of edge } (v_i, v_j), & \text{if edge exists} \\ \infty & \text{if edge does not exist} \end{cases}$$

We can assume that this graph can have negative-weight edges (but it must not have negative weight cycles). The algorithm to find the shortest paths between any two vertices in graph G is the $O(n^3)$ worst-case time complexity algorithm given by this pseudocode:

```
FLOYD-WARSHALL(W)
n = W.rows
D^(0) = W
for k = 1 to n
    let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
    for i = 1 to n
        for j = 1 to n
            $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
return $D^{(n)}$
```

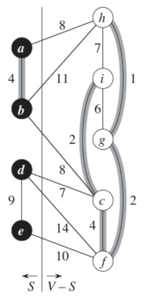
Question 2. Given the directed graph shown in the picture, find the distances between any two vertices in this graph – represent the vertices as a 4×4 matrix $D^{(0)} = (d_{ij}^{(0)}) = (w_{ij})$ where the rows and the columns correspond to the vertices A, B, C, D (in this order), but d_{ij} is initialized to the weights of edges in the graph.



Show the matrices $D^{(1)}$, $D^{(2)}$, $D^{(3)}$, $D^{(4)}$ that have elements $d_{ij}^{(k)}$ showing what is the shortest path from v_i to v_j using either the shortest path $v_i \rightsquigarrow v_j$ found in an earlier iteration or the path $v_i \rightsquigarrow v_k \rightsquigarrow v_j$ going through the vertex v_k (whichever is shorter).

2.E. **Given an undirected graph, find its MST using Prim's or Kruskal's algorithm.** Run Prim's and Kruskal's algorithm, analyze its worst-case time complexity for different graph representations (adjacency matrices and adjacency lists).

Question 3. Consider the following graph $G(V, E)$ with weighted edges.



The vertices are split into two disjoint sets S and $V - S$ (separated by a vertical line). Such splits of vertices are named *cuts* (flows are also used for maximum flow algorithms). Some edges in this graph are already added to a MST (they are highlighted and shown thicker and darker).

(A) Which edge that goes across the cut (connects a vertex from S to a vertex from $V - S$) is such that it is "safe" to add to the existing MST edges so that we can build a valid Minimum Spanning Tree with keeping this edge.

(B) In how many ways can you add all the missing edges to the existing/highlighted edges so that you get a complete MST?

Question 4. Given a graph $G(V, E)$ with n vertices and m edges, the complexity of the Prim's algorithm is $O(m \log n)$ (you need to loop over all m edges; and one operation of manipulating the priority queue costs $O(\log n)$).

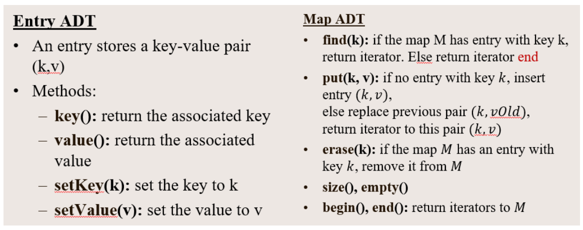
Describe a variant of Prim's algorithm that runs on the matrix representation of the graph in $O(n^2)$ time.

2.F. **Run Ford-Fulkerson or Edmonds-Karp algorithm to find the maximum flow.** Similar to Assignment 10.

3 Use and Analyze Maps, Sets and Hashing

3.A. **Find time complexity for a Map ADT function (hashing, BST, list implementation).** We want to implement the Map ADT (typically, a single function therein), and we use some underlying data structure – it may be less efficient than the typical choice (hashing). In this task you can write the pseudocode to implement this Map ADT function and estimate its time complexity.

Question 1.
The Map ADT is shown below:



Entry ADT

- An entry stores a key-value pair (k,v)
- Methods:
  - **key():** return the associated key
  - **value():** return the associated value
  - **setKey(k):** set the key to k
  - **setValue(v):** set the value to v

Map ADT

- **find(k):** if the map M has entry with key k, return iterator. Else return iterator end
- **put(k, v):** if no entry with key k, insert entry (k, v),
  else replace previous pair $(k, vOld)$,
  return iterator to this pair (k, v)
- **erase(k):** if the map M has an entry with key k, remove it from M
- **size(), empty()**
- **begin(), end():** return iterators to M

Describe, how to implement `find(...)`, `put(...)` and `erase(...)` using the underlying data structure.

(A) Implement map ADT using just some instances of "stack" data structure (instructors know how to implement `Map[Keys,Values]` with two instances of `Stack[Keys]` and two instances of `Stack[Values]`).
(B) Implement map ADT using a Binary Search Tree (BST).
(C) Implement map ADT using a hashtable.

For each of the three methods and three functions find the worst-case time complexity as $O(f(n))$, where n – the number of entries in Map right before the function is called.

3.B. **Given a hash function, identify collisions or estimate probabilities of collisions.** The task provides a hash function definition in mathematical notation (or a prehash function and a module); we want to compute specific values or estimate the probabilities of collisions. You may need to use computer simulation to find this.

