Single Source Shortest Paths

What is the least-cost solution for getting from point A to point B?

Shortest Paths Problem

Given a weighted, directed graph G = (V, E) with edge weights w and a path definition of p = $< v_0, v_1, .., v_k >$ with weight w(p) = $\Sigma_{i=1}^{k} w(v_{i-1}, v_i)$, find the shortest path weight from vertex u to vertex v

$$\delta(u,v) = \begin{cases} min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

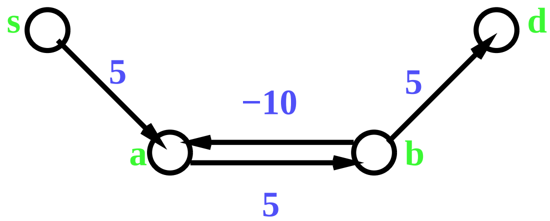
____________________________ worked on unweighted (unit weight) graphs.

Variants

- __________________ — shortest path from some vertex s to every other vertex
- _________________________ — shortest path to some vertex d from every other vertex (reverse direction on edges and run single source algorithm)
- _______________ — shortest path from u to v (run single source algorithm with s = u; nothing better asymptotically)
- ____________ — shortest path from u to v for every pair of vertices (single source for every vertex, but can do better)

Negative Weights

We encounter problems when the graphs include negative cycles.



Shortest Path from s to d can become arbitrarily short (there is no shortest path).

Dijkstra's Algorithm: nonnegative weights

Bellman-Ford: negative weights are okay; detects negative cycles

---

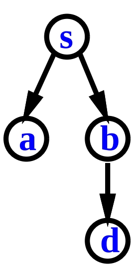
Shortest Paths Tree (single source)

Predecessor Subgraph $G_{pred} = (V_{pred}, E_{pred})$ for G = (V, E).

$$V_{pred} = \{v \in V \mid pred(v) \neq NIL\} \cup \{s\}$$

$$E_{pred} = \{(pred(v), v) \in E \mid v \in V_{pred} - \{s\}\}$$

The unique simple path from s to v in G_{pred} is a shortest path from s to v in G.



Shortest Path Problem Has Optimal Substructure

Corollary 25.2

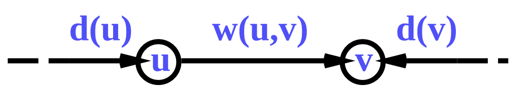
If shortest path p from s to v can be decomposed into $s \overset{p'}{\leadsto} u \rightarrow v$, then p' is a shortest path from s to u, and $\delta(s,v) = \delta(s,u) + w(u,v)$.

Because the problem has optimal substructure, we can try to use dynamic programming and greedy algorithms.

Relaxation

This is a tightening of the upper bound d(v) on the shortest path weight from s to v.

Maintain d(v) and pred(v) for each vertex v.




```
Relax(u, v, w)
    if d(v) > d(u) + w(u,v)
    then d(v) = d(u) + w(u,v)
        pred(v) = u

Init-Single-Source(G, s)          ; G = (V, E)
    foreach v in V
        d(v) = ∞
        pred(v) = NIL
    d(s) = 0
```

Lemmas 25.4 – 25.9 show Relaxing after Init-Single-Source will eventually reach the shortest path weight and predecessor graph will be a shortest path tree (assuming no negative-weight cycles).

Dijkstra's Algorithm [1959] (nonnegative weights only)

1. Q = V
2. S = {}
3. repeat
4. select a vertex u from Q
5. Use $\delta(s,u)$ to update other values
6. S = S $\cup$ {u}
7. until Q = {}

- Use priority queue for Q with d(v) as the key
  - Extract-Min to select from Q
  - Decrease-Key to change d(v)
- Greedy choice for next vertex to add to S (contains vertices whose shortest path is known)
  - Because of optimal substructure

