Shortest Paths & Wrapping Up Dynamic Programming

Shortest Paths: Cyclic Graphs

- **Problem.** Find the cost of the shortest path from s to any node v in a directed graph G
- G can have cycles with non-negative cost (but assuming no negative cycles for now)
- Subproblem:
  - $D(v, i)$: (optimal) cost of shortest path from s to v using at most i edges

Shortest Paths: Cyclic Graphs

- **Subproblem.** $D(v, i)$: cost of shortest path from s to v using at most i edges
- Base cases?
  - $D(s, i) = 0$ for any i
  - $D(v,0) = \infty$ for any $v \neq s$
- Final answer/output for shortest path cost to node v?
  - $D(v, n-1)$, why?

Shortest Paths: Cyclic Graphs

- How many edges can be on a shortest path between two nodes?
- Can there be a cycle on a shortest path?
    - We assumed no negative cycles
    - What about a positive cycle?
    - What about a cycle of cost 0?
        - Can remove them without hurting cost
- Thus, we can assume shortest paths are simple
    - Max number of edges on a simple path?
    - $n - 1$

Shortest Paths: Recurrence

- **Subproblem.** $D(v, i)$: cost of shortest path from s to v using at most i edges
- Base cases?
  - $D(s, i) = 0$ for any i
  - $D(v, 0) = \infty$ for any $v \neq s$
- Output $D(v, n-1)$
- How do we formulate the recurrence?
  - Case 1. Shortest path to v uses $\leq i - 1$ edges
  - Case 2. Shortest path to v uses exactly i edges

Shortest Paths: Recurrence

- **Subproblem.** $D(v, i)$: cost of shortest path from s to v using at most i edges
- Base cases?
  - $D(s, i) = 0$ for any i
  - $D(v, 0) = \infty$ for any $v \neq s$
- Output $D(v, n - 1)$
- Recurrence:

$$D(v, i) = \min\{D(v, i - 1), \min_{(u,v)\in E} \{D(u, i - 1) + w_{uv}\}\}$$

- Called the Bellman-Ford-Moore algorithm

Shortest Paths: Recurrence

$$D(v, i) = \min\{D(v, i-1), \min_{(u,v)\in E} \{D(u, i-1) + w_{uv}\}\}$$

- Memoization: Two-dimensional array
- Evaluation order:
    - $i : 1 \to n - 1$
    - Vertices can be evaluated in any order
- Analysis
    - Space?
    - Running time?

Shortest Paths: Recurrence

$$D(v, i) = \min\{D(v, i-1), \min_{(u,v)\in E} \{D(u, i-1) + w_{uv}\}\}$$

- Memoization: Two-dimensional array
- Evaluation order:
  - $i : 1 \rightarrow n - 1$
  - Vertices can be evaluated in any order
- Analysis
  - Space? $O(n^2)$ entries in table
  - Running time? $O(n^3)$
    - Each entry in table takes $O(n)$ to compute
    - $O(n^2)$ entries

Bellman-Ford: Improved Analysis

- Recurrence for $D(v, i)$

$$D(v, i) = \min\{D(v, i-1), \min_{(u,v)\in E} \{D(u, i-1) + w_{uv}\}\}$$

- For a given i, v, $d[v, i]$ looks at each incoming edge of v
- Takes indegree(v) accesses to the table
- For a given i, filling $d[-, i]$ takes
    - $\sum_{v\in V}$ indegree(v) accesses
    - This is at most $O(n+m) = O(m)$ for $m \geq n-1$ (assuming G is connected)
- To fill n rows, overall running time is $O(nm)$

Bellman-Ford-Moore Correctness

- **Lemma.** (Correctness) $d[v, i]$ is the cost of a shortest path from s to v using at most i edges
- Proof. [By induction on i]
  - Base case: $i = 0$
    - $d[s, i] = 0$ for all i, $d[u,0] = \infty$ for $u \neq s$
  - Induction hypothesis:
    Assume that $d[v, i]$ is the cost of a shortest path from s to v using at most i edges
  - Inductive step: prove for $i + 1$
  - **Observe**: $d[v, i]$ never increases (only goes down)

