Dynamic Programming

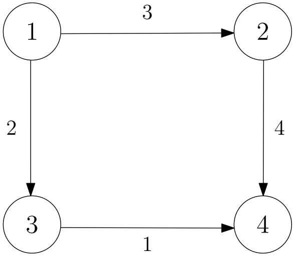
- All Pairs Shortest Paths Problem
- APSP: Dynamic Programming Formulation
- Floyd Warshall Algorithm

IMDAD ULLAH KHAN

APSP Problem

Input: A weighted graph $G = (V, E, w)$
Output: Shortest paths from every vertex $u \in V$ to every other $v \in V$

The APSP problem can be represented by a $n \times n$ matrix $D = [d_{ij}]$, where $d_{ij} = d(u_i, u_j)$ for $i, j = 1, \ldots, n$, and n is the number of vertices in V.



	1	2	3	4
1	0	3	2	3
2	∞	0	∞	4
3	∞	∞	0	1
4	∞	∞	∞	0

The goal is to compute the matrix D efficiently.

APSP: Dynamic Programming Formulation

Dynamic programming idea: For any pair of vertices (i, j), consider all possible intermediate vertices k that lie on a shortest path from i to j

Fix some ordering on vertices

Let $d_{ij}^{(k)}$ denote the length of a shortest path from i to j that only uses vertices $\{1, 2, \ldots, k\}$ as intermediate vertices

Then we have the following recurrence relation:

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

The base case is $d_{ij}^{(0)} = w_{ij}$, where w_{ij} is the weight of the edge (i, j), or ∞ if there is no such edge

The final solution is $d_{ij}^{(n)}$, where n is the number of vertices in the graph

Floyd-Warshall Algorithm: Implementation with 3d arrays

Floyd Warshall algorithm can be implemented with a 3-d ($n \times n \times n$) array A to store the intermediate results

- $A[i][j][k]$ represents the shortest path from i to j using only vertices 1 to k as intermediate vertices
- Initially $A[i][j][0]$ is the same as the adjacency matrix of the graph, with ∞ representing no edge between two vertices
- Finally $A[i][j][n]$ gives the shortest path between all pairs of vertices

Floyd-Warshall Algorithm: Implementation with 3d arrays

Algorithm 1 Floyd Warshall Algorithm using 3d Matrix

```
$n \leftarrow$ number of vertices in $G$
$A \leftarrow$ new $n \times n \times n$ matrix
for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do
        $A[i][j][0] \leftarrow w_{ij}$ ▷ where $w_{ij}$ is the weight of the edge $(i, j)$, or $\infty$ if there is no such edge
for $k = 1$ to $n$ do
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            $A[i][j][k] \leftarrow \min\{A[i][j][k-1], A[i][k][k-1] + A[k][j][k-1]\}$
return $A[i][j][n]$ for all $i, j$
```

Floyd-Warshall Algorithm: Implementation with 3d arrays

- The time complexity of the Floyd Warshall algorithm with a 3d matrix is $O(n^3)$
- Because we have three nested loops, each iterating from 1 to n, and each iteration performs a constant amount of work
- The space complexity of the Floyd Warshall algorithm using a 3d matrix is also $O(n^3)$

Floyd-Warshall Algorithm: Implementation with 2d array

Floyd Warshall algorithm can also be implemented with a 2-d $(n \times n)$ array D to store the intermediate results

- $D[i][j]$ represents the shortest path from i to j using any intermediate vertices
- Initially D is the same as the adjacency matrix of the graph, with ∞ representing no edge between two vertices
- Finally D gives the shortest path between all pairs of vertices

Floyd-Warshall Algorithm: Implementation with 2d array

Algorithm 2 Floyd Warshall Algorithm using 2d Matrix

```
$n \leftarrow$ number of vertices in $G$
$D \leftarrow$ matrix of edge weights of $G$
for $k = 1$ to $n$ do
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            $D[i][j] \leftarrow \min\{D[i][j], D[i][k] + D[k][j]\}$
return $D$
```

Floyd-Warshall Algorithm: Implementation with 2d array

- The time complexity of the Floyd Warshall algorithm using a 2d matrix is $O(n^3)$
- Because we have three nested loops, each iterating from 1 to n, and each iteration performs a constant amount of work
- The space complexity of the Floyd Warshall algorithm using a 2d matrix is $O(n^2)$
- Because we need to store n^2 elements in the matrix D
- This is an improvement over the 3d matrix implementation, which requires $O(n^3)$ space