

# CS180: FINAL STUDY GUIDE

## CONNOR JENNISON

---

**Book Sections:** Ch1, 2.1, 2.2, 2.4, Ch 3, 4.1, 4.4, 4.5, 5.1, 5.2, 5.3, 5.4, 6.1, 6.2, 6.3, 6.4, 6.6, 6.8, 7.1, 7.2, 7.5, 7.6, 7.8, 7.9, 7.11, 8.1

### Topics/Problems

- Chapter 1: Introduction
  - Stable Matching (pg. 1)
  - Interval Scheduling (pg. 12)
- Chapter 2: Algorithm Analysis
  - Computational Tractability (pg. 29)
  - Asymptotic Order (pg. 35)
  - Common Running Times (pg. 47)
- Chapter 3: Graphs
  - Basic Definitions (pg. 73)
  - Connectivity and Traversal (pg. 78)
  - Implementing Traversal (Queues/Stacks) (pg. 87)
  - Testing Bipartiteness (pg. 94)
  - Connectivity in Directed Graphs (pg. 97)
  - Directed Acyclic Graphs/Topological Ordering (pg. 99)
- Chapter 4: Greedy Algorithms
  - Interval Scheduling: Greedy Algorithm (pg. 116)
  - Shortest Paths in a Graph (pg. 137)
  - Minimum Spanning Tree Problem (pg. 142)
- Chapter 5: Divide And Conquer
  - Mergesort Algorithm (pg. 210)
  - Further Recurrence Relations (pg. 214)
  - Counting Inversions (pg. 221)
  - Finding Closest Pair of Points (pg. 225)
- Chapter 6: Dynamic Programming
  - Weighted Interval Scheduling: Recursive Procedure (pg. 252)
  - Principles of Dynamic Programming (pg. 258)
  - Segmented Least Squares: Multi-way Choices (pg. 261)
  - Subset Sums/Knapsacks: Adding a Variable (pg. 266)
  - Sequence Alignment (pg. 278)

- Shortest Paths in a Graph (pg. 290)
- Chapter 7: Network Flow
  - Max-Flow Problem/Ford-Fulkerson Algorithm (pg. 338)
  - Maximum Flows and Minimum Cuts in a Network (pg. 346)
  - Bipartite Matching Problem (pg. 367)
  - Disjoint Paths in Directed and Undirected Graphs (pg. 373)
  - Survey Design (pg. 384)
  - Airline Scheduling (pg. 387)
  - Project Selection (pg. 396)
- Chapter 8: NP and Computational Intractability
  - Polynomial-Time Reduction (pg. 459)

## Chapter 1: Introduction

### Stable Matching

While  $\exists$  an unmatched man...

- Pick a man that hasn't been matched yet (arbitrary), called  $m$
- Take the highest woman left in his priority list, called  $w$  and attempt to temporarily match them
  - If the woman is unmatched, she automatically accepts
  - If the woman is matched with another man  $m'$ , compare in her priority list
    - If  $m$  is higher than  $m'$  on her list, she leaves  $m'$  who is now unmatched again, and matches with  $m$ .
    - Else,  $m'$  is higher than  $m$  on her list, she keeps  $m'$  and  $m$  remains unmatched
  - In either case, remove  $w$  from the priority list of  $m$ , since she has been considered either way

### Interval Scheduling

- Sort the intervals by ending time from earliest to latest
- While we have intervals to look at still
  - Add the earliest ending interval to the schedule
  - Eliminate all overlapping intervals

# Chapter 2: Algorithm Analysis

## Computational Tractability

- An algorithm is efficient if it has a polynomial runtime

## Asymptotic Order

- **Order:** a function  $T(n) = O(f(n))$  if  $\exists$  constants  $n_0, c$  such that  $\forall n \geq n_0$ , we have  $T(n) \leq cf(n)$ 
  - Can think of order as an **upper bound**.
- **Omega Notation:** a function  $T(n) = \Omega(f(n))$  if  $\exists$  constants  $n_0, c$  such that  $\forall n \geq n_0$ , we have  $T(n) \geq cf(n)$ 
  - Can think of Omega notation as a **lower bound**
- If a problem/algorithm is  $O(n_1)$  and  $\Omega(n_2)$  such that  $n_1 = n_2$ , then we say the problem is  $\Theta(n)$

## Common Running Times

- $O(\log(n))$ ,
- Linear:  $O(n)$
- $O(n \log n)$
- Quadratic:  $O(n^2)$
- Non-Polynomial

# Chapter 3: Graphs

## Basic Definitions/Applications

- **Graph** - a graph is defined by a set of vertices (nodes) and edges (links). we write this formally by saying  $G = (V, E)$  with, for example,  $V = \{a, b, c, d\}$ ,  $E = \{(b, c), (a, b), (a, c), (a, d)\}$ 
  - graphs can be **directed/undirected**
  - graphs can be **weighted/unweighted**
  - graphs can be **connected/disconnected**
- If a graph has 2 or less edges with an odd number of edges attached to it, it is called an **Eulerian Graph**

