Single Source Shortest Paths

→ In a shortest-paths problem, a directed graph $G=(V,E)$ with weight function $w: E \to R$ mapping edges to real valued weights.

→ The weight of path $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is the sum of the weight of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

→ Shortest path weight from u to v by

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\}, & \text{If there is a path from } u\text{-}v. \\ \infty, & \text{otherwise.} \end{cases}$$

→ A shortest path from vertex u to vertex v is then defined as any path p with weight

$$w(p) = \delta(u,v)$$

→ Single-source shortest-paths problem: A graph $G=(V,E)$ we want to find a shortest path from a given source vertex $s \in V$ to every vertex $v \in V$

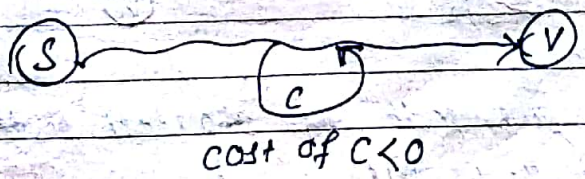
→ Single-destination shortest paths problem: find a shortest path to a given destination vertex t from every vertex v.

→ Single-pair shortest-path problem: find a shortest path from u to v for given vertices u and v.

→ All-pairs shortest-paths problem: find a shortest path from u to v for every pair of vertices u and v.

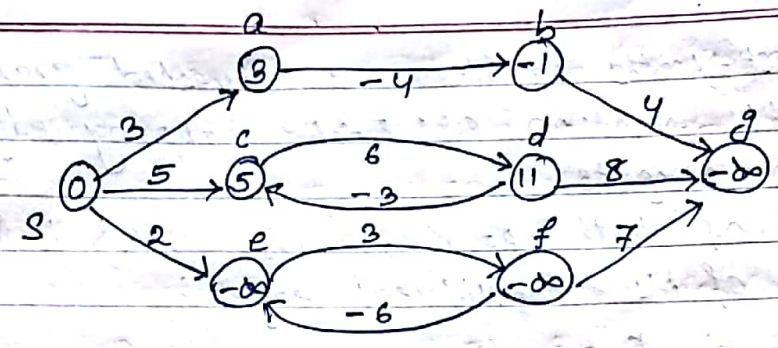
Shortest path : Existance

→ If some path from s to v contains a negative cost cycle then, there doesnot exist a shortest path, Otherwise there exists a shortest path from s to v.



cost of $C < 0$

→ If there is a negative-weight cycle on some path from s to v, we define $\delta(s,v) = -\infty$



→ There are infinitely many paths from s to c: $\langle s,c\rangle, \langle s,c,d,c\rangle, \langle s,c,d,c,d,c\rangle$ and so on.

→ The cycle $\langle c,d,c\rangle$ has weight

$$6+(-3)=3>0$$

So, the shortest path from s to c is $\langle s,c\rangle$ with weight $\delta(s,c)=5$.

→ Similarly, there are infinitely many paths from s to e: $\langle s,e\rangle, \langle s,e,f,e\rangle, \langle s,e,f,e,f,e\rangle$ and so on.

→ The cycle $\langle e,f,e\rangle$ has weight $3+(-6)=-3<0$. Hence, there is no shortest path from s to e so, $\delta(s,e)=-\infty$

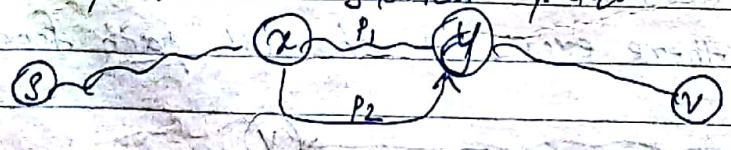
→ Dijkstra's algorithm assume that all edge weights in the input graph are non-negative.

→ Bellman-Ford algorithm, allow negative-weight edges in the input graph and produce a correct answer as long as no negative-weight cycles are reachable from the source.

→ If there is such a negative-weight cycle, the algorithm can detect and report its existence.

