

Floyd's Algorithm

All pairs shortest path

Complexity of Dijkstra's Algorithm

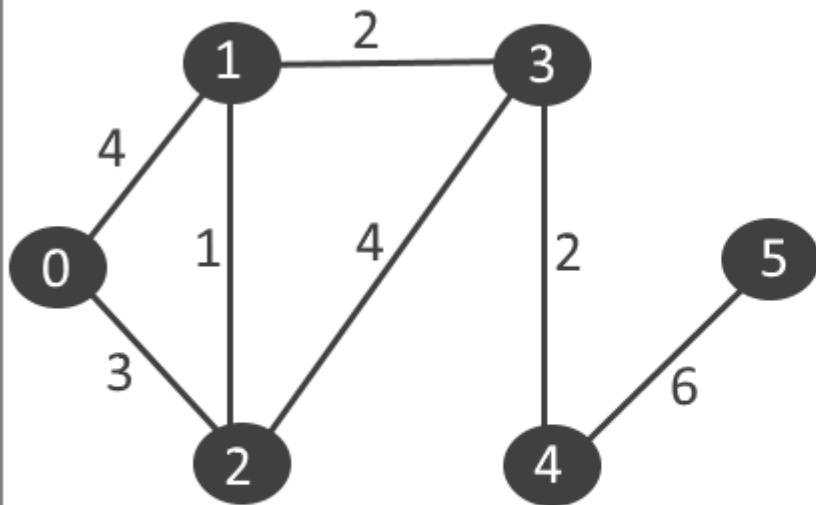
Time Complexity of Dijkstra's Algorithm is $O(v^2)$ but with min-priority queue it drops down to $O(v + E \log v)$

Displaying shortest paths using Dijkstra's Algorithm

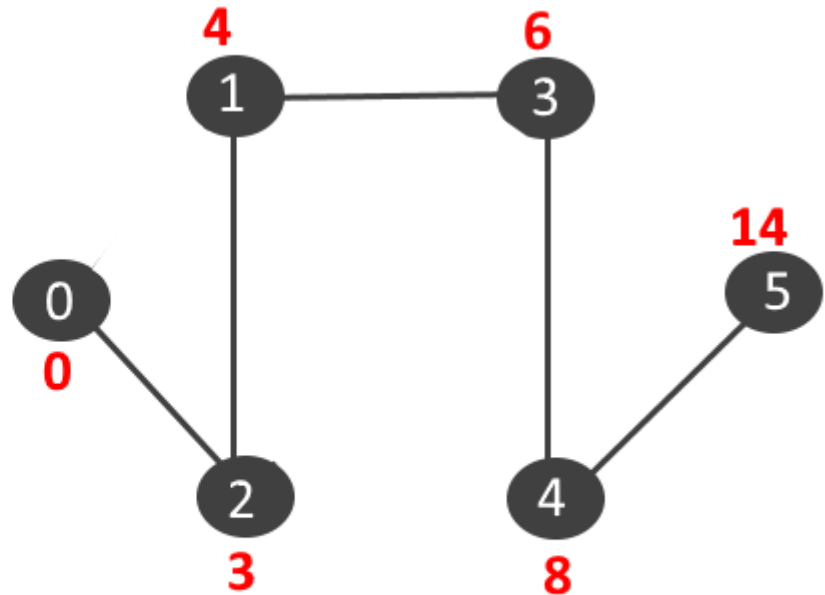
A predecessor vector can be used for this purpose

