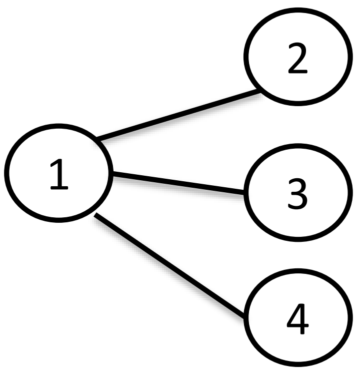
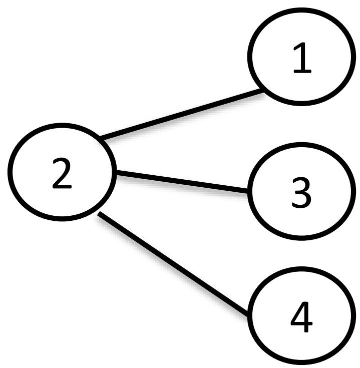
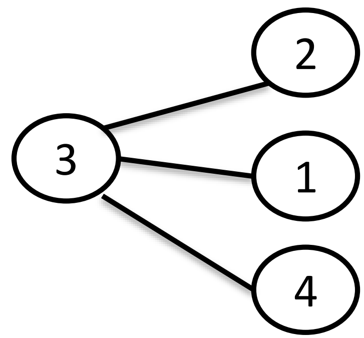
All Pairs shortest paths using Dynamic Programming

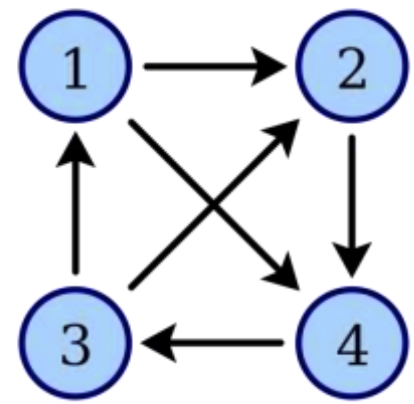
Problem:

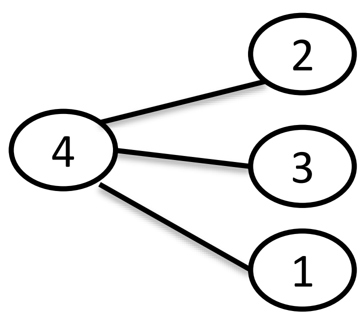
Given a weighted digraph G=(V,E) determine the length of the shortest path (i.e., distance) between all pairs of vertices in G . Here we assume that there are no cycles with zero or negative cost.











Solutions

Dijkstra's algorithm

- It’s a single source shortest path algorithm ie single vertex to all other vertices
- Time complexity $O(|E| \log |V|)$
- If graph is dense then $E= |V^2|$ then complexity is $O(|V^2| \log|V|)$
- If this algorithm runs for all vertices in the graph then complexity will become $O(|V^3| \log|V|)$
- Will not work for negative edge weights

Solutions

Bellman – Ford algorithm

- Slower than Dijkstra's algorithm but allows negative edge weights
- It's a single source shortest path algorithm ie single vertex to all other vertices

- Time complexity O(|E| |V|)
- If graph is dense then E= $|V^2|$ then complexity is O($|V^3|$)
- If this algorithm runs for all vertices in the graph then complexity will become O($|V^4|$)
- Will not work for negative cycles in the graph

Solutions
Floyd-Warshall algorithm

- It helps to find the shortest path in a weighted graph with positive or negative edge weights.
- A single execution of the algorithm is sufficient to find the lengths of the shortest paths between all pairs of vertices.
- Time complexity $O(|V^3|)$
- Negative-weight edges may be present, but no negative-weight cycles.

Floyd-Warshall algorithm

The all-pairs shortest-path problem is to determine a matrix D such that d(i , j) is the length of a shortest path from i to j.

$$D \qquad \begin{array}{cc} & \xrightarrow{\qquad\quad i \qquad\quad} \\ j \Big\downarrow & \begin{bmatrix} 0 & 3 & 8 & \infty \\ \infty & 0 & 4 & 11 \\ \infty & \infty & 0 & 7 \\ 4 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$d_{ij} = \begin{cases} 0 & if\ i = j \\ d(i,j) & if\ \ i\ \neq j\ and\ \ (i,j)\ \in E \\ \infty & if i\ \neq j\ and\ \ (i,j)\ \notin\ E \end{cases}$$

Floyd-Warshall algorithm

Step 1 : Decomposition

Definition: The vertices $v_2, v_3, ..., v_{l-1}$ are called the *intermediate vertices* of the path $p = \langle v_1, v_2, ..., v_{l-1}, v_l \rangle$.

