DSA3 Lab session 6

This week we saw a lot of algorithms

- Shortest path:
  - Dijkstra
  - Shortest path on DAG
  - Bellman-Ford
  - Floyd-Warshall
- MST
  - Prim-Jarnik
  - Kruskal
  - Chu-Liu/Edmonds

Greedy algorithms

- At every step, make a locally optimal choice
- The expectation is that this leads to a globally optimal solution
- Greedy algorithms are often simpler and much faster
- You’re not trying to do an exhaustive search through all combinatorial possibilities with greedy algms

Greedy algorithms

- Of the ones you saw this week:
  - Dijkstra
  - Prim-Jarnik
  - Kruskal

- Not greedy:
  - Bellman-Ford
  - Chu-Liu/Edmonds
  - Floyd-Warshall

Greedy algorithms: Optimal substructure

- An optimal solution to the problem contains optimal solutions of its subproblems

- Let’s say A -> B -> C -> D is the shortest path
  - A - > B -> C is the shortest from A to C
  - B -> C -> D is the shortest path from B to D
  - B -> C is the shortest etc etc

- If shorter intervening paths (the subproblems) exist, then that means they can be used, and A -> B -> C -> D == shortest would be false

Greedy algorithms: Greedy choice property

- You can make the best-looking choice at any point and commit to it permanently, and it will be part of a globally optimal solution
- In other words, you can make a choice, recursively solve the remaining sub-problem, and that will be the globally optimal solution
- If this property does not hold, then you have to resort to more computationally intensive/exhaustive solutions

Greedy algorithms: Greedy choice property

- Dijkstra: the “permanent commit” happens when you remove a node from the priority queue (Q in your slide pseudocode) and mark it as finalized

- Prim-Jarnik: happens when you add a node to your in-progress tree

- Kruskal: happens when you add an edge to your forest

Where greedy fails, an example

- Knapsack problem
  - You're playing Skyrim and have 10kg carry capacity left
  - You want to bring the most valuable combination of loot back to sell

  - Loot A: 9kg, 90 gold – 10 gold per kg
  - Loot B: 5kg, 48 gold – 9.6 gold per kg
  - Loot C: 5kg, 48 gold – same

Where greedy fails: Dijkstra

- We know that Dijkstra fails for graphs with negative edges

- What does that mean in terms of greedy choice property?
  - You can make the best-looking choice at any point and commit to it permanently, and it will be part of a globally optimal solution

What’s the alternative?

- If we can’t pick a local optimal choice and assume that it leads to the global optimal solution, then we have to consider multiple local choices and how they affect later subproblems and the final globally optimal solution

- Dynamic programming is one solution, often the best

Dynamic Programming

- Bellman-Ford relaxes edges in passes
  - Each pass relies on previously calculated distances from previous edge-relax passes, so we're not calculating from the beginning
- Floyd-Warshall for shortest paths does something similar
  - Shortest path table is calculated at each step using results from previous steps

Let’s consider some use cases

- For shortest path we looked at:
  - Dijkstra
  - Shortest-path on directed acyclic graphs
  - Bellman-Ford
  - (briefly) Floyd-Warshall

- We’ve been hired to build a GPS app. What’s the best solution? Why?

- What if we want to avoid rush hour highway traffic?

Let’s consider some use cases

- We want to analyze Forex opportunities. Which algorithm and why?
- Forex: foreign exchange
- Arbitrage: taking advantage of a difference in prices in two or more markets, making deals to capitalize on the difference, with profit being the difference between market prices at which a good is traded

