Basic Algorithms (Section 7) Practice Final
Instructor: Jiaxin Guan

December 5, 2024

1. This is a practice exam to help you prepare. You do not need to turn it in.
2. You have 110 minutes to complete the real final, this exam is intended to be comparable in length and difficulty.
3. You may use **two** double-sided, letter-size "cheatsheets". The use of phones, computers, or other reference material during the exam is not permitted.
4. You may use any algorithm or theorem we saw in class (or homework) without proof, as long as you state it correctly. For all questions asking you to give algorithms, you do not have to give detailed pseudo-code, or even any pseudo-code. It is enough to give a clear description of your algorithm. You should additionally give a brief justification of the correctness and the claimed run time of your proposed algorithm.
5. The structure of the real final will be somewhat similar to what is below. In general, you should expect questions of the following types:
   (a) Questions that can be solved if you understood what was presented in class (definitions, how the algorithms work).
   (b) Questions that can be solved by reducing to something we did in class (e.g. use some algorithm from class to solve a new problem).
   (c) Questions that can be solved by adapting an idea from class (e.g. Question 1 in HW5 asks you to modify the rod cutting DP algorithm to handle unique lengths).
6. This exam contains 7 pages (including this cover page).

Question	Points	Score
Multiple Choices	20	
Short Answers	30	
Brute Force Book Reading	25	
Reachability Sparsification	25	
Total:	100	

1 Multiple Choices (20 points)

Choose the best answer for each of the questions below. No justification is needed.

1. Imagine we want to sort a list of numbers in descending order using RadixSort. Which of the following modifications (made individually) should we make to the algorithm shown in class? (Note: "MSD" stands for "Most Significant Digit" and "LSD" stands for "Least Significant Digit".) ..... ( )
   (A) We bucket the numbers in MSD-to-LSD order instead of LSD-to-MSD.
   (B) We no longer need to pad the numbers with leading zeros.
   (C) We "unload" each bucket in last-in-first-out order, instead of first-in-first-out.
   (D) In each iteration of CountingSort, we traverse the buckets in descending order, starting with the n-th bucket and ending with the 0-th bucket.

2. Which of the following algorithms **does not** run in (worst-case) $O(n^2)$ time? . ( )
   (A) Kosaraju-Sharir algorithm (the algorithm we saw in lecture for finding strongly connected components) on a complete graph (a graph in which there is an edge between every pair of nodes) with n nodes.
   (B) Floyd-Warshall on a connected graph with n nodes where each node has degree exactly 3.
   (C) Dijkstra's algorithm (implemented with a Fibonacci Heap) on a directed acyclic graph with n nodes and positive edge weights where each node has up to 7 outgoing edges.
   (D) None of the above, i.e. all of them run in $O(n^2)$ time.

3. Let $\mathcal{A}$ be a comparison-based sorting algorithm for sorting an array of length n, and let T be the corresponding decision tree. Which of the following is **false** about T? ( )
   (A) T has at least $n!$ leaves
   (B) The depth of T is $\Omega(n \log n)$
   (C) Running $\mathcal{A}$ corresponds to an internal node of T
   (D) None of the above.

4. Let $G = (V, E)$ be a weighted directed graph. The shortest path (just the path, not the cost) from a node $s \in V$ to a node $t \in V$ will remain unchanged if: .....( )
   (A) Each edge weight $w(u, v)$ is replaced by $C \cdot w(u, v)$ for a constant $C > 0$.
   (B) Each edge weight $w(u, v)$ is replaced by $w(u, v) + C$ for a constant $C > 0$.
   (C) Each edge weight $w(u, v)$ is replaced by $w(u, v) - C$ for a constant $C > 0$.
   (D) None of the above.

5. You have compiled a collection of n important graphs on n vertices, $G_1, \ldots, G_n$. You wish to order these graphs (or pointers to these graphs) by the number of edges. Your assistant has helpfully marked each graph with how many edges it contains before you begin. Which sorting algorithms would run fastest (asymptotically)? .....( )
   (A) Merge Sort
   (B) Quick Sort (with median-of-medians as pivot rule)
   (C) Radix Sort
   (D) Counting Sort

2 Short Answers (30 points)

Give a short answer to each of the questions below. Unless specified, no justification is needed for full credit. However, in the case of a wrong answer, partial credits might be awarded if some progress is shown.

1. Suppose you have access to a magic box that can circularly shift any contiguous subarray by 1, in-place, in time $O(1)$. (For example, given the array $A = [0, 1, 2, 3, 4, 5]$, after an $O(1)$-time call `magicbox`$(A[1:5])$, we have $A = [0, 2, 3, 4, 1, 5]$). Can we sort an array A of n elements using a comparison-based algorithm with additional access to this magic box in time $O(n)$? **Briefly justify your answer.**

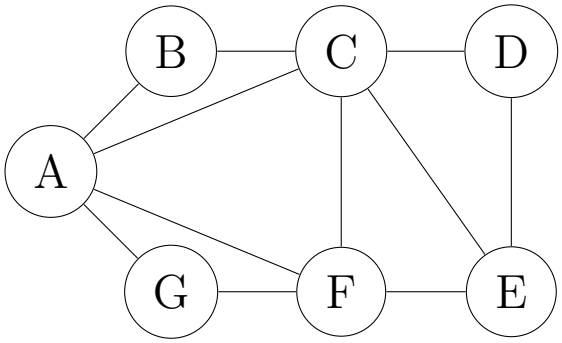
2. Consider a file that uses the following list of symbols with the following frequencies:

Letter	A	B	C	D	E
Frequency	0.08	0.07	0.25	0.15	0.45

   Find an optimal prefix code based on Huffman's algorithm. Describe both the code (i.e., mapping from symbols to bit strings) and the corresponding tree.