Vertex	Predecessor p_i
s	s^*
u	x
v	x
x	s
y	x

Shortest Path Tree from Dijkstra's Algorithm



Undirected Graph



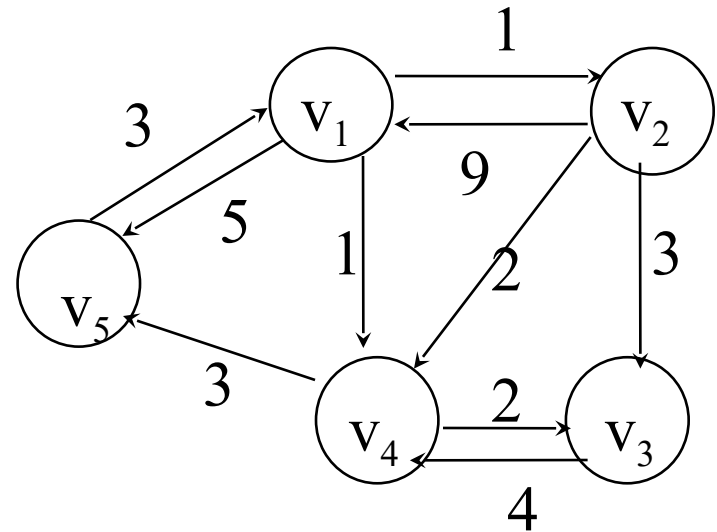
Shortest Path Tree (SPT)

All pairs shortest path

- ***The problem:*** find the shortest path between every pair of vertices of a graph
- ***The graph:*** may contain negative edges but no negative cycles
- ***A representation:*** a weight matrix where
 - $W(i,j)=0$ if $i=j$.
 - $W(i,j)=\infty$ if there is no edge between i and j .
 - $W(i,j)$ = “weight of edge”
- **Note:** we have shown principle of optimality applies to shortest path problems

The weight matrix and the graph

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0



The subproblems

- How can we define the shortest distance $d_{i,j}$ in terms of “smaller” problems?
- One way is to restrict the paths to only include vertices from a restricted subset.
- Initially, the subset is empty.
- Then, it is incrementally increased until it includes all the vertices.

The subproblems

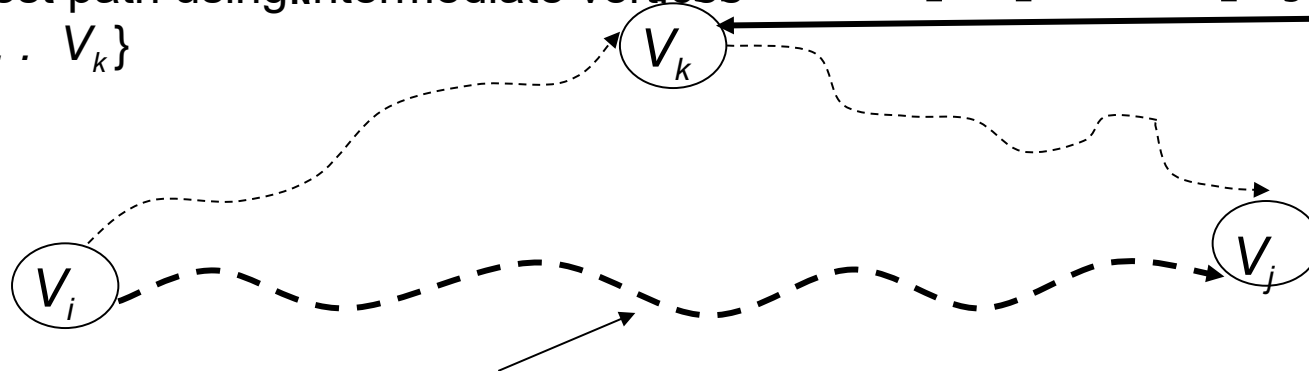
- Let $D^{(k)}[i,j]$ =weight of a shortest path from v_i to v_j using only vertices from $\{v_1, v_2, \dots, v_k\}$ as intermediate vertices in the path
 - $D^{(0)}=W$
 - $D^{(n)}=D$ which is the goal matrix
- How do we compute $D^{(k)}$ from $D^{(k-1)}$?

The Recursive Definition:

Case 1: A shortest path from v_i to v_j restricted to using only vertices from $\{v_1, v_2, \dots, v_k\}$ as intermediate vertices does not use v_k . Then $D^{(k)}[i, j] = D^{(k-1)}[i, j]$.

Case 2: A shortest path from v_i to v_j restricted to using only vertices from $\{v_1, v_2, \dots, v_k\}$ as intermediate vertices does use v_k . Then $D^{(k)}[i, j] = D^{(k-1)}[i, k] + D^{(k-1)}[k, j]$.

