Practice Questions and Reading Material

Note: The chapters and exercises from the books are provided as supportive references for the material covered during the lectures.

Greedy Algorithms: Horn formulas, Set Cover

Reading material: Chapters 5.2, 5.3, 5.4 from DPV, Chapters 16.2, 16.3 from CLRS2
Practice questions:

- What is the main principle behind greedy algorithms?
- Why does the greedy algorithm work for the coin changing problem given the US coin system?
- What is the idea in Huffman encoding in order to achieve data compression? What is the prefix-free property?
- Provide the Huffman encoding algorithm and argue its running time.
- Describe the greedy algorithm for logical reasoning with Horn formulas.
- What is the property of the greedy algorithm for the set cover example? Prove it.

Elements of Dynamic Programming - Matrix Multiplication - Longest Increasing Subsequence - Longest Common Subsequence

Reading material: Chapters 6.2, 6.3, 6.5 from DPV, Chapters 15.2, 15.3, 15.4 from CLRS2
Practice questions:

- How does the dynamic programming approach decompose the chain-matrix multiplication into subproblems and how are their solutions combined in order to solve a larger problem?
- What is the dynamic programming algorithm for chain-matrix multiplication? What is its running time?
- You may be asked to compute a product of matrices using the dynamic programming approach and report the number of operations required given the dimensionality of the input matrices.
- What type of subproblems does the dynamic programming approach consider for solving the longest increasing subsequence and how does it combine them in order to solve larger ones? What is the running time of the dynamic programming solution for the longest increasing subsequence problem?
- You may be given an example sequence and asked to compute its longest increasing subsequence following a dynamic programming approach. Your solution should provide the intermediate values computed by the algorithm (e.g., length of longest subsequence, previous pointer for each digit).
- What are the subproblems defined by the dynamic programming approach for the longest common subsequence problem and how are they combined to solve larger problems? What is the running time of the dynamic programming solution for the longest common subsequence problem?

- You may be given two strings and asked to compute their longest common subsequence following a dynamic programming approach. Your solution should provide the intermediate values computed by the algorithm (e.g., the matrix of values corresponding to the cost of the matching and the parent pointers).

Elements of Dynamic Programming - Knapsack problem - Graph Representation

Reading material: Chapters 6.4, 3.1 from DPV, Chapter 22.1 from CLRS2
Practice questions:

- What are the subproblems defined by the dynamic programming approach for the knapsack problem with repetition (or without repetition) and how are they combined to solve larger problems? What is the running time of the dynamic programming solution for the knapsack problem with repetition (or without repetition)?
- You may be given the parameters of a knapsack problem and asked to compute the answer following a dynamic programming approach (for either case: with or without repetition). Your solution should provide the intermediate values computed by the algorithm (e.g., the vector or matrix of intermediate payoffs and objects selected by the intermediate solutions).
- If the input to a problem is $x_1, \ldots, x_n$ and the subproblems identified by a dynamic programming solution have the form $x_1, \ldots, x_i, i \leq n$, how many subproblems exists? Similarly, for inputs $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$, where the subproblems are $x_1, \ldots, x_i, (i \leq n)$ and $y_1, \ldots, y_j (j \leq m)$. Similarly, for input $x_1, \ldots, x_n$ and subproblems $x_i, \ldots, x_j$.
- What are the advantages and disadvantages of an adjacency matrix representation for graphs? What are the advantages and disadvantages of an adjacency list representation for graphs?

Depth-First Search

Reading material: Chapters 3.2,3.3,3.4 from DPV, Chapter 22.3, 22.4, 22.5 from CLRS2
Practice questions:

- Provide an algorithm that discovers in linear time the set of nodes in a graph reachable from a specific node. Argue about its correctness.
- Provide an algorithmic description for depth-first search. What is its running time?
- What is the pre-visit and post-visit order of nodes in depth-first search?
- How can depth-first search be used in order to detect the connected components of a graph?
- You may be provided an undirected or directed graph, a start node and asked to provide the search tree arising from a depth-first search. You may also be asked to identify tree edges, forward edges, back edges and cross edges (directed cases).
- Show that a directed graph has a cycle if and only if its depth-first search reveals a back edge.
- What is a directed acyclic graph (DAG)? What is the topological ordering of nodes in a DAG and why it is useful?
- How is it possible to identify a sink node or a source node in a DAG using depth-first search?
- What is the definition of strongly connected components? How is it possible to compute the decomposition of a directed graph into its strongly connected components? Provide an algorithm and argue about its running time.

Breadth-First Search, Properties of Shortest Paths

Reading material: Chapters 4.1, 4.2 from DPV, Chapter 22.2 from CLRS2
Practice questions:

- Provide the algorithm that performs breadth-first search on a graph. What is the inductive argument for the correctness of the approach? What is the running time of the algorithm?
- What is the definition of a single-source shortest path problem? What is the definition of a single-destination shortest path problem? What is the definition of a single-pair shortest path problem? What is the definition of all-pairs shortest path problem? Which of these problems are equivalent? Do we know faster algorithms for the single-pair shortest path problem than the single-source shortest path problem?
- What is the "optimal substructure" property of shortest paths? Prove its validity.
- What happens with shortest paths on graphs that contain negative cycles? Can a shortest path contain positive cycles?