Shortest Path: Properties

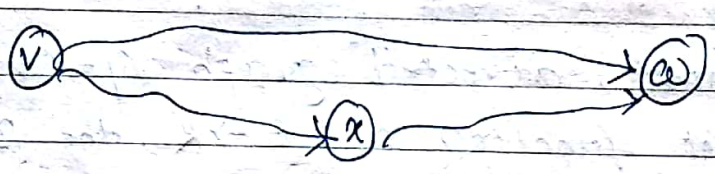
1. Optimal Substructure Property: All sub-paths of shortest paths are shortest paths.



Let p_1 be x-y sub path of shortest path s-v.
Let p_2 be any x-y path.
Then, cost of $p_1 \leq$ cost of p_2, otherwise p is not shortest path.

(2) Triangle inequality: Let $d(v, w)$ be the length of the shortest path from v to w, Then

$$d(v, w) \leq d(v, x) + d(x, w)$$



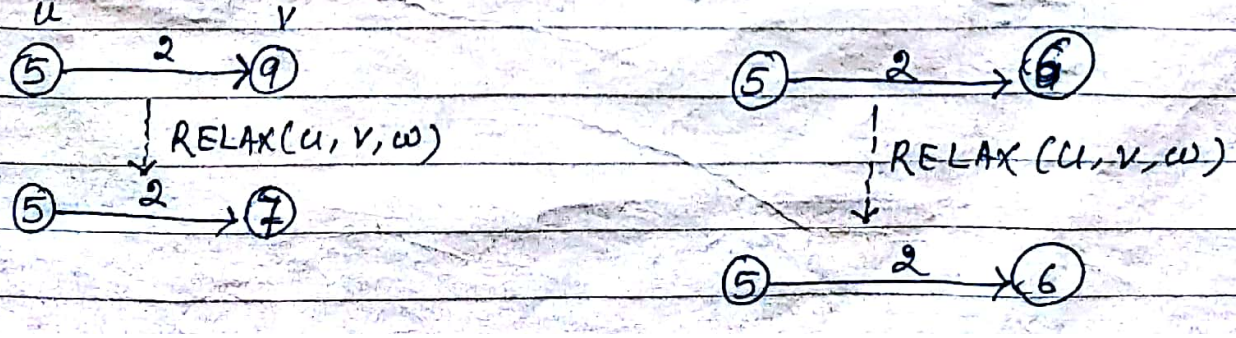
RELAXATION:

→ The single source shortest paths algorithms are based on a technique known as relaxation.

→ Relaxation is a method that repeatedly decreases the upper bound on the actual shortest-path weight of each vertex until the upper bound equals the shortest path weight.

```
INITIALIZE-SINGLE-SOURCE(G, s)
1. for each vertex v ∈ V[G]
2.     do d[v] ← ∞
3.        π[v] ← NIL
4. d[s] ← 0
```

```
RELAX(u, v, w)
1. if d[v] > d[u] + w(u, v)
2.     then d[v] ← d[u] + w(u, v)
3.          π[v] ← u
```



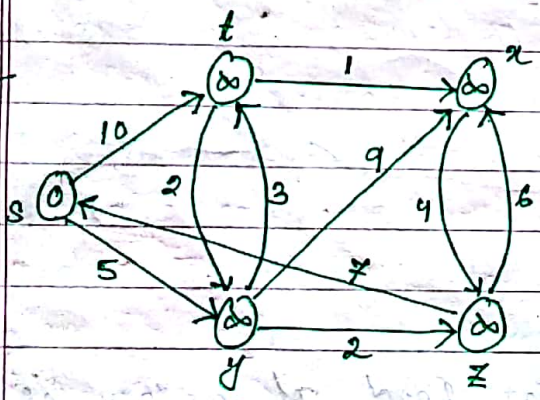
DIJKSTRA'S ALGORITHM (Greedy Approach)

→ Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted directed graph $G=(V,E)$ for which all edge weights are non-negative.