## Connectivity/Traversal

- Breadth First Search
  - Investigate all points nearest our starting point before moving to the next point
  - Queue-based
  - BFS Tree: A node at level  $i$  in the tree has a shortest path to the starting point of  $i$
  - $O(e + v)$
- Depth First Search
  - Investigate all points down a path when searching
  - Stack-based
  - $O(e + v)$

### Graph Representation/Implementing Graphs

- Adjacency matrix
  - $[i, j]$  is 0 if there is no edge between  $i$  and  $j$ , 1 otherwise
  - if graph is undirected, then this matrix is symmetric
- Adjacency List
  - array of linked lists, each element of array represents node, linked list of all nodes it is connected to
  - better for directed graphs
- BFS
  - Use the queue ("first in first out")
- DFS
  - Use the stack ("last in last out")

### Testing Bipartiteness

- Algorithm
  - Run BFS and denote the  $i$ th layer of the tree as  $L_i$ 
    - At an even layer, color all nodes red
    - At an odd layer, color all nodes blue
  - After BFS, check all edges to ensure each node has a different color.

### Connectivity in Directed Graphs

- **strongly connected component** - two points  $a$  and  $b$  are strongly connected if  $\exists$  a path from  $a$  to  $b$  as well as a path from  $b$  to  $a$ 
  - The set of strongly connected components is a disjoint set

- Any strongly connected component must be a cycle

### Directed Acyclic Graphs/Topological Sort

- Terms
  - **directed acyclic graph** - a directed graph with no cycles
  - **in-degrees** - the number of directed edges coming in to that vertex,  $deg^-(v)$
  - **out-degrees** - the number of directed edges leaving a vertex. Denoted  $deg^+(v)$
  - **source** - a vertex  $v$  such that  $deg^-(v) = 0$
- **Kahn's Algorithm** (only on a DAG):  $O(e + n)$ 
  - Choose a source
  - Output the source
  - Update in-degrees of all of the vertices connected to the source
    - If any of these become a source, add them to the source list
  - Move back to step 1 and perform this recursively.
- Topological Sort Properties
  - Not necessarily unique, can be more than one solution

## Chapter 4: Greedy Algorithms

- **Greedy Paradigm** - Take a problem. Look at one or two items in the problem and make a decision very quickly without really haven't seen the entire problem. Once make decision you stick to it.

### Greedy Interval Scheduling

- Problem from before greedily chooses interval to select. Greedy algorithm stays ahead of all other algorithms (for proof)
- Multiple processors: put the next ending task on the lowest available processor.

### Shortest Paths in a Graph

- **Dijkstra's Shortest Path Algorithm** (<http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/>)
  - Runtime using adjacency matrix:  $O(n^2)$ , good for dense graphs
  - Runtime using heap:  $O(e \log n)$ , good for sparse graphs

## Minimum Spanning Tree Problem

- **spanning tree (ST)**: a graph with the following properties
  - it is a tree
  - it has the minimum number of edges ( $n - 1$  assuming graph has  $n$  vertices)
  - it spans/touches every vertex
- **minimum spanning tree (MST)**: a spanning tree such that it has minimum weight
- **Minimum Spanning Tree Theorem**
  - Take a graph, and split it into two partitions that has the following properties
    - The two partitions are disjoint
    - Each of the two partitions is non-empty
    - All points in the graph are in either of two partitions
  - Consider all edges between partition 1 and partition 2, and denote the minimum weighted edge  $e_{min}$ .
  - Then  $\exists$  a Minimum Spanning Tree that includes  $e_{min}$ .
- **Prim's MST Algorithm**

```
1  • Given a graph G, n vertices, e edges. Assume MST exists
2  • Create two partitions
3      - L: contains an arbitrary vertex
4      - R: contains all other vertices
5  • While there are still vertices in R
6      - Find the minimum edge between L and R, denoted as e[min]
7      - Move e[min] into list for MST
8      - Take vertex connected to e in partition R, and move it to
partition L
9
10 Same runtime as Dijkstra's
```

- **Kruskal's MST Algorithm**

```
1  Given a graph G, n vertices, e edges, Assume MST exists
2  • Sort all the edges by weight in non-decreasing order
3  • While there are less than (n-1) edges in the MST
4      - Consider the minimum edge that has not been considered
5      - If including this edge in the MST creates a cycle
6          * Discard and move on
7      - Else including this edge in MST doesn't create cycle
8          * Add to MST and move to next edge
9
10 O(elogv) assuming given sorted edges)
```

- Union-Find/Kruskal
  - Every vertex is in its own set at the start. We have two operations
    - **union**: take two vertices, and take the union of them
    - **find**: take two vertices, and see if they are in the same group
  - Makes checking for cycles, adding them to groups more efficient
  - Implemented as a Balanced BST
- **K-Clustering**
  - Start with each vertex in its own group
  - Perform Kruskal until we have the desired  $k$  clusters.

## Chapter 5: Divide and Conquer