Shortest path using intermediate vertices $\{V_1, \dots, V_k\}$



Shortest Path using intermediate vertices $\{V_1, \dots, V_{k-1}\}$

The recursive definition

- Since

$$D^{(k)}[i,j] = D^{(k-1)}[i,j] \text{ or}$$

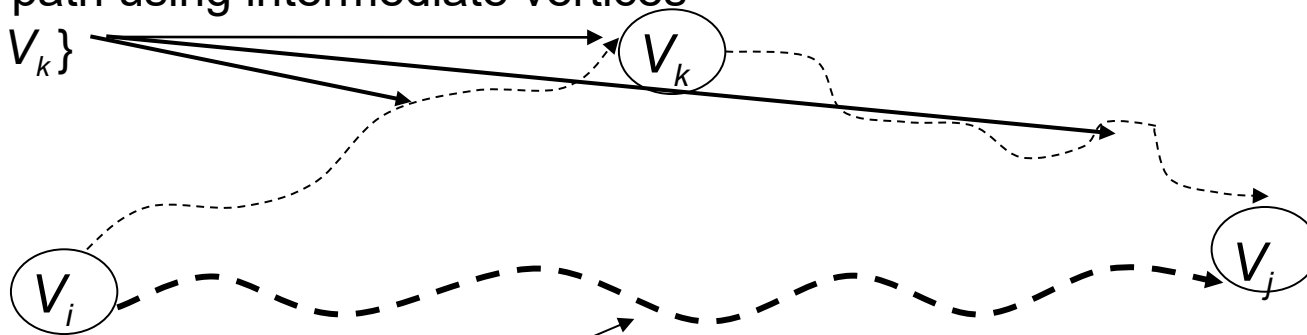
$$D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j].$$

We conclude:

$$D^{(k)}[i,j] = \min\{ D^{(k-1)}[i,j], D^{(k-1)}[i,k] + D^{(k-1)}[k,j] \}.$$

Shortest path using intermediate vertices

$\{V_1, \dots, V_k\}$



Shortest Path using intermediate vertices $\{V_1, \dots, V_{k-1}\}$

The pointer array P

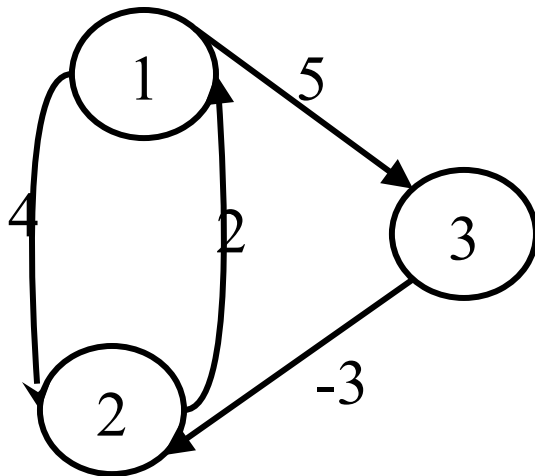
- Used to enable finding a shortest path
- Initially the array contains 0
- Each time that a shorter path from i to j is found the k that provided the minimum is saved (highest index node on the path from i to j)
- To print the intermediate nodes on the shortest path a recursive procedure that print the shortest paths from i and k , and from k to j can be used

Floyd's Algorithm Using $n+1$ D matrices

Floyd//Computes shortest distance between all pairs of
//nodes, and saves P to enable finding shortest paths

1. $D^0 \leftarrow W$ // initialize D array to W []
2. $P \leftarrow 0$ // initialize P array to [0]
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. if ($D^{k-1}[i, j] > D^{k-1}[i, k] + D^{k-1}[k, j]$)
7. then $D^k[i, j] \leftarrow D^{k-1}[i, k] + D^{k-1}[k,$
 $j]$
8. $P[i, j] \leftarrow k;$
9. else $D^k[i, j] \leftarrow D^{k-1}[i, j]$