Dijkstra's Algorithm

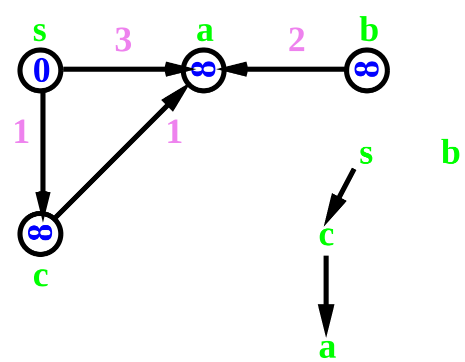
```
Dijkstra(G, w, s)                              ANALYSIS
  Init-Single-Source(G, s)                      O(V)
  Q = V
```

```
(S = {})
while Q <> {}                 Q (LINEAR  Q (BINARY)  Q (FIBONAC
   u = Extract-Min(Q)            ARRAY)      HEAP)       HEAP)
   (S = S u {u})                 O(V^2)      VlgV        VlgV
   foreach v in Adj(u)
      Relax(u, v, w)              O(E)       ElgV         E
                              ---------------------------------
                               O(V^2 + E)  O((V+E)lgV) O(E+VlgV
                                                       amortize
```

Theorem 25.10 and Corollary 25.11:

Dijkstra's algorithm produces a shortest path tree.

Example



S = {}	0 ∞ ∞ ∞ Q = {s, a, b, c}
0 S = {s}	3 ∞ 1 Q = {a, b, c}
0, 1 S = {s, c}	2 ∞ Q = {a, b}
0, 1, 2 S = {s, c, a}	∞ Q = {b}
0, 1, 2, ∞ S = {s, c, a, b}	

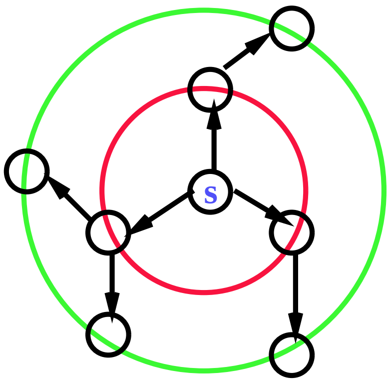
Bellman-Ford Algorithm

Accepts negative weights

Detects negative cycles

Algorithm: Relax edges rippling from source.

At end, if d(v) > d(u) + w(u,v), then ___________________



Pseudocode

```
Bellman-Ford(G, w, s)                                    ANALYSIS
   Init-Single-Source(G, s)                               O(V)
   for i = 1 to (|V| - 1)                                 O(VE)
      foreach (u,v) in E
         Relax(u, v, w)
   foreach edge (u,v) in E                                O(E)
      if d(v) > d(u) + w(u,v)
      then return false
   return true
                                                          -----
                                                          O(VE)
```

Theorem 25.14

If Bellman-Ford returns true, then G_{pred} forms a shortest path tree, else there exists a negative weight cycle.

Examples

Consider edges in the order s → a, s → c, c → a, b → a.

Click mouse to advance to next frame.

More Examples

Consider edges in the order s → a, a → b, b → a, b → c.

$$d(b) > d(a) + w(a,b)$$
$$2 > 0 + 1$$

Negative weight cycle!

Shortest Paths in DAGs

```
DAG-SS(G, w, s)
    Topologically sort V        ; Θ(V + E)
    Init-SS(G, s)               ; Θ(V)
    foreach u in V in order     ; Θ(E)
        foreach v in Adj(u)
            Relax(u, v, w)
```

Running Time: $\Theta(V + E)$

Application: PERT chart

All-Pairs Shortest Paths

Problem: Given a directed graph G = (V,E) with weight function w: E → R mapping edges to real-valued weights, find the shortest path between every pair of vertices u,v ∈ V that minimizes the sum of the weights along the edges of the path.

Solution 1: Run ________________________________ on every vertex.

Dijkstra, O(VE lg V) using binary heap and priority queue

Bellman-Ford, O(V^2E)

Note that E = O(V^2) in the worst case.

Solution 2: ______________________ approach

Matrix Multiplication Approach with repeated squaring, $\Theta(V^3 lgV)$

Floyd-Warshall, $\Theta(V^3)$

Application: Mileage chart for a road atlas.

Notation

Algorithms use an **adjacency matrix** w to represent graphs, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of directed edge} & \text{if } i \neq j \text{ and } (i,j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

Negative weights are allowed, but not negative cycles.

Algorithms output a **distance matrix** D, where d_{ij} is the weight of the shortest path from i to j.

For example, $d_{ij} = \delta(i, j)$.

Algorithms may also output **predecessor matrix** Π, where π_{ij} is NIL if either i=j or there is no path from i to j; otherwise, π_{ij} is the predecessor of j on a shortest path from i.