3.C. **Analyze hash table data structures, if collisions are resolved by chaining.** Given a hashing method,

3.D. **Build hash tables, if collisions are resolved by linear probing.** In this problem hashtable uses open addressing (all the keys and their values are entered in a linear table). Hash collisions are resolved by probing (and picking the first empty slot). For this data structure deletion has to be done carefully (inserting `MISSING` entries whenever new gaps are created).

3.E. **Run and analyze rolling hash – polynomial and cyclic polynomial variants.**

Question 2. Consider the following rolling hash function:

$$H = s^{k-1}(h(c_1)) \oplus s^{k-2}(h(c_2)) \oplus \ldots \oplus s(h(c_{k-1})) \oplus h(c_k).$$

It is defined for a sequence (sliding window) of k characters $c_1, \ldots, c_k$, where $s^i(byte)$ denotes left rotating shift of 8 bits. For example $s^3(\texttt{10001010}) = \texttt{01010100}$ means shifting all bits 3 positions to the left (and append the bits that are "shift out" to the very end).

(A) Find the value of $H(\text{"}ABC\text{"})$, where the hashvalues $h(\ldots)$ of letters "A", "B", "C", etc. are their ASCII codes (see `https://www.asciitable.com/`).
(B) Find the value of $H(\text{"}ABCD\text{"})$ as you append one new letter "D". Write the expression to compute it using $H(\text{"}ABC\text{"})$ (without scanning the letters "ABC" once again).
(C) Find the value of $H(\text{"}BCD\text{"})$ as you skip the first letter "A".

4 Use and Analyze String Search Algorithms

4.A. **Run and analyze naive string search algorithm.** Analyze the speed of naive string search assuming something about the distribution of input data.

Question 1. Count the number of comparisons in the naive string search algorithm as it finds pattern $P = \texttt{0001}$ in the text $T = \texttt{000010001010001}$. Draw the alignments of pattern P under the text and circle all those characters that need to be compared for each alignment. Then count how many of characters were circled.

Question 2. Suppose that pattern P and text T are randomly chosen strings of length m and n, respectively. Both are written in an alphabet Σ of size $|\Sigma| = 10$. Since P and T are both random, the probability that any two characters will match is $p = 1/|\Sigma| = 1/10$. Estimate the expected number of character comparisons in the naive search algorithm using parameters m and n.

4.B. **Run and analyze Rabin-Karp algorithm with a given rolling hash function.**

Question 3. Assume that your alphabet is all 10 digits; the rolling hash function is division modulo $q = 11$. Write the formulas for $RH.skip(c)$ and $RH.append(c)$ functions.

Question 4. Assume that your alphabet is all 10 digits; the rolling hash function is division modulo $q = 11$. You want to find pattern P = ”`26`” in the text T = ”`3141592653589793`”. How many misleading (*spurious*) cache hits happen when running Rabin-Karp algorithm?

4.C. **Run and analyze KMP algorithm, the prefix function used by this algorithm.**

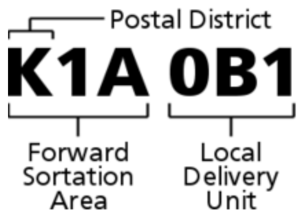
Question 5. In the string T = `947892879497` find the pattern P = `9497`.
(A) Find the prefix function $\pi(i)$ used by the Knuth-Morris-Pratt algorithm.
(B) Draw all the alignments of pattern P under the text T that are checked by the Knuth-Morris-Pratt algorithm and circle the characters that are being compared. Count the total number of compared characters.

4.D. **Run NFA on a string, describe languages accepted.** Similar to the Assignment 14.

4.E. **Convert NFAs to regular expressions and vice versa.** Similar to the Assignment 14.

4.F. **Write regular expressions to match, search or substitute certain string patterns.**

Question 6. Postal codes in Canada (see `https://bit.ly/3dlyVU4`) follow the convention that they consist of 6 symbols – alternating sequence of uppercase Latin letters and digits. Use the function `re.sub(...)` as in Python to swap two parts of a postal code. For example, the code `K1A0B1` should become `0B1K1A`. Your regular expression should work on a single string (one line of text) and swap only those postal codes which are enclosed by word boundaries (for example, words `xxxK1A0B1` or `K1A0B1yyy` should not be recognized as postal codes and no swapping should occur.



Question 7. Write a regular expression to recognize all those strings that contain only digits and represent integer numbers divisible by 8. Use the Python function `re.fullmatch(...)`.

Question 8. Write a regular expression to find all 8-digit sequences and prepend the Latvia area code. For example, a phone number `12345678` should become `+371-12345678`.

4.G. **Build a ”trie” from the given set of strings.** A trie shows common prefixes; it can be used to find the predecessor for the given pattern P in time linear in $|P|$.

Question 9. Consider the following strings:

`BAR$, BARE$, BEAR$, BEARD$, BEARER$, BEER$, BEE$, BIRD$`

(A) Draw a *trie* for the following collection of 8 strings. The strings are written in an alphabet containing all 26 uppercase letters, and dollar sign alphabetically precedes all of them (and the separating commas and spaces are not part of the alphabet).
(B) How many edges can be compressed in this trie?