- Let $d_{ij}^{(k)}$ be the length of the shortest path from i to j such that *all* intermediate vertices on the path (if any) are in set $\{1, 2, \ldots, k\}$.

  $d_{ij}^{(0)}$ is set to be w_{ij}, i.e., no intermediate vertex.
  Let $D^{(k)}$ be the $n \times n$ matrix $[d_{ij}^{(k)}]$.

- Claim: $d_{ij}^{(n)}$ is the distance from i to j. So our aim is to compute $D^{(n)}$.
- Subproblems: compute $D^{(k)}$ for $k = 0, 1, \cdots, n$.

Floyd-Warshall algorithm

Step 2: Structure of shortest paths

Observation 1: A shortest path does not contain the same vertex twice. Proof: A path containing the same vertex twice contains a cycle. Removing cycle gives a shorter path.

Observation 2: For a shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$, there are two possibilities:

1. k is not a vertex on the path,
   The shortest such path has length $d_{ij}^{(k-1)}$.

2. k is a vertex on the path.
   The shortest such path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Shortest path
from i to k

Shortest path
from k to j

Floyd-Warshall algorithm

Consider a shortest path from i to j containing the vertex k. It consists of a subpath from i to k and a subpath from k to j.
Each subpath can only contain intermediate vertices in $\{1, ..., k-1\}$, and must be as short as possible, namely they have lengths $d_{ik}^{(k-1)}$ and $d_{kj}^{(k-1)}$.

Hence the path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Combining the two cases we get

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

Floyd-Warshall algorithm

Step 3: the Bottom-up Computation

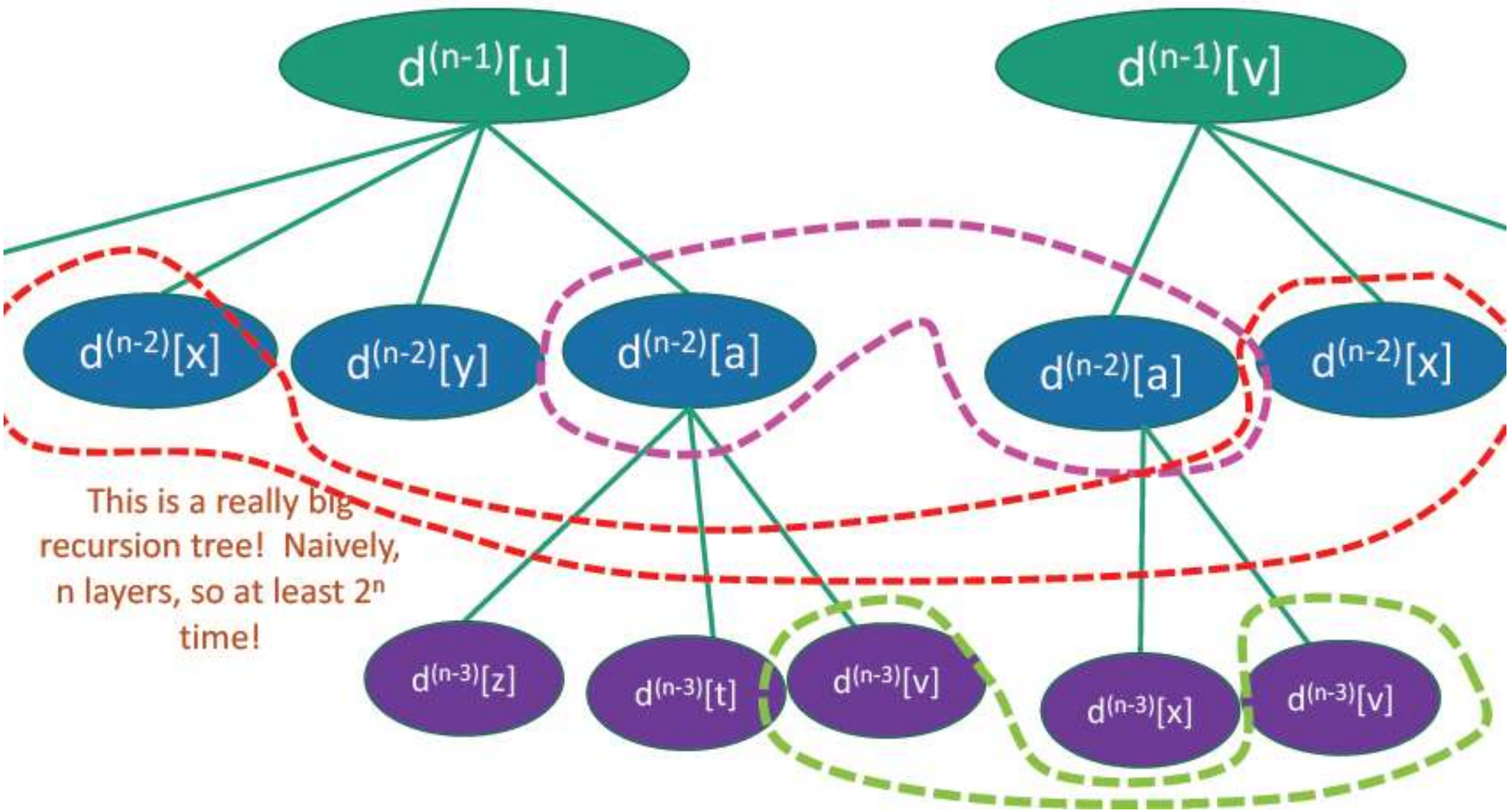
- Bottom: $D^{(0)} = [w_{ij}]$, the weight matrix.