From the predecessor matrix Π, we can derive the **predecessor subgraphs** $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$ for each vertex i $\in$ V as

$$V_{\pi,i} = \{j \in V \mid \pi_{i,j} \neq NIL\} \cup \{i\}$$

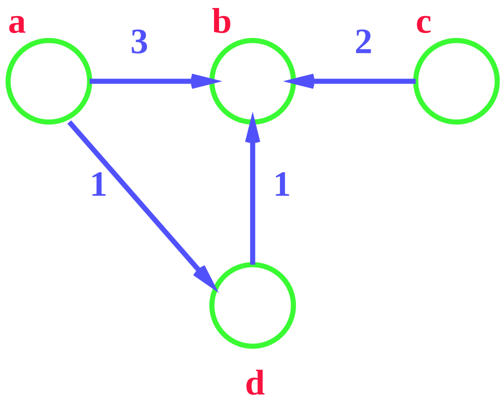
$$E_{\pi,i} = \{(\pi_{i,j}, j) \mid j \in V_{\pi,i} \text{ and } \pi_{i,j} \neq NIL\}$$

Pseudocode

Given the predecessor matrix Π, we can print the shortest path from i to j:

```
Print-Path(Π, i, j)
   if i = j
   then print i
   else if π_{i,j} = NIL
      then print "no path exists from i to j"
      else Print-Path(Π, i, π_{i,j})
         print j
```

Example



$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \end{array}$$

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 2 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \end{array}$$

$$\Pi = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} NIL & d & NIL & a \\ NIL & NIL & NIL & NIL \\ NIL & c & NIL & NIL \\ NIL & d & NIL & NIL \end{bmatrix} \end{array}$$

$$G_{\pi,i} = \begin{array}{cccc} a & b & c & d \\ \downarrow & & \downarrow & \downarrow \\ d & & b & b \\ \downarrow & & & \\ b & & & \end{array}$$

Matrix Multiplication Approach

Consider a shortest path p from vertex i to vertex j containing at most m edges.

- If i=j, then p has ___________
- If i≠j, then p = $i \overset{p'}{\rightsquigarrow} k \rightarrow j$, where p' contains m-1 edges

By Lemma 25.1 (subpaths of shortest paths are shortest paths), p' is a shortest path from i to k, and $\delta(i,j) = \delta(i,k) + w_{kj}$.

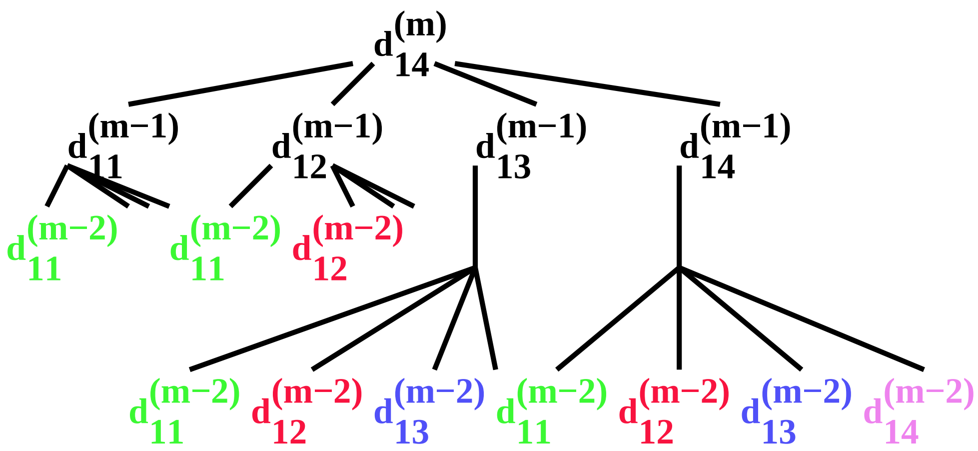
Thus, the shortest path problem exhibits optimal substructure.

Recursive Solution

- $d_{ij}^{(m)}$ = weight of path from i to j containing at most m edges
- $d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$
- $d_{ij}^{(m)} = \min_{1 \leq k \leq n}\{d_{ik}^{(m-1)} + w_{kj}\} \quad m \geq 1,\ n = |V|$
- $\delta(i,j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = ..$

  There are at most ____ edges in the shortest path from i to j assuming no negative weight cycles.