(C) Write a word that is not among the 8 strings and would need the most string comparisons before we locate its predecessor in the trie.

Question 10. Assume that the alphabet Σ being used to store a trie with n nodes has a large number of letters $d = |\Sigma|$ (such as Unicode with up to 65536 symbols or something like that). A single node in the trie can be implemented in various ways – for each method estimate the space taken by this tree and also the worst-case time to search for a pattern P containing $|P| = m$ characters.

(A) Every node of a trie is itself implemented as a binary search tree with the letters from Σ being used as the search keys.
(B) Every node of a trie is implemented as a linked list with letters from Σ linked in nodes with additional pointers going to the child nodes.
(C) Every node of a trie is implemented as a hash table with letters from Σ being used as hash keys.

4.H. **Convert "trie" to an LCP array; convert an array to a Cartesian tree.**

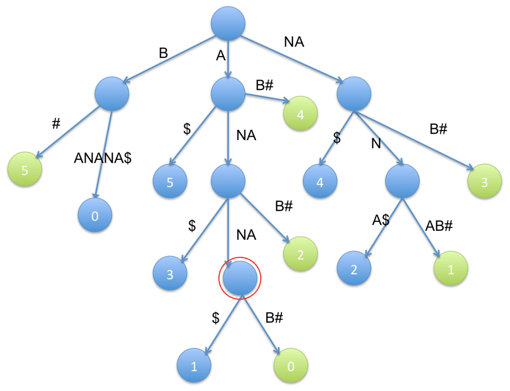
Question 11. Consider the following strings:

`ALARM$, ALAS$, ALL$, ALLEGE$, $ALLELE, ALLEY$, ALLOW$, ALLOY$, ALLY$`

(A) Build an array showing the longest common prefix (LCP) information for any two alphabetically adjacent strings from this list.
(B) Create a Cartesian tree from the array built in the previous step.
(C) Show the longest common prefix of two strings: `ALARM` and `ALLEY` in both the LCP array and the related Cartesian tree.

4.I. **Build a suffix tree/array, show how it can answer a given search query.** A suffix tree is a trie built from suffixes taken from one or more strings (and suffix array is the longest common prefix information for that trie). They can be used to answer the least common ancestor (LCA) and range-minimum queries (RMQ).

Question 12. The image below shows a suffix tree that is built from the following two strings: $P =$ `BANANA$`, $P_{rev} =$ `ANANAB#`.
P_{rev} is string P reversed. Two different end-markers are used: `$` (original string) and `#` (reversed string).



All the leaves that are suffixes of the original string are shown in blue, but all the leaves that are suffixes of the reversed string are shown in green. The tree contains a circled vertex with the following property: It is the deepest inner node that has both blue and green successors. It also turns out that `ANANA` the longest palindrome that belongs to both P and P_{rev}.

(A) If we replace $P =$ `BANANA` with an arbitrary string in this example, will the deepest inner node with both blue and green successors be a substring to both P and P_{rev}?
(B) Will the deepest node be a palindrome? Can this suffix tree find the longest palindrome contained in P.
(C) Describe the brute-force algorithm finding the longest palindrome contained within the pattern P. What is its time complexity?

Answers

Topics 1A–1E: Use graph representations and graph properties.

1. $2m = deg(v_1) + \ldots + deg(v_n)$. The largest value for $2m$ is $5n$, the smallest value is $3n$. Once we express n from these inequalities, we get $2m/5 \leq n \leq 2m/3$.
2. The largest number of edges is 45 (full graph of 10 vertices plus two more isolated vertices).
3. Observe that all in-degrees and out-degrees should equal 5 (otherwise the in-arrow count would differ from the out-arrow count). The number of nodes $n \geq 6$; if you know the number of edges m, then $n = m/5$.
4. For example, consider the adjacency matrix and two ADT functions. In order to find `e.endVertices()` for the given edge (v_i, v_j) just look at the column and row index (it is $O(1)$ time). For `insertVertex(o)` you need to allocate a larger matrix, so it is $O(n^2)$ time.
5. **(A)** Procedure is similar to matrix multiplication; it normally takes $O(n^3)$ steps (you can also speed up using Strassen's algorithm). **(B)** For each vertex you need to scan all the edges in the adjacency lists, it takes $O(n \cdot m)$.
6. It is typically more efficient to run BFS on adjacency lists. If you must use adjacency matrix, then you need to visit all edges and for each edge scan one line of the matrix. It takes $O(n \cdot m)$.
7. **(A)** Graph G^2 is more dense, the "breadth" of each level in the original tree grows by a factor $maxdegree(v)$ (maximum degree in the original graph G).

Topics 2A–2F: Run and Analyze Graph Algorithms.
TBD
Topics 3A–3E: Use and Analyze Maps, Sets and Hashing
TBD
Topics 4A–4I: Use and Analyze String Search Algorithms.
TBD