3. Recall that the Kosaraju-Sharir algorithm finds the SCCs of a directed graph. Imagine we start the algorithm at different vertices, is it possible that the algorithm gives us different results? **Briefly justify your answer.**

4. Recall the vertex cover problem and the independent set problem from lecture. Find the minimum vertex cover and the maximum independent set for the following graph.



5. Recall the following optimal substructure used in the Floyd-Warshall algorithm

$$DP[u, v, k] = \mathsf{min}(DP[u, k, k-1] + DP[k, v, k-1], DP[u, v, k-1]),$$

where $DP[u, v, 0]$'s are initialized to be $w(u, v)$, and the cost for the shortest path from u to v is given by $DP[u, v, n]$.

It seems that it might require a three-dimensional array DP, but actually it is possible to implement it with only a two-dimensional array. Below is such a pseudo-code implementation, but with three blanks. **Fill each blank with i, j, or k for a correct implementation.**

FLOYD-WARSHALL(w, n) :

```
1: for i = 1 to n do
2:    for j = 1 to n do
3:       DP[i, j] ← w(i, j)
4:    end for
5: end for
6: for _____= 1 to n do
7:    for _____= 1 to n do
8:       for _____= 1 to n do
9:          cost ← DP[i, k] + DP[k, j]
10:         if cost < DP[i, j] then
11:            DP[i, j] ← cost
12:         end if
13:      end for
14:   end for
15: end for
```

3 Brute Force Book Reading (25 points)

You are given a book with n chapters.

Each chapter has a specified list of other chapters that need to be understood in order to understand this chapter. To understand a chapter, you must read it after you understand every chapter on its required list.

Currently you don't understand any of the chapters, and here is what you plan to do: you are going to read the book from the beginning till the end repeatedly until you understand the whole book. Note that if you read a chapter at a moment when you don't understand some of the required chapters, you don't understand this chapter.

(a) (5 points) Give an example of a book that you can never fully understand. "Fully understand" means understanding all the chapters in the book.

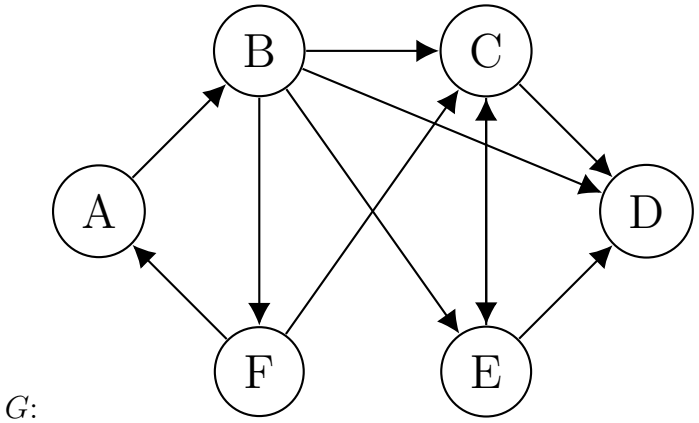
(b) (9 points) Give an algorithm to determine if a book is possible to understand fully. For full credit, your algorithm should run in time $O(n^2)$. Briefly justify the correctness and runtime of your proposed algorithm.

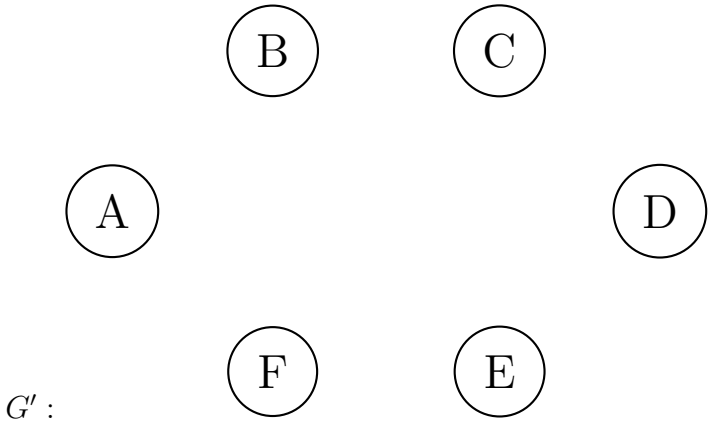
(c) (16 points) Now suppose the book is possible to understand fully. Give an algorithm to determine how many times would it take to read the book from the beginning till the end in order for you to fully understand the book. For full credit, your algorithm should run in time $O(n^2)$. Briefly justify the correctness and runtime of your proposed algorithm.

4 Reachability Sparsification (25 points)

Suppose you are given a directed graph $G = (V, E)$ and you are asked to find a *sparse representation* of this graph that preserves the reachability structure. Namely, you want to output a directed graph $G' = (V, E')$ with the smallest possible $|E'|$ such that for every pair of vertices u, v: there is a path from u to v in G if and only if there is a path from u to v in G'.

(a) (7 points) Consider the following graph G, what would its corresponding G' look like? Draw it on the provided graph below. No justification is needed.





(b) (18 points) Assume that G has a constant number of strongly connected components, give an algorithm to find such a graph G' given G. For full credit your algorithm should run in time $O(|V| + |E|)$. Briefly justify correctness and runtime of your proposed algorithm.