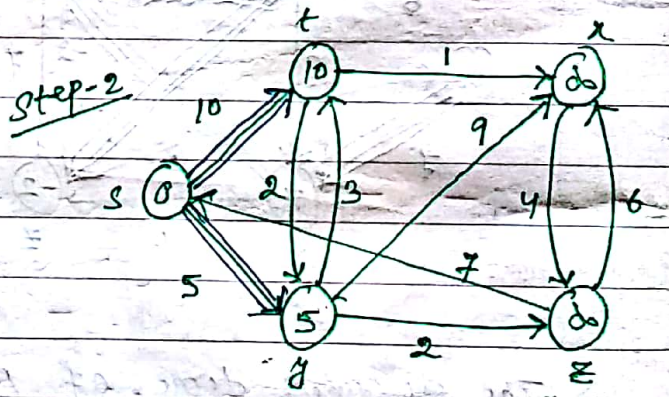
→ We assume that $w(u,v) \geq 0$ for each edge.

```
DIJKSTRA(G, w, s)
1. INITIALIZE-SINGLE-SOURCE(G, s)
2. S ← ∅
3. Q ← V[G]
4. while Q ≠ ∅
5.     do u ← EXTRACT-MIN(Q)        Q(log v)   } θ(V log v)
6.        S ← S ∪ {u}
7.        for each vertex v ∈ Adj[u]
8.            do RELAX(u, v, w)     Q(log v)   } θ(E log v)
```

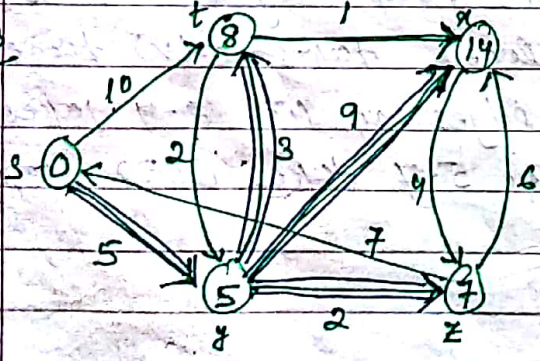
step-1



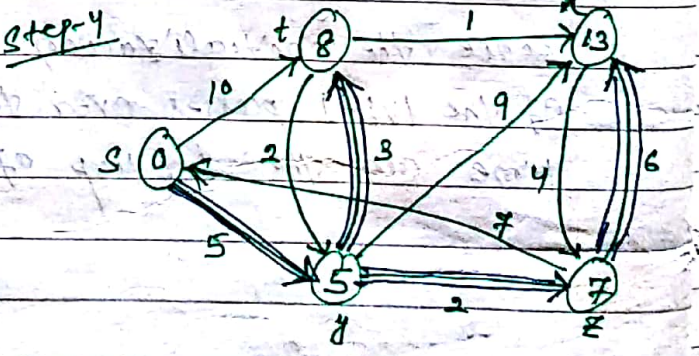
step-2



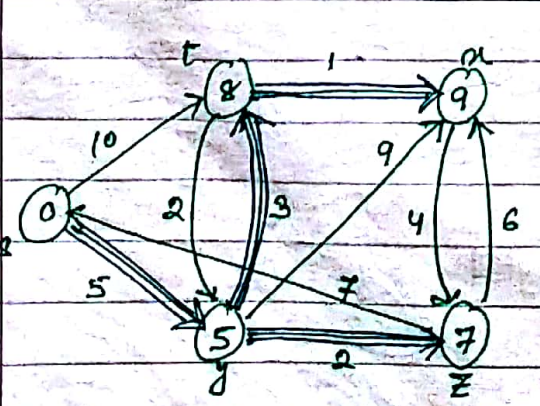
step-3



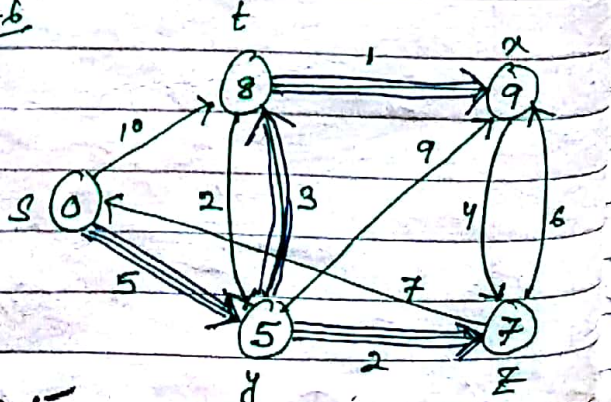
step-4



step-5



step-6



Complexcity:

→ Dijkstra's algorithm maintains the min-priority queue Q by operations such as INSERT operation on line no-3, EXTRACT-MIN operation on line 5 and DECREASE-KEY on line 8 by implicitly calling RELAX.

→ INSERT is invoked once per vertex.

→ EXTRACT-MIN operation also no. of vertex times. Because each vertex $v \in V$ is added to set S exactly once.

→ Each edge is RELAX for $|E|$ times by DECREASE-KEY operations.

→ Like prim's algorithm
EXTRACT-MIN takes $O(\log v)$ times.
EXTRACT-MIN is called for $|V|$ times.
So, total time for EXTRACT-MIN will take $O(V \log v)$

→ RELAX operation will do DECREASE-KEY operation of priority queue which will take $O(\log v)$ times.

→ RELAX operation is executed for $|E|$ times i.e. for each edge.

→ So, total time required for RELAX operation is $O(E \log v)$

→ So total time for Dijkstra algorithm is

~~$O((E+V)\log v)$~~

$O(E \log v) + O(V \log v)$

$= O((E+V) \log v)$

$= O(E \log v)$

$p(s) = p(t) = p(y) = p(x) = p(z) = 1$

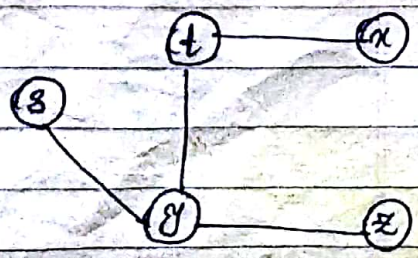
$p(y) = y$

$p(z) = z$

$p(t) = t$

$p(x) = x$

s	t	y	x	z	w
0	10	5	∞	∞	y
0	8	5	14	7	z
0	8	5	13	7	t
0	8	5	11	7	x



ALL PAIRS SHORTEST PATHS

→ All pairs shortest paths problem computes the shortest path from each vertex V to every other u.

→ This algorithm use an adjacency matrix representation.

→ The input is an $n \times n$ matrix W representing the edge weights of an n-vertex directed graph $G = (V, E)$

$W = w_{ij}$, where

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

→ Negative-weight edges are allowed, but we assume for the time being that the input graph contains no negative-weight cycles.

→ An $n \times n$ matrix $D = d_{ij}$ where d_{ij} contains the weight of a shortest path from vertex i to vertex j.

→ Let $\delta(i,j)$ denote the shortest-path weight from vertex i to vertex j, then $d_{ij} = \delta(i,j)$

→ we also compute a predecessor matrix $\Pi = \pi_{ij}$ where π_{ij} is NIL if either $i = j$ or there is no path from i to j and otherwise π_{ij} is some predessor of j on a shortest path from i.

classmate
Date
Page

THE FLOYD-WARSHALL ALGORITHM (Dynamic programming approach)

→ This algorithm is a graph analysis algorithm for finding shortest paths in a weighted, directed graph.

