Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

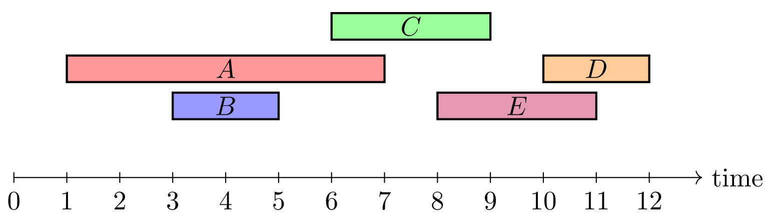
Graphs and Greedy

1 Interval Scheduling

In lecture we saw the interval scheduling problem. For the review session, we will take a look at it again and focus on understanding the exchange argument.

You are given n intervals, where interval i has start time s_i and finish time f_i. Two intervals are *compatible* if they do not overlap. Your goal is to select the largest possible set of mutually compatible intervals.

Example: Consider the following 5 intervals:



The optimal solution is $\{B, C, D\}$ with 3 intervals.

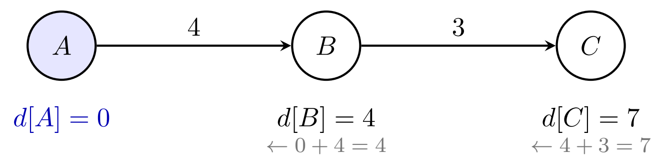
Algorithm: Sort intervals by finish time. Greedily select intervals: always pick the next interval that doesn't overlap with the last selected one.

Prove that this greedy algorithm is optimal using an exchange argument.

2 Quick Graph Algorithms Review

All four algorithms in the table below share a single core operation: **edge relaxation**. The key insight is that *global shortest paths are found by repeatedly applying a simple local update.*

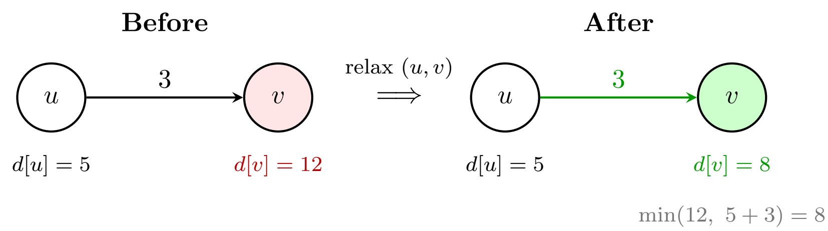
Why "local"? Consider the chain below. Node C cannot know its distance from A until B has figured out its *own* distance first. Once $d[B]$ is settled, C simply asks: "What is B's distance, plus the edge weight to reach me from B?"



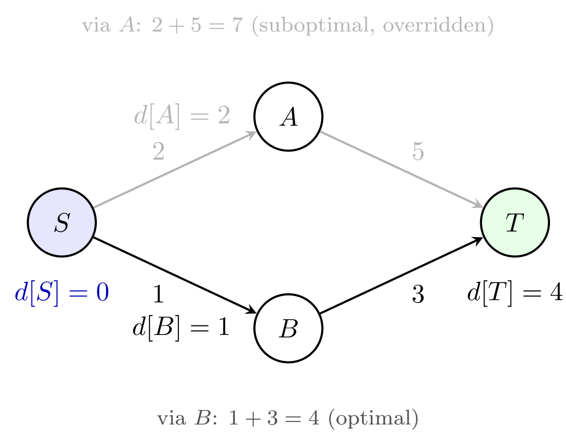
This operation updating $d[v]$ based on a neighbor u is called **relaxing** edge (u, v):

$$d[v] \leftarrow \min\big(d[v], \quad d[u] + w(u, v)\big)$$

Relaxation in action. A single relaxation step can improve a distance estimate:



Multiple paths competing. When two paths lead to the same node, relaxation automatically keeps only the shorter one:



The four algorithms below all apply this same relaxation rule they differ only in *which edges they relax and in what order.*

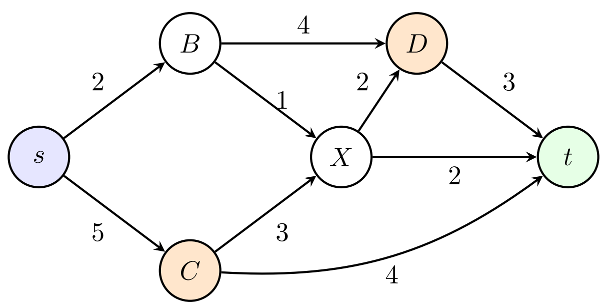
Algorithm	Key Features	Runtime
DFS	Cycle detection, topological sort, SCCs	$O(\|V\| + \|E\|)$
BFS	Layer-by-layer ("onion"); shortest paths in *unweighted* graphs	$O(\|V\| + \|E\|)$
Dijkstra	Like BFS but with non-uniform layer sizes; fails with negative edge weights	$O((\|V\| + \|E\|) \log \|V\|)$
Bellman-Ford	Handles negative edges; detects negative weight cycles	$O(\|V\| \cdot \|E\|)$

3 Eat Before Class

You are given a map of the Berkeley campus represented as a directed, weighted graph $G = (V, E)$ with positive edge weights. You start at your home node $s \in V$, and you need to get to your class at node $t \in V$. However, you are hungry and want to grab food on the way. You are given a set of nodes $F \subseteq V$ that represent your favorite food locations.

Your task is to design an efficient algorithm to find the length of the shortest path from s to t that visits at least one food location $f \in F$.

Example: Consider the following graph where $F = \{C, D\}$.



In this example, the shortest path from s to t without any restrictions is $s \to B \to X \to t$ with length $2 + 1 + 2 = 5$. However, this path does not visit any node in F. The shortest path that visits a food location is $s \to B \to X \to D \to t$ with length $2 + 1 + 2 + 3 = 8$, which visits $D \in F$. Another valid path is $s \to C \to t$ with length $5 + 4 = 9$, which visits $C \in F$. The shortest valid path is $s \to B \to X \to D \to t$ with length 8.

Describe your algorithm, prove its correctness, and analyze its runtime.

Divide & Conquer and Dynamic Programming

1 Maximum Subarray Sum

In homework we saw this problem. For review, we will try to focus on learning how recursion works.

Given an array A of n integers, the *maximum subarray sum* is the largest sum of any contiguous subarray of A (including the empty subarray). In other words, the maximum subarray sum is:

$$\max_{i \leq j} \sum_{k=i}^{j} A[k]$$

For example, the maximum subarray sum of $[-1, 3, 4, -2, 7, 8, -3, -1]$ is 20, the sum of the contiguous subarray $[3, 4, -2, 7, 8]$.

$$\begin{array}{|c|c|c|c|c|c|c|c|}\hline -1 & 3 & 4 & -2 & 7 & 8 & -3 & -1 \\ \hline \end{array}$$

$$\text{sum} = 3 + 4 + (-2) + 7 + 8 = 20$$

Design an $O(n \log n)$-time divide-and-conquer algorithm for this problem.

2 Knapsack with Repetition

You have a knapsack with capacity W and n types of items. Item i has weight w_i and value v_i. You may use each item type as many times as you like (unlimited supply). Find the maximum total value of items that fit in the knapsack.

Example: Suppose $W = 10$ and we have:

Item	**Weight** w_i	**Value** v_i
1	6	30
2	3	14
3	4	16