Overlapping Subproblems



Number of unique subproblems: O(V^3)

Apply Dynamic Programming

- Optimal Substructure
- Overlapping Subproblems

Note similarity to matrix multiplication:

$$m_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \min_{1 \le k < j} \{ m_{i,k} + m_{k+1,j} + p_{i-1} p_k p_j \} & \text{if } i \ne j \end{cases}$$

In fact, our algorithm for computing D involves "multiplying" the adjacency matrix by itself n-1 times.

Bottom-Up Strategy

Let $D^{(i)} = W^i$ be the distance matrix after considering paths of length $\leq$ i.

- $D^{(1)} = W$
- $D^{(2)} = D^{(1)} \cdot W = W^2$
- ...
- $D^{(n-1)} = D^{(n-2)} \cdot W = W^{n-1}$

Extending By One More Edge

```
Extend-Shortest-Paths(D,W)                 ; (A, B)
    n = rows(D)
    initialize D'                          ; n x n matrix (C)
    for i = 1 to n
        for j = 1 to n
            d'_ij = ∞                      ; (initialize to 0)
            for k = 1 to n
                d'_ij = min(d'_ij, d_ik + w_kj)   ; (c_ij = c_ij + a_ik · b_kj)
    return D'                              ; (C)
```

```
Slow-APSP(W)
    n = rows(W)
    D^(1) = W
    for m = 2 to n-1
```

$$D^{(m)} = \text{Extend-Shortest-Paths}(D^{(m-1)}, \text{W})$$

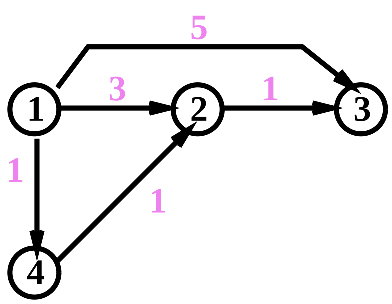
return $D^{(n-1)}$

Running times:

Extend-Shortest-Paths: $\Theta(n^3)$

Slow-APSP: $\Theta(n^4)$

Example



$$D^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \end{array}$$

$$D^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 2 & 4 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & 2 & 0 \end{bmatrix} \end{array}$$

1 2 3 4

$$D^{(3)} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 2 & 3 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & 2 & 0 \end{bmatrix}$$

Fast APSP

Note that $D^{(m)} = D^{(n-1)}$ for all m $\geq$ n-1.

- $D^{(1)} = W$
- $D^{(2)} = W^2 = W.W$
- $D^{(4)} = W^4 = W^2.W^2$
- $D^{(8)} = W^8 = W^4.W^4$

Continue until $D^{2^{\lceil lg(n-1) \rceil}}$ since $2^{\lceil lg(n-1) \rceil} \geq n-1$.

This requires only $\lceil lg(n-1) \rceil$ matrix products (calls to Extend). This is called "repeated squaring".

Fast-APSP(W)
    n = rows(W)
    $D^{(1)}$ = W
    m = 1
    while n-1 > m ; $\lg n$
        $D^{(2m)}$ = Extend-Shortest-Paths($D^{(m)}$, $D^{(m)}$) ; n^3
        m = 2m
    return $D^{(m)}$

Running time: $\Theta(n^3 \lg n)$

Floyd-Warshall APSP Algorithm

- negative weight edges
- no negative cycles (but can be added)

Intermediate Structure Of Shortest Path

An **intermediate** vertex of a simple path p = $< v_1, v_2, .., v_l >$ is any vertex of p other than v_1 and v_l.

The Floyd-Warshall (FW) algorithm works by successively reducing the number of intermediate vertices that can occur in a shortest path and its subpaths.

Let graph G = (V, E) have vertices V numbered 1..n, V = {1, 2, .., n}, and consider a subset {1, 2, .., k} for some k.

Let p be the minimum weight path from vertex i to vertex j whose intermediate vertices are drawn from {1, 2, .., k}. One of two situations then occur:

1. k is not an intermediate vertex of p

$$i \overset{p}{\leadsto} j$$

contains vertices from {1, 2, .., k-1}

2. k is an intermediate vertex of p

$$i \overset{p_1}{\leadsto} k \overset{p_2}{\leadsto} j$$

contains vertices from {1, 2, .., k-1}

p_1 is the shortest path from i to k

p_2 is the shortest path from k to j

(by Lemma 25.1)