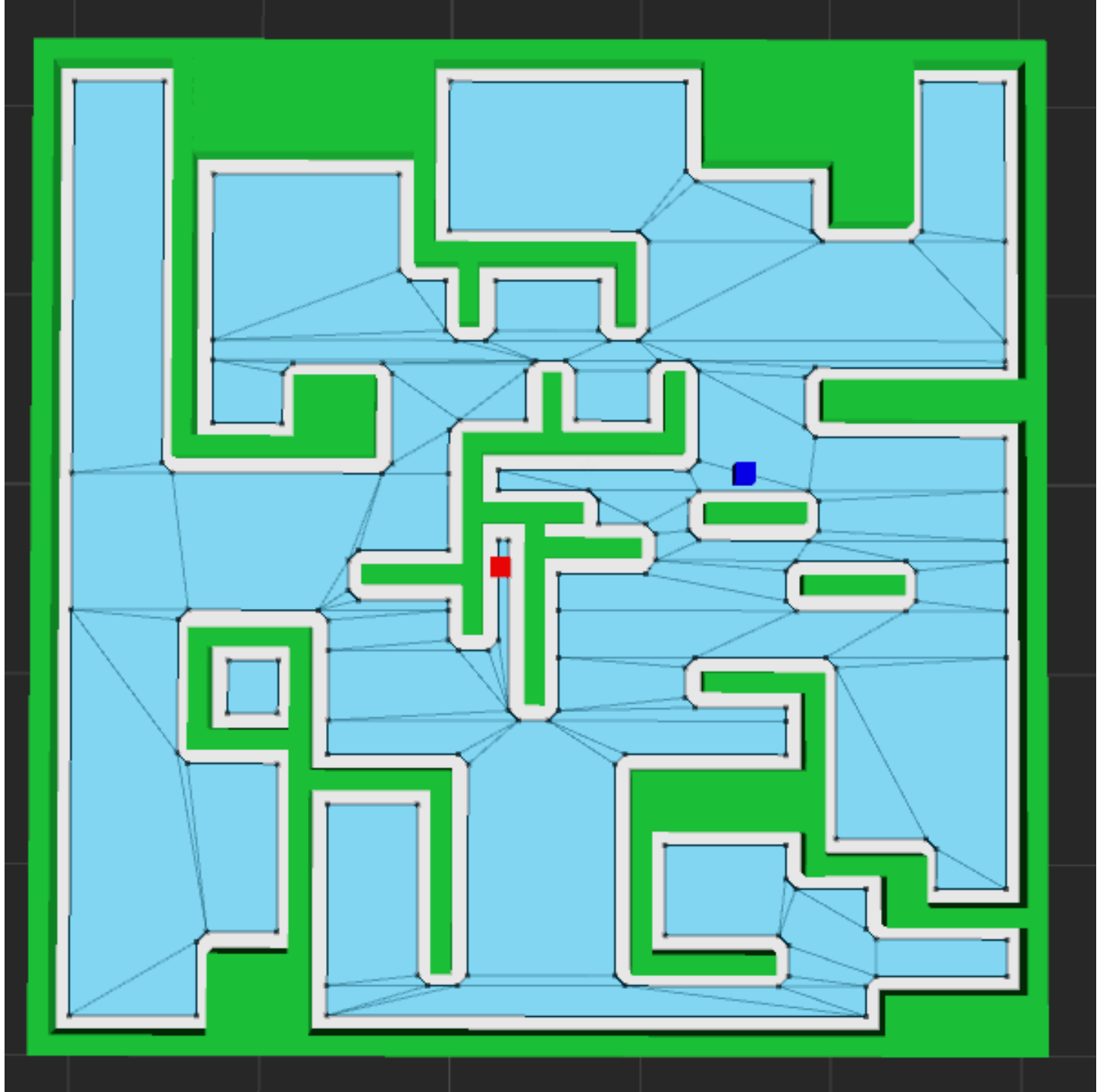
Negative cycle

- What are negative cycles?
- Why do they ruin everything?
- Let’s reconsider our Forex question from the last slide.

This will not be part of your final

- Modern pathfinding:

- A* + navmesh



Undirected graph MST

- Prim-Jarnik and Kruskal can be considered “opposite” approaches in some ways, top-down or bottom-up
- Let’s consider some ways they can be differentiated
- Which is better for a dense graph (many edges) or a sparse graph?
- What’s the best way to store the edges (of the ways you’ve learned)?

Prim-Jarnik vs Kruskal use cases

- We want to build roads between a set of cities, and the edges represent the cost.