- Compute $D^{(k)}$ from $D^{(k-1)}$ using

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

for $k = 1, ..., n$.

Do we have any sub optimal structure and overlapping sub problems



Floyd-Warshall algorithm

Floyd-Warshall(w, n)

{ for $i = 1$ to n do　　　　initialize

　　for $j = 1$ to n do

　　{ $d[i, j] = w[i, j]$;

　　　$pred[i, j] = nil$;

　　}

　for $k = 1$ to n do　　　　dynamic programming

　　for $i = 1$ to n do

　　　for $j = 1$ to n do

　　　　if $(d[i, k] + d[k, j] < d[i, j])$

　　　　　$\{d[i, j] = d[i, k] + d[k, j]$;

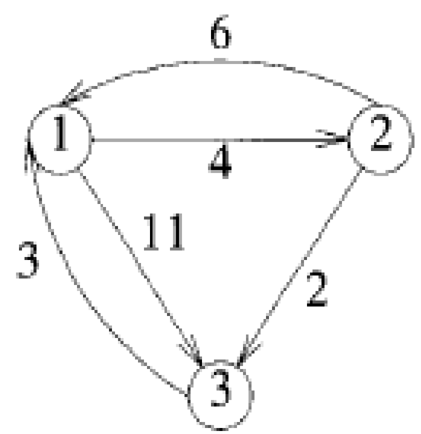
　　　　　$pred[i, j] = k;\}$

　return $d[1..n, 1..n]$;

}

Floyd-Warshall algorithm

```
Algorithm AllPaths(cost, A, n)
// cost[1 : n, 1 : n] is the cost adjacency matrix of a graph with
// n vertices; A[i, j] is the cost of a shortest path from vertex
// i to vertex j. cost[i, i] = 0.0, for 1 ≤ i ≤ n.
{
    for i := 1 to n do
        for j := 1 to n do
            A[i, j] := cost[i, j]; // Copy cost into A.
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                A[i, j] := min(A[i, j], A[i, k] + A[k, j]);
}
```



(a) Example digraph

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

(b) A^0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

(c) A^1

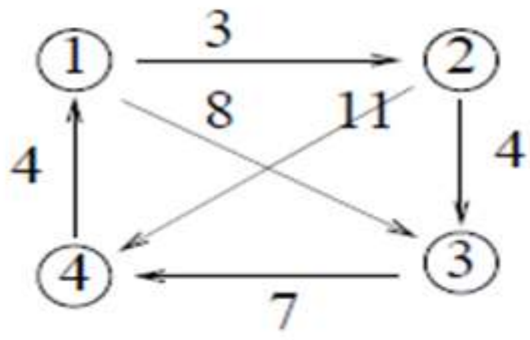
A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

(d) A^2

A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

(e) A^3

EXAMPLE



```
Algorithm AllPaths(cost, A, n)
// cost[1 : n, 1 : n] is the cost adjacency matrix of a graph with
// n vertices; A[i, j] is the cost of a shortest path from vertex
// i to vertex j. cost[i, i] = 0.0, for 1 ≤ i ≤ n.
{
    for i := 1 to n do
        for j := 1 to n do
            A[i, j] := cost[i, j]; // Copy cost into A.
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                A[i, j] := min(A[i, j], A[i, k] + A[k, j]);
}
```

Time Complexity : $O(n^3)$