CS 430 Lecture 29 Activities

All-Pairs Shortest Paths Problem

- Given a directed graph $G = (V, E)$, weight function $w : E \to R$, $|V| = n$,
- Goal: create an $n \times n$ matrix of shortest-path distances from every vertex to every other vertex $\delta(u, v)$,
- Could run Bellman-Ford once from each vertex:
  - $O(|V|^2|E|)$ which is $O(|V|^4)$ if the graph is dense ($|E| \approxeq |V|^2$).
- If no negative-weight edges, could run Dijkstra's algorithm once from each vertex:
  - $O(|V||E| \lg |V|)$ with binary heap–$O(|V|^3 \lg |V|)$ if dense.
- We'll see how to do in $O(|V|^3)$ in all cases with dynamic programming (we have already shown the shortest path problem has optimal substructure.)

The formal problem statement:

- Assume that G is given as an adjacency matrix of weights: $W = (w_{ij})$, with vertices numbered 1 to n.
$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \text{weight of } (i, j) & \text{if } i \neq j, (i, j) \in E, \\ \infty & \text{if } i \neq j, (i, j) \notin E, \end{cases}$$
- Output is the shortest path matrix $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$.

Dynamic Programming Steps

1. Define structure of optimal solution, including what are the largest sub-problems.
2. Recursively define optimal solution
3. Compute solution using table bottom up
4. Construct Optimal solution

To help us develop the first dynamic programming approach, we can restate the All-Pairs Shortest Paths problems as follow.

Find the shortest path from every vertex to every other vertex considering at most paths of $|V|-1$ edges (longest simple path for $|V|$ vertices).

1. Define structure of optimal solution.
2. Recursively define optimal solution

Slow All-Pairs Shortest Paths Algorithm

Algorithm 29.1 Slow All-Pairs Shortest Paths Algorithm

Compute a solution bottom-up: Compute $L^{(1)} = W$, then $L^{(2)}$ from $L^{(1)}$, etc..., $L^{(n-1)}$

function EXTEND(L, W, n)
  $L' \leftarrow$ an $n \times n$ matrix
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to n **do**
      $L'_{ij} \leftarrow \infty$
    end for
    for $k \leftarrow 1$ to n **do**
      $L'_{ij} \leftarrow \min(L'_{ij},\ L_{ik} + W_{kj})$
    end for
  end for
  return L'
end function

function SLOW-APSP(W, n)
  $L^{(1)} \leftarrow W$
  for $m \leftarrow 2$ to $n-1$ **do**
    $L^{(m)} \leftarrow$ EXTEND$(L^{(m-1)}, W, n)$
  end for
  return $L^{(n-1)}$
end function

3. What is the runtime of EXTEND and SLOW-ASPS?

Improving on SLOW-ASPS

Note the code to multiply two $n \times n$ matrices (AB) together to get C, an $n \times n$ matrix.

Algorithm 29.2 Multiply Matricies

function MULTIPLY-MATRICIES(A, B)
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to n **do**
      $C_{ij} \leftarrow 0$
      for $j \leftarrow 1$ to n **do**
        $C_{ij} \leftarrow C_{ij} + A_{ik}B_{kj}$
      end for
    end for
  end for
end function

Table 29.1: The shortest path containing two edges.

$L^{(2)}$	A	B	C	D	E
A	0	3_A	8_A	2_E	-4_A
B	3_D	0	-4_D	1	7
C	∞	4	0	5_B	11_B
D	2	-1_C	-5	0	-2_A
E	∞	∞	∞	6	0

Augmented with ∞ when no edge exists

4. How does this matrix multiply code compare to the EXTEND code? Why do we care? $O(|V|^4)$

Faster All-Pairs Shortest Paths Algorithm

Algorithm 29.3 Faster All-Pairs Shortest Paths Algorithm

```
Compute a solution bottom-up: Compute L^(1) = W, then L^(2) from L^(1), then L^(4) from L^(2),
    etc..., L^(n-1)
function FASTER-APSP(W, n)
    L^(1) ← W
    m ← 1
    while m < n − 1 do
        L^(2m) ← EXTEND(L^(m), L^(m), n)
        m ← 2m
    end while
    return L^(m)
end function
```

5. What is the runtime of FASTER-ASPS? $O(|V|^3 \lg |V|)$

Floyd-Warshall Algorithm

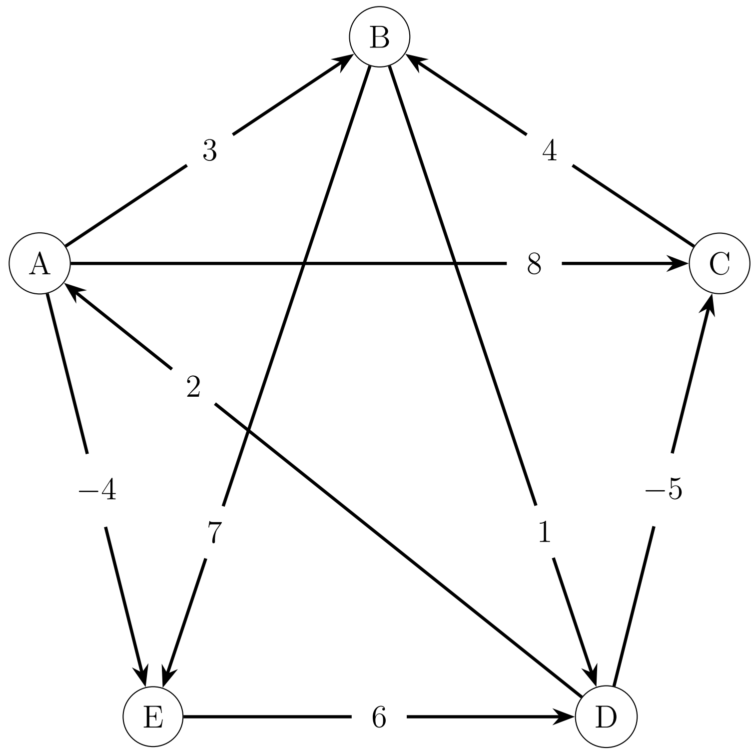
To help us develop another dynamic programming approach, we can state the All-Pairs Shortest Paths problem as follows:
Find the shortest path from every vertex to every other vertex considering at most all other vertices intermediate on the paths.

6. Define structure of optimal solution. Assume optimal shortest path with possibly all other vertices along the path at most $|V| - 2$. Remove a vertex from k along the path, k used or k not used.

7. Recursively define optimal solution and write pseudocode.

8. What is the run time of Floyd-Warshall?

9. Demonstrate Floyd-Warshall.



An example of Floyd-Warshall: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16