- We’re doing image-segmentation, finding objects or similar regions in an image.
  - Essentially, we want to group “similar” pixels together
  - Edge weights represent “dissimilarity” between adjacent pixels

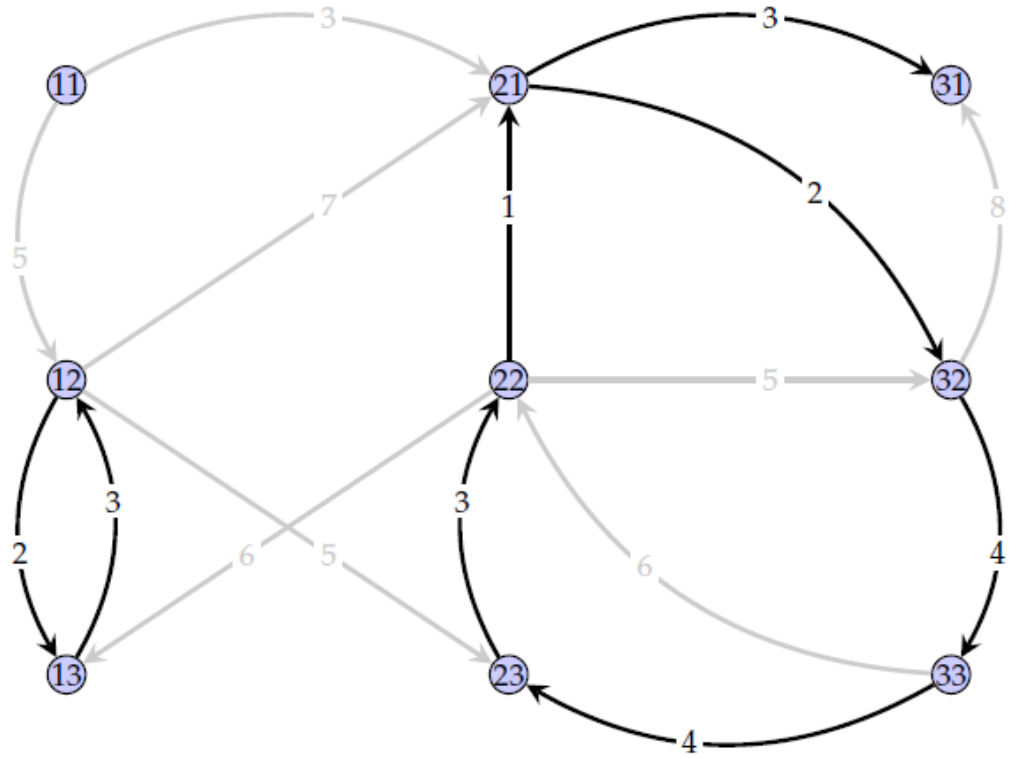
Why is Chu-Liu/Edmonds a whole different thing?

- MST with undirected graph:
  - connect all the nodes with minimum total cost

- MST with directed graph:
  - Every node must have exactly one parent
  - … with minimum total cost

Chu-Liu/Edmonds

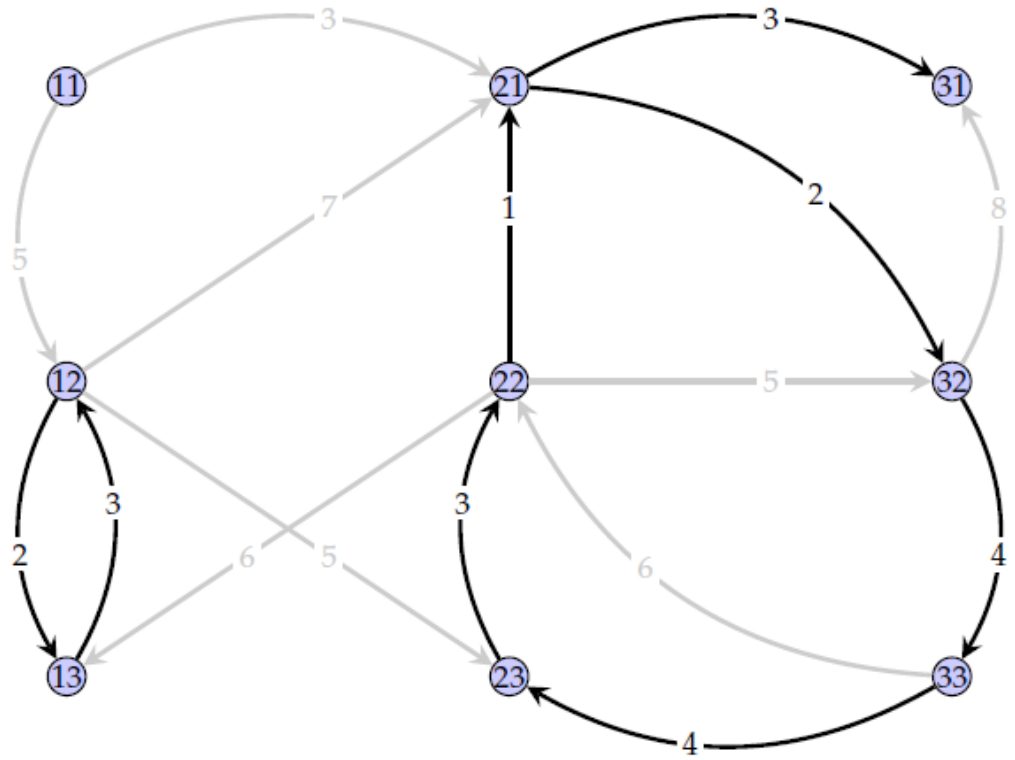
- Picking the minimum incoming edge for each node often results in cycles
- Greedy fails, and fails hard



