CS 97SI: INTRODUCTION TO PROGRAMMING CONTESTS

Jaehyun Park

Today's Lecture

- Shortest Path Problem
- Floyd-Warshall Algorithm
- Dijkstra's Algorithm
- Bellman-Ford Algorithm
  - System of difference constraints

- Maybe: Problem Discussion

Shortest Path Problem

- Input: a weighted graph $G = (V, E)$
  - The edges can be directed or not
  - Sometimes, we allow negative edge weights
- Output: the path between two given nodes u and v that minimizes the total weight (or cost, length)
  - Sometimes, we want to compute all-pair shortest paths
  - Sometimes, we want to compute shortest paths from u to all other nodes

Floyd-Warshall Algorithm

- Given a directed weighted graph G
- Outputs a matrix D where d_{ij} is the shortest distance from node i to j
- Can detect a negative-weight cycle
- Runs in $\Theta(n^3)$ time
- Extremely easy to code
  - Coding time less than a few minutes

Floyd-Warshall Pseudocode

- Initialize D to the given cost matrix
- For $k = 1 \dots n$:
  - For all i and j:
    - $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$
- If $d_{ij} + d_{ji} < 0$ for some i and j, then the graph has a negative weight cycle
- Done!
  - But how does this work?

How does Floyd-Warshall work?

- Define $f(i, j, k)$ as the shortest distance from i to j, using $1 \dots k$ as intermediate nodes
  - $f(i, j, n)$ is the shortest distance from i to j
  - $f(i, j, 0) = \text{cost}(i, j)$
- The optimal path for $f(i, j, k)$ may or may not have k as an intermediate node
  - If it does, $f(i, j, k) = f(i, k, k-1) + f(k, j, k-1)$
  - Otherwise, $f(i, j, k) = f(i, j, k-1)$
- Therefore, $f(i, j, k)$ is the minimum of the two quantities above

How does Floyd-Warshall work?

- We have the following recurrences and base cases
  - $f(i,j,0) = \text{cost}(i,j)$
  - $f(i,j,k) = \min\{f(i,k,k-1) + f(k,j,k-1),$
    $f(i,j,k-1)\}$
- From the values of $f(\cdot,\cdot,k-1)$, we can calculate $f(\cdot,\cdot,k)$
  - It turns out that we don't need a separate matrix for each k; overwriting the existing values is fine
- That's how we get Floyd-Warshall algorithm

Dijkstra's Algorithm

- Given a directed weighted graph G and a source s
  - Important: The edge weights have to be nonnegative!
- Outputs a vector d where d_i is the shortest distance from s to node i
- Time complexity depends on the implementation:
  - Can be $O(n^2 + m)$, $O(m \log n)$, $O(n \log n)$
- Very similar to Prim's algorithm
- Intuition: Find the closest node to s, and then the second closest one, then the third, etc.

Dijkstra's Algorithm

- Maintain a set of nodes S, the shortest distances to which are decided
- Also maintain a vector d, the shortest distance estimate from s
- Initially, $S = \{s\}$, and $d_v = \text{cost}(s, v)$
- Repeat until $S = V$:
  - Find $v \notin S$ with the smallest d_v, and add it to S
  - For each edge $v \to u$ of cost c:
    - $d_u = \min(d_u, d_v + c)$

Bellman-Ford Algorithm

- Given a directed weighted graph G and a source s
- Outputs a vector d where d_i is the shortest distance from s to node i
- Can detect a negative-weight cycle
- Runs in $\Theta(nm)$ time
- Extremely easy to code
  - Coding time less than a few minutes

Bellman-Ford Pseudocode

- Initialize $d_s = 0$ and $d_v = \infty$ for all $v \neq s$
- For $k = 1 \ldots n - 1$:
  - For each edge $u \to v$ of cost c:
    - $d_v = \min(d_v, d_u + c)$
- For each edge $u \to v$ of cost c:
  - If $d_v > d_u + c$:
    - Then the graph contains a negative-weight cycle

Why does Bellman-Ford work?

- A shortest path can have at most $n-1$ edges
- At the kth iteration, all shortest paths of k or less edges are computed
- After $n-1$ iterations, all distances are final: for every edge $u \to v$ of cost c, $d_v \leq d_u + c$ holds
  - Unless there is a negative-weight cycle
  - This is how the negative-weight cycle detection works

System of Difference Constraints

- Given m inequalities of the form $x_i - x_j \leq c$
- Want to find real numbers $x_1, \ldots, x_n$ that satisfy all the given inequalities

- Seemingly this has nothing to do with shortest paths
  - But it can be solved using Bellman-Ford

Graph Construction

- Create node i for every variable x_i
- Make an imaginary source node s
- Create zero-weight edges from s to all other nodes
- Rewrite the given inequalities as $x_i \leq x_j + c$
  - For each of these constraint, make an edge from j to i with weight c

- Now we run Bellman-Ford using s as the source

What happens?

- For every edge $j \to i$ with cost c, the shortest distance d vector will satisfy $d_i \leq d_j + c$
  - Setting $x_i = d_i$ gives a solution!
- What if there is a negative-weight cycle?
  - Assume that $1 \to 2 \to \cdots k \to 1$ is a negative-weight cycle
  - From our construction, the given constraints contain $x_2 \leq x_1 + c_1$, $x_3 \leq x_2 + c_2$, etc.
  - Adding all of them gives $0 \leq (\text{something negative})$
  - i.e. the given constraints were impossible to satisfy

System of Difference Constraints

- It turns out that our solution minimizes the *span* of the variables: $\max x_i - \min x_i$
  - We won't prove it
  - This is a big hint on POJ 3169!