Example

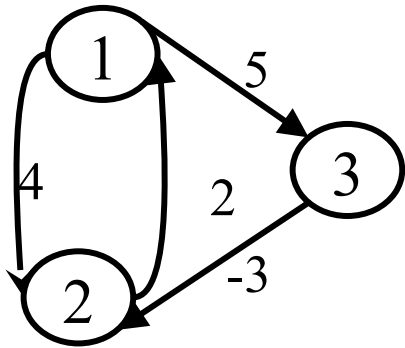


$$W = D^0 =$$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

$$P =$$

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0



$$D^0 =$$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

$k = 1$

Vertex 1 can
be intermediate
node

$$D^1 =$$

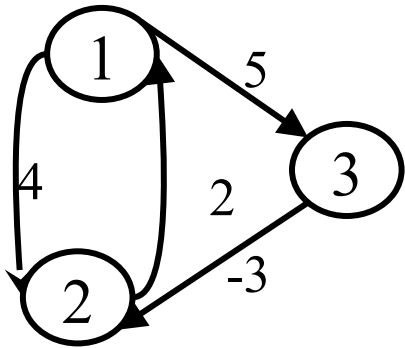
	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

$$\begin{aligned}
 D^1[2,3] &= \min(D^0[2,3], D^0[2,1]+D^0[1,3]) \\
 &= \min(\infty, 7) \\
 &= 7
 \end{aligned}$$

$$P =$$

	1	2	3
1	0	0	0
2	0	0	1
3	0	0	0

$$\begin{aligned}
 D^1[3,2] &= \min(D^0[3,2], D^0[3,1]+D^0[1,2]) \\
 &= \min(-3, \infty) \\
 &= -3
 \end{aligned}$$



$$D^1 =$$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

$k = 2$

Vertices 1, 2
can be
intermediate

$$D^2 =$$

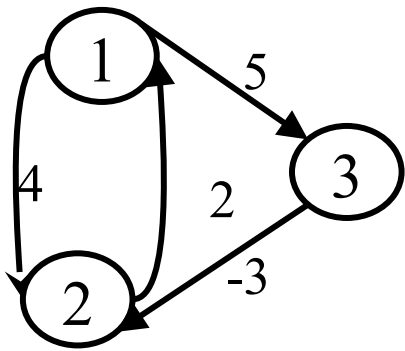
	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$$\begin{aligned} D^2[1,3] &= \min(D^1[1,3], D^1[1,2]+D^1[2,3]) \\ &= \min(5, 4+7) \\ &= 5 \end{aligned}$$

$$P =$$

	1	2	3
1	0	0	0
2	0	0	1
3	2	0	0

$$\begin{aligned} D^2[3,1] &= \min(D^1[3,1], D^1[3,2]+D^1[2,1]) \\ &= \min(\infty, -3+2) \\ &= -1 \end{aligned}$$



$$D^2 =$$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

$k = 3$

Vertices 1, 2, 3
can be
intermediate

$$D^3 =$$

	1	2	3
1	0	2	5
2	2	0	7
3	-1	-3	0

$$\begin{aligned} D^3[1,2] &= \min(D^2[1,2], D^2[1,3] + D^2[3,2]) \\ &= \min(4, 5 + (-3)) \\ &= 2 \end{aligned}$$

$$P =$$

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

$$\begin{aligned} D^3[2,1] &= \min(D^2[2,1], D^2[2,3] + D^2[3,1]) \\ &= \min(2, 7 + (-1)) \\ &= 2 \end{aligned}$$

Floyd's Algorithm: Using 2 D matrices

Floyd

1. $D \leftarrow W$ // initialize D array to $W[]$
2. $P \leftarrow 0$ // initialize P array to $[0]$
3. for $k \leftarrow 1$ to n
 // Computing D' from D
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. if ($D[i, j] > D[i, k] + D[k, j]$)
7. then $D'[i, j] \leftarrow D[i, k] + D[k, j]$
8. $P[i, j] \leftarrow k$;
9. else $D'[i, j] \leftarrow D[i, j]$
10. Move D' to D .

Can we use only one D matrix?

- $D[i,j]$ depends only on elements in the k th column and row of the distance matrix.
- We will show that the k th row and the k th column of the distance matrix are unchanged when D^k is computed
- This means D can be calculated *in-place*

The main diagonal values

- Before we show that k th row and column of D remain unchanged we show that the main diagonal remains 0
- $$\begin{aligned} D^{(k)}[j,j] &= \min\{ D^{(k-1)}[j,j] , \quad D^{(k-1)}[j,k] + D^{(k-1)}[k,j] \} \\ &= \min\{ 0, \quad D^{(k-1)}[j,k] + D^{(k-1)}[k,j] \} \\ &= 0 \end{aligned}$$
- Based on which assumption?