Chu-Liu/Edmonds

- This is your lecture example, after selecting incoming edge with lowest weight and removing others

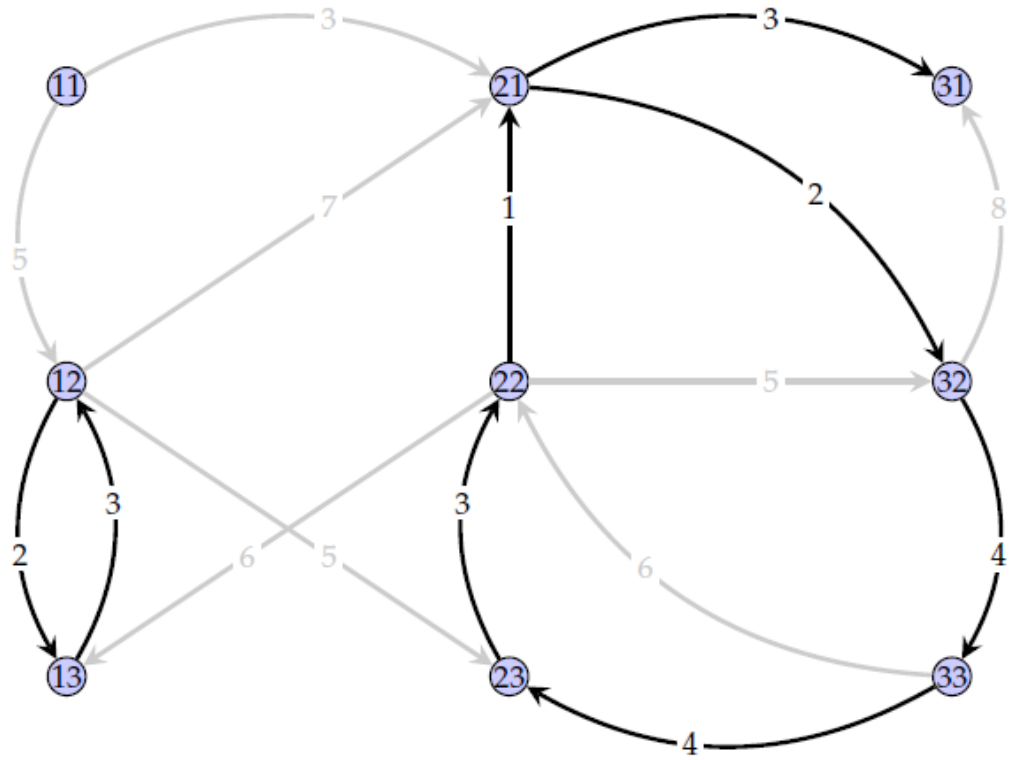
- We can consider each cycle as a “supernode”, treating it as one unit



Chu-Liu/Edmonds

- Cycles represent local optima, because we have already selected the cheapest parent for each node

- Adding one edge from outside the circle, and removing one intra-cycle edge, breaks the cycle in the cheapest way



Chu-Liu/Edmonds

- Add the lowest-cost incoming edge that can be used to break the cycle, and break the cycle
- Each node in the original “supernode” shares one external parent/ancestor
- “break cost” or weight reduction is not the same as base edge weight