Bellman-Ford-Moore Correctness

- Let P be the **shortest** $s \rightsquigarrow v$ path with at most $i + 1$ edges.
- Let (u, v) be last edge on P, and Q be the subpath from $s \rightsquigarrow u$
- Then Q has at most i edges, and must be shortest $s \rightsquigarrow u$ path
- By inductive hypothesis, $d[u, i] = w(Q)$
- We have $d[v, i+1] = \min\{d[v, i], d[u, i] + w[u, v]\}$
    - $d[u, i] + w[u, v] = w(Q) + w[u, v] = w(P)$
    - Thus, $d[v, i+1] \leq w(P)$
    - $d[v, i+1]$ cannot be strictly less than $w(P)$
    - $d[v, i+1]$ is based on an actual path to v in the algorithm
    - Thus, $d[v, i+1] = w(P)$ $\blacksquare$

Extracting Shortest Path

- Once we have the shortest path table, we can extract the actual shortest path in $O(m)$ time
    - Consider edges with $d[v, i] = d[u, i-1] + w_{uv}$
- Or we can do extra booking-keeping during the dynamic program to store pointers to path
    - Maintain pred$[v, i]$ that points to node leading to v on shortest path $s \rightsquigarrow v$ using at most i edges

Improving Space

- **Observation.** $d[-, i]$ only depends on $d[-, i-1]$
  - Use a one dimensional array $d[v]$, which stores the cost of the shortest $s - v$ path found so far
  - Maintain pred$[v]$ that points to node leading to v on shortest path $s \rightsquigarrow v$ found so far
  - Keep improving estimate ($i \to 1, \ldots, n-1$ now acts like a counter)
  - *(Optimization) If no estimate $d[v]$ changes during an iteration, we can just stop*

Bellman-Ford-Moore Algorithm

- Initialize:
    - For each node $v \neq s$: d[v] ← ∞, pred[v] ← null
- Initialize: d[s] ← 0
- For i = 1 to n – 1 # no of passes
    - For each node v
        - For each edge (v, w) ∈ E :
            - If d[w] > d[v] +w[v, w]:
                - d[w] ← d[v] +w[v, w]
                - pred[w] ← v

Detecting a Negative Cycle

- **Problem.** Given a weighted directed graph with edge weights w_e find if it contains a negative cycle
- Let us solve a slightly different problem first
- Given a graph G and source s, find if there is negative cycle on a $s \rightsquigarrow v$ path for any node v
- Suppose there is a negative cycle on a $s \rightsquigarrow v$ path
  - Then $\lim_{i \to \infty} D(v, i) = -\infty$
- If $D(v, n) = D(v, n-1)$ for every node v then no negative cycles exits! Why?
  - Table values converge $\implies$ shortest $s \rightsquigarrow v$ path exists

Detecting a Negative Cycle

- **Lemma.** If $D(v, n) < D(v, n-1)$ then any shortest $s \rightsquigarrow v$ path contains a negative cycle.
- Proof. [By contradiction]
  - Since $D(v, n) < D(v, n-1)$, the shortest $s \rightsquigarrow v$ path has exactly n edges
  - By pigeonhole principle, the path must contain a repeated node— let the cycle be W
  - If W has non-negative weight, removing it would give us a shortest path with less than n edges $\Rightarrow\Leftarrow$

Detecting a Negative Cycle

- Now we know how to detect negative cycles on a shortest path from s to some node v
- How do we solve the problem in general (that is, given a graph does it have any negative cycle?)
- Idea: Problem reduction!
  - Given graph G, add a source s and connect it to all vertices in G with edge weight 0
  - Let the new graph be G'
  - G has a negative cycle iff G' has a negative cycle!