Dijkstra's algorithm

Reading material: Chapters 4.3,4.4,4.5 from DPV, Chapter 24.3 from CLRS2
Practice questions:

- Provide Dijkstra's algorithm for computing single-source shortest paths. What is the requirement in order to be able to apply Dijkstra's algorithm?
- You may be provided an example graph and asked to return the search tree that arises from Dijkstra, as well as the state of the priority queue during its iteration of the algorithm.
- Prove the correctness of Dijkstra's algorithm on non-negative weight graphs.
- What is the best running time of Dijkstra's algorithm and for what implementation of the priority queue data structure?
- What is the running time of Dijkstra's algorithm if the priority queue is implemented as a simple array? What is the running time of Dijkstra's algorithm if the priority queue is implemented as a binary heap? When is each of these implementations preferred over the other?

Bellman-Ford and Floyd-Warshall algorithms

Reading material: Chapters 4.6,4.7,6.1 from DPV, Chapters 24.1, 24.2, 25.2 from CLRS2
Practice questions:

- Provide the Bellman-Ford algorithm for computing single-source shortest paths. What is the running time of the approach and why? Why is it correct?
- You may be provided a graph and asked to trace the dynamic programming matrix that arises from the operation of Bellman-Ford.
- How can you detect the existence of negative cycles using the Bellman-Ford algorithm?
- What is an efficient approach for computing single-source shortest paths on directed acyclic graphs that may contain negative weight edges and what is the running time of this solution?
- Why is the Floyd-Warshall algorithm preferred over calling $|V|$ times the Bellman-Ford algorithm on a graph to solve all-pair shortest path problems?

- What is the dynamic programming formulation of Floyd-Warshall, i.e., what are the subproblems defined by the algorithm and how are they combined in order to address more complex problems?

Floyd-Warshall and Johnson's algorithms

Reading material: Chapters 25.2, 25.3 from CLRS2, Chapter 6.6 from DPV2
Practice questions:

- Provide the Floyd-Warshall algorithm for solving all-pair shortest path problems. What is the running time of the approach?
- You may be provided a graph and asked to compute the dynamic programming matrix that arises from a few iterations of the Floyd-Warshall algorithm.
- What is the transitive closure of a graph and how can it be computed following a dynamic programming approach? What is the relation to the Floyd-Warshall algorithm?
- If a graph has non-negative edges, is it preferable to run the Floyd-Warshall algorithm to solve an all-pair shortest path problem or is it preferable to call $|V|$ times Dijkstra's algorithm?
- What is the main idea in Johnson's algorithm and why can it be preferable over the Floyd-Warshall algorithm when solving all-pair shortest path problems?
- Prove the following: If we define a value $h : V \rightarrow \mathbb{R}$ for every vertex of a graph $G(V, E)$ and update the weights w of a graph according to the rule: $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ for every edge $(u, v) \in E$, then a path $p =< u_0, v_1, \ldots, v_k >$ is a shortest path on G according to weights $\hat{w}$ only if it is also a shortest path according to weights w. Furthermore, negative cycles exist on the graph G according to the weights $\hat{w}$ only if they exist for the weights w as well.
- Given the above transformation of weights for a graph $G(V, E)$, how should the values h be computed for the vertices of the graph so that all weights end up having non-negative values?
- Describe Johnson's algorithmic steps. What is its running time?

Related exercises from DPV:
Chapter 3: 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.9, 3.11, 3.12, 3.15, 3.16, 3.18, 3.21, 3.22, 3.26, 3.31
Chapter 4: 4.1, 4.2, 4.3, 4.5, 4.8, 4.9, 4.10, 4.11, 4.12, 4.17, 4.21
Chapter 5: 5.13, 5.14, 5.15, 5.16, 5.17, 5.18, 5.19, 5.26, 5.28, 5.29, 5.30, 5.31, 5.33, 5.34
Chapter 6: 6.1, 6.2, 6.3, 6.4, 6.7, 6.12, 6.17, 6.18, 6.19, 6.21, 6.22

Related exercises from CLRS2:
Chapter 15: 15.2.1, 15.2-2, 15.2-4, 15.2-5, 15.3-3, 15.4-2
Chapter 16: 16.2-4, 16.2-5, 16.2-7, 16.3-1, 16-3.2, 16.3-3, 16.3-4, 16.3-5, 16.3-6
Chapter 22: 22.1.3, 22.1.5, 22.2.3, 22.2.6, 22.3.1, 22.4.3, 22.4.5, 22.5.1, 22.5.4, 22.5.6
Chapter 24: 24.3.4, 24.3.6, 24.3.8
Chapter 25: 25.2.6, 25.2.8, 25.3.4