Floyd's Algorithm using a single D

Floyd

1. $D \leftarrow W$ // initialize D array to $W[]$
2. $P \leftarrow 0$ // initialize P array to $[0]$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. if ($D[i, j] > D[i, k] + D[k, j]$)
7. then $D[i, j] \leftarrow D[i, k] + D[k, j]$
8. $P[i, j] \leftarrow k$;

Printing intermediate nodes on shortest path from q to r

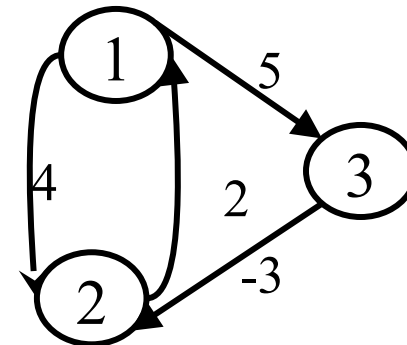
```

path(index q, r)
  if (P[ q, r ]!=0)
    path(q, P[q, r])
    println( "v"+ P[q, r])
    path(P[q, r], r)
  return;
//no intermediate nodes
else return
    
```

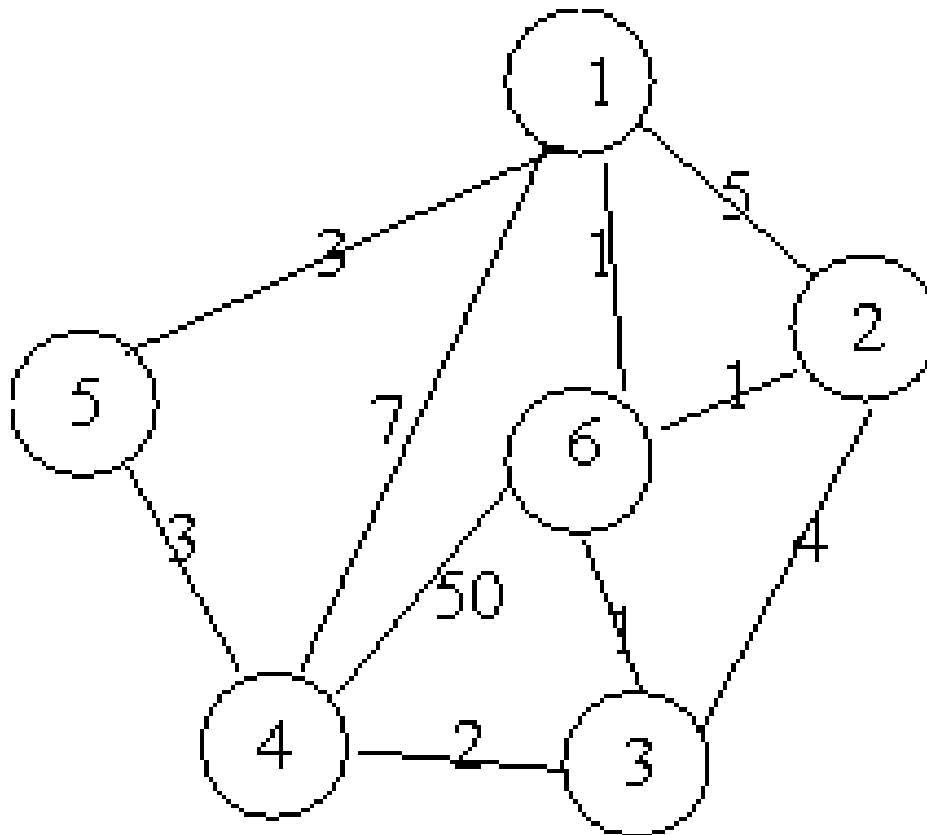
Before calling path check $D[q, r] < \infty$, and
print node q, after the call to
path print node r

