Dynamic Programming

Subhashis Majumder

Prof. & HoD, CSE; Dean UG

Main facets of Dynamic Programming (DP)

- Has similarity with Divide and Conquer (D & C)
- D & C partitions problems into **independent subproblems**, solves them recursively, then combines the solutions
- DP is applicable when the sub-problems are **dependent**.
- DP will solve every subsubproblem **only once** and will **save its answer in a table**
- DP is typically applied to **optimization problems**

The 4 Steps of DP

- **Characterize the structure** of an optimal solution
- **Recursively** define the **value** of an optimal solution
- Compute the value of an optimal solution in a **bottom-up** fashion
- Construct an optimal solution from **already computed** information

All pair Shortest Paths

- G = (V, E) is a weighted directed graph with weight function w: E → R.
- Want to output a table so that the entry in u's row and v's column will be the weight of a shortest path from u to v.
- If we use Dijkstra's algorithm, and if min-priority queue is implemented
  - i) as array, then $O(V^3 + VE) = O(V^3)$
  - ii) as binary heap – $O(VE \lg V)$ (improvement for sparse graph)
  - iii) Fibonacci heap – $O(V^2 \lg V + VE)$

Other alternatives

- If we use Bellman Ford algorithm – then advantage is that –ve weight cycles can now be present – for dense graphs $O(V^2 E) = O(V^4)$.
- 3 algortihms for APSP -
- i) Matrix multiplication algorithm
- ii) Floyd Warshal algorithm
- iii) Johnson’s algo for sparse graphs
- First 2 algorithms use adjacency matrices.

Starting Basics

- Assume vertices are numbered 1, 2,..., n = |V|, so that input is a n x n matrix W representing the edge-weights of an n-vertex directed graph G = (V, E), W = (w_{ij})
- w_{ij}= 0 if i = j

  = ∞ if i ≠ j, (i, j) does not belong to E

  = the weight of the directed edge (i, j), if i ≠ j

  and (i, j) belongs to E

Some more trivia

- Assume –ve weight edges are allowed, but no –ve weight cycles are allowed
- Output is an n x n matrix $D = (d_{ij})$, where at termination the matrix becomes $d_{ij} = \delta(i, j)$
- Also will compute predecessor matrix $\pi = (\pi_{ij})$, where π_{ij} = nil if either $i = j$, or there is no path from i to j, otherwise it is the predecessor of j in some shortest path from i

Predecessor Subgraphs

- The subgraph induced by the ith row of the π matrix is the shortest paths tree with root i
- For each vertex i belonging to V, we define the predecessor subgraph of G for i (root) as –
- $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$ where
- $V_{\pi,i} = \{j$ belongs to $V : \pi_{ij} \neq nil\} \cup \{i\}$ and
- $E_{\pi,i} = \{(\pi_{ij}, j) : j$ belongs to $V_{\pi,i} - \{i\}\}$
- $G_{\pi,i}$ gives a shortest paths tree with i as root

Procedure: PrintAllPairShortestPaths(π, i, j)

if i = j

    then print I

    else if π_{ij} = nil

        then print “no path from i to j”

        else   PrintAllPairShortestPaths(π, i, π_{ij})

                print j

- Dynamic Programming algo based on Matrix Multiplication takes $O(V^3 \log V)$ time
- However Floyd-Warshal algo takes $O(V^3)$ time

Floyd-Warshal Algorithm

- Assumption - No negative cycles are present
- Structure of a Shortest Path –
- Intermediate vertex of a simple path $p = \langle v_1, v_2, \ldots v_l \rangle$ is any vertex p other than v_1 or v_l , i.e., any vertex from the set $\langle v_2, v_3, \ldots v_{l-1} \rangle$
- Let $V = \{1, 2, \ldots.., n\}$, consider $\{1, 2, \ldots, k\}$, a subset of V.
- For any pair $i, j \in V$, consider paths from i to j where intermediate vertices are drawn from $\{1, 2, \ldots, k\}$ and let p be a minimum weight simple i->j path from among them

F-W algo contd.

- Floyd Warshal algo exploits whether vertex k is present or not as an intermediate vertex of p.
- If k is not an intermediate vertex of p, then a shortest path from i to j with all intermediate vertices in {1, 2, …, k – 1} is also a shortest path with intermediate vertices in the set {1, 2, …, k}
- Otherwise if k is an intermediate vertex of path p, then we break down p into 2 subpaths p1 from i $\longrightarrow$ k and p2 from k $\longrightarrow$ j where p1 and p2 are paths with all intermediate vertices from {1, 2, …, k – 1}

- Let $d_{ij}^{(k)}$ = shortest path distance from i to j with all intermediate vertices in the set {1, 2, …, k},
- When k = 0, there is no intermediate vertices at all
- Hence $d_{ij}^{(0)} = w_{ij}$ (such a path has at most one edge)
- Recursive solution to all-pair-shortest-path-problem
- $d_{ij}^{(k)} = w_{ij}$ if k = 0
- $\min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
- The matrix $D^{(n)} = d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$

Computing Shortest Path Bottom-up

Floyd-Warshal(W)

n <- rows[W]

D $^{(0)}$ <- W

for k <- 1 to n

do for i <- 1 to n

    do for j <- 1 to n

        do $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

return D $^{(n)}$

- Complexity – $O(n^3)$

Constructing a Shortest Path

- Construct a matrix Π while computing D, like $\pi^{(0)}$, $\pi^{(1)}$, …, $\pi^{(n)} = \Pi$, where
- $\pi_{ij}^{(k)}$ is defined as predecessor of vertex j on the shortest path from i to j with all intermediate vertices in the set {1, 2, …, k}
- $\pi_{ij}^{(0)}$ = nil if i = j or $w_{ij} = \infty$

  = i if $i \neq j$ and $w_{ij} < \infty$
- Now, for $k \geq 1$, if we choose a path going through k while going from i to j where $k \neq j$, choose the predecessor of j in path from k to j with vertices from {1,2, …, k -1}

Final Predecessor Expression

- $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$ if $d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

  $= \pi_{kj}^{(k-1)}$ if $d_{ij}^{(k-1)} \ge d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

- Transitive Closure of G is called G* = (V, E*)

  where E* = {(i, j): there is a path p from i to j in G}

- We can give a $O(n^3)$ algorithm by assigning weight = 1 to each edge of G and run Floyd Warshal algorithm.