Recursive Solution

- $d_{ij}^{(k)}$ = weight of eventually shortest path from i to j with all intermediate vertices in {1, 2, .., k}
-

$$d_{ij}^{(k)} \ = \begin{cases} w_{ij} & \text{if} \ \ k = 0 \\ min(d_{ij}^{(k-1)}, \ d_{ik}^{(k-1)} \ + \ d_{kj}^{(k-1)}) & \text{if} \ \ k \geq 1 \end{cases}$$

- $D^{(n)} \ = \ [d_{ij}^{(n)}] \ = \ [\delta(i,j)]$, n = |V|

Pseudocode

```
Floyd-Warshall(W)
    n = rows(W)
    D^(0) = W
    for k = 1 to n
        for i = 1 to n
            for j = 1 to n
                d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    return D^(n)
```

- Three nested for loops: $\Theta(n^3)$, n = |V|
- Better than $O(V^3 lgV)$ and $O(V^4)$ of SSSP

Constructing Shortest Path

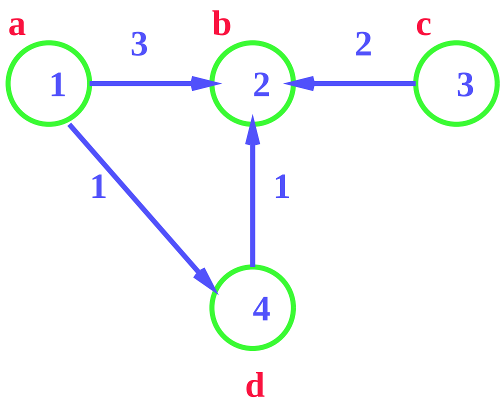
- Want predecessor matrix Π

- Can compute Π from final D matrix, or
- Can compute $\Pi^{(0)}$, .., $\Pi^{(n)}$ as you go

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{otherwise } (i \rightsquigarrow k \rightsquigarrow j) \end{cases}$$

Example



$$W = D^{(0)} = \begin{bmatrix} 0 & 3 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \quad \Pi^{(0)} = \begin{bmatrix} \text{NIL} & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix}$$

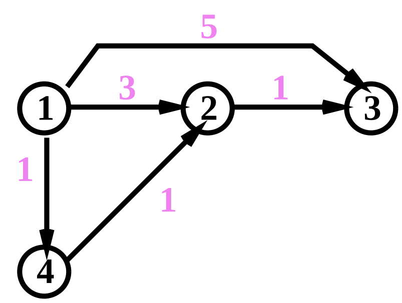
$$D^{(1)} = \begin{bmatrix} 0 & 3 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \quad \Pi^{(1)} = \begin{bmatrix} \text{NIL} & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & 3 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \qquad \Pi^{(2)} = \begin{bmatrix} \text{NIL} & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix}$$

$$D^{(3)} = \begin{bmatrix} 0 & 3 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \qquad \Pi^{(3)} = \begin{bmatrix} \text{NIL} & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 2 & \infty & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 2 & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \qquad \Pi^{(4)} = \begin{bmatrix} \text{NIL} & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix}$$

Example



$$W = D^{(0)} = \begin{bmatrix} 0 & 3 & 5 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \qquad \Pi^{(0)} = \begin{bmatrix} \text{NIL} & 1 & 1 & 1 \\ \text{NIL} & \text{NIL} & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix}$$

$$D^{(1)} = \begin{bmatrix} 0 & 3 & 5 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & \infty & 0 \end{bmatrix} \quad \Pi^{(1)} = \begin{bmatrix} \text{NIL} & 1 & 1 & 1 \\ \text{NIL} & \text{NIL} & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & \text{NIL} & \text{NIL} \end{bmatrix} = \Pi^{(0)}$$

$$D^{(2)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & 2 & 0 \end{bmatrix} \quad \Pi^{(2)} = \begin{bmatrix} \text{NIL} & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & 2 & \text{NIL} \end{bmatrix}$$

$$D^{(3)} = D^{(2)} \quad \Pi^{(3)} = \Pi^{(2)}$$

$$D^{(4)} = \begin{bmatrix} 0 & 2 & 3 & 1 \\ \infty & 0 & 1 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 1 & 2 & 0 \end{bmatrix} \quad \Pi^{(4)} = \begin{bmatrix} \text{NIL} & 4 & 4 & 1 \\ \text{NIL} & \text{NIL} & 2 & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 4 & 2 & \text{NIL} \end{bmatrix}$$

Applications