More DP: Dividing Work

- Suppose we have to scan through a shelf of books, and this task can be split between k workers
- We do not want to reorder/rearrange the books, so instead we divide the shelf into k regions
- Each worker is assigned one of the regions
- What is the fairest way to divide the shelf up?

More DP: Dividing Work

- Suppose we have to scan through a shelf of books, and this task can be split between k workers
- We do not want to reorder/rearrange the books, so instead we divide the shelf into k regions
- Each worker is assigned one of the regions
- What is the fairest way to divide the shelf up?
- If the books are equal length, we can just partition into equal sizes regions
- What if books are not equal size?
    - How can we find the fairest partition of work?

The Linear Partition Problem

- **Input.** A input arrangement S of nonnegative integers $\{s_1, \ldots, s_n\}$ and an integer k
- **Problem.** Partition S into k ranges such that the maximum sum over all the ranges is minimized
- **Example.**
  - Consider the following arrangement
    $$100\ 200\ 300\ 400\ 500\ 600\ 700\ 800\ 900$$
  - Suppose $k = 3$, where should we partition to minimize the maximum sum over all ranges?

The Linear Partition Problem

- **Input.** A input arrangement S of nonnegative integers $\{s_1, \ldots, s_n\}$ and an integer k
- **Problem.** Partition S into k ranges such that the maximum sum over all the ranges is minimized
- **Example.**
  - Consider the following arrangement

    $100\ 200\ 300\ 400\ 500\ 600\ 700\ 800\ 900$

  - Suppose $k = 3$, where should we partition to minimize the maximum sum over all ranges?

    $100\ 200\ 300\ 400\ 500 \mid 600\ 700 \mid 800\ 900$

Recursive Formulation

- Notice that the kth partition starts after we place the $(k-1)$th "divider"
- Let us consider an optimal solution, where can it have the last divider?
  - Between some elements, suppose between ith and $(i+1)$the element where $1 \leq i \leq n$
  - What is the cost of placing the last divider here? Max of:
  - Cost of the last partition $\displaystyle\sum_{j=i+1}^{n} s_j$
  - Cost of the optimal way to partition the elements to the "left" — this is a smaller version of the same problem!

Dynamic Programming Recurrence

- Subproblem?
    - $M(i, j)$ be the minimum cost over all partitions of $\{s_1, \ldots, s_i\}$ into j ranges
- Base cases?
    - $M(1,\ j) = s_1$ for all $1 \leq j \leq k$
    - $M(i,\ 1) = \sum_{t=1}^{i} s_t$ for all $1 \leq i \leq n$
- Recurrence?
    - $M(i, j) = \min_{1 \leq i' \leq i} \max\{M(i', j-1), \sum_{t=i'+1}^{i} s_t\}$
- Final solution: $M(n, k)$

Running Time

- $M(i,j) = \min_{1 \le i' \le i} \max\{M(i', j-1), \sum_{t=i'+1}^{i} s_t\}$
- Final solution: $M(n,k)$
- Evaluation order? Row major order
- Running time?
    - Size of table: $O(k \cdot n)$
    - How long to compute a single cell?
        - Depends on n other cells
    - $O(n^2 \cdot k)$ time

Running Time

- Running time
  - Size of table: $O(k \cdot n)$
  - How long to compute a single cell?
    - Depends on n other cells
  - $O(n^2 \cdot k)$ time
  - Is this a pseudo polynomial running time?
  - How big can k get?
    - At most n non-empty partitions of n elements
    - $O(n^3)$ algorithm in the worst case

Dynamic Programming Practice

- Longest Common Subsequence Problem
- We are given two strings: string A of length n, and string B of length m.
- Our goal is to produce their longest common subsequence: the longest sequence of characters that appear left-to-right (but not necessarily in a contiguous block) in both strings.
- For example, consider:
    - A = abazdc
    - B = bacbad
- In this case, the LCS has length 4 and is the string abad

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)