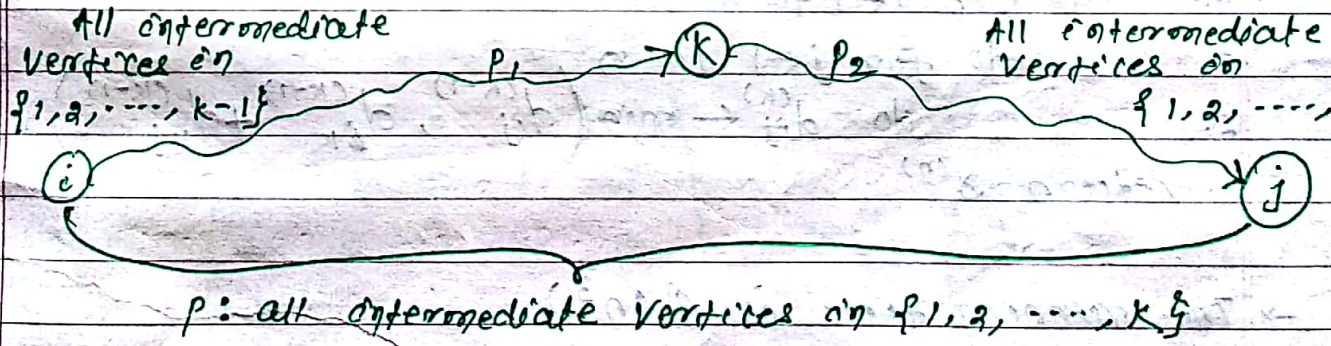
→ Negative-weight edges may be present, but we assume that there are no negative-weight cycles.

→ The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a path $p = \langle v_1, v_2, \ldots v_m \rangle$ is any vertex of p other than v_1 or v_m i.e. any vertex in the set $\{v_2, v_3, \ldots v_{m-1}\}$

→ Let the vertices of G be $V = \{1, 2, \ldots, n\}$ and consider a subset $\{1, 2, \ldots, k\}$ of vertices for some k.

→ For any pair of vertices $i, j \in V$ consider all paths from i to j whose intermediate vertices are from $\{1, 2, \ldots, k\}$

→ The Floyd-Warshall algorithm explores a relationship between path p and shortest paths from i to j with all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$



→ The relationship depends on whether or not k is an intermediate vertex of path p.

*. If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set $\{1, 2, \ldots, k-1\}$. A shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$ is also a shortest path from i to j with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.

*. If k is an intermediate vertex of path p, then we break p down into $i \xrightarrow{p_1} k \xrightarrow{p_2} j$.

Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are $\{1, 2, \ldots, k\}$

When $k=0$, a path from vertex i to vertex j with no intermediate vertex.

Hence, $d_{ij}^{(0)} = w_{ij}$

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

```
FLOYD-WARSHALL(W)
1. n ← rows[W]                                   θ(1)
2. D^(0) ← W                                     θ(n)
3. for k ← 1 to n                                θ(n)
4.     do for i ← 1 to n                         θ(n)
5.         do for j ← 1 to n                     θ(1)
6.             do d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
7. return D^(n)
```

→ The running time is $\theta(n^3)$

Constructing a shortest path:

→ We can compute predecessor matrix Π which is defined to be the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.

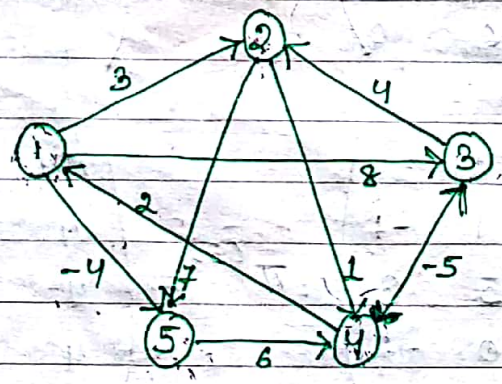
→ The recursive formulation of $\pi_{ij}^{(k)}$. When $k=0$, a shortest path from i to j has no intermediate vertices at all, thus

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

→ For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$, where $k \neq j$

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

→ Example: Apply Floyd-Warshall algorithm for constructing shortest path. Show the matrices $D^{(k)}$ and $\pi^{(k)}$ computed by the Floyd-Warshall algorithm for the graph.



Solution:

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

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \pi^{(0)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \pi^{(1)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \pi^{(2)} = \begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \pi^{(4)} = \begin{pmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \pi^{(5)} = \begin{pmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$