P =

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0



Example



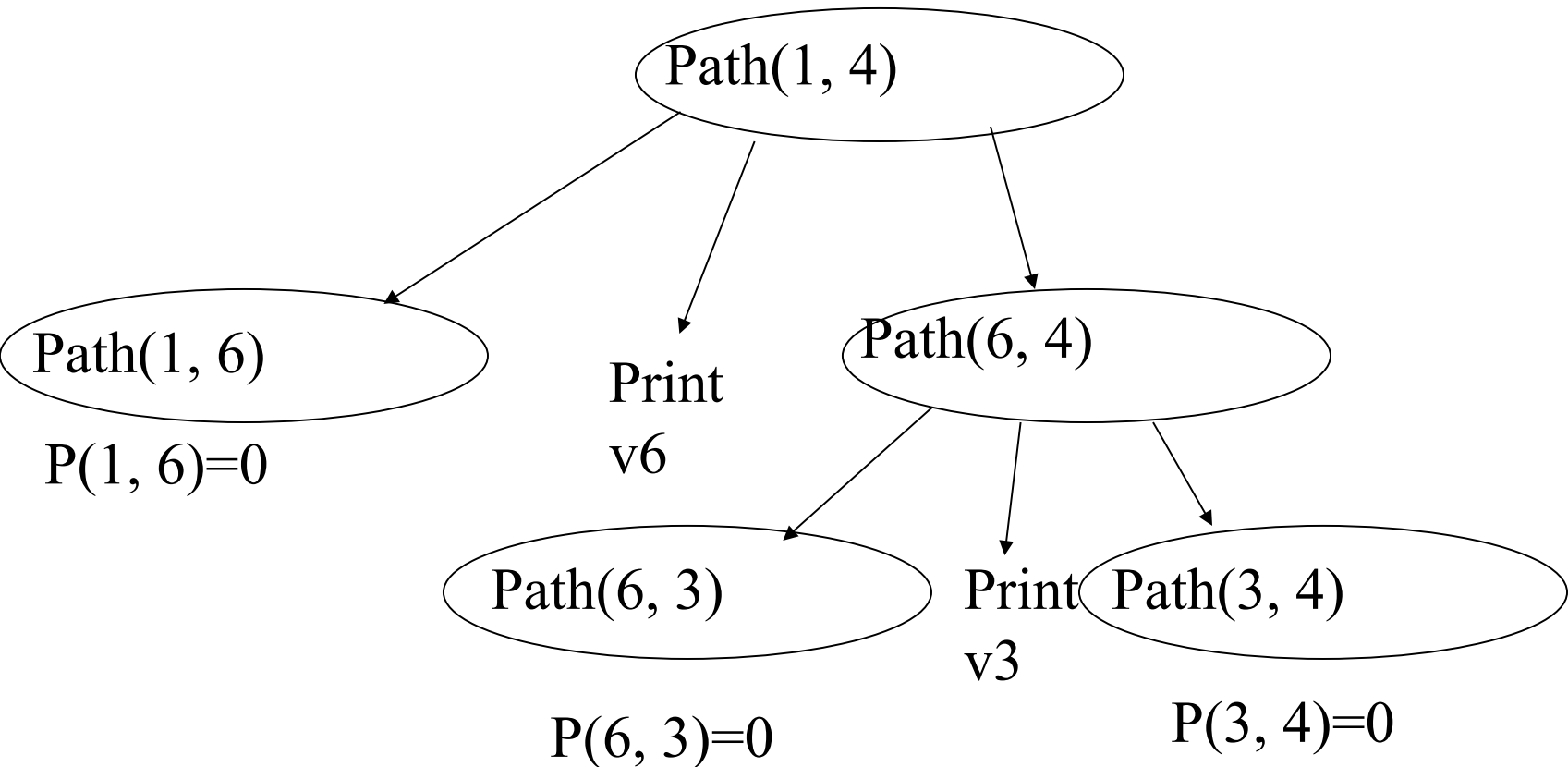
The final distance matrix and P

$$D^6 =$$

	1	2	3	4	5	6
1	0	2(6)	2(6)	4(6)	3	1
2	2(6)	0	2(6)	4(6)	5(6)	1
3	2(6)	2(6)	0	2	5(4)	1
4	4(6)	4(6)	2	0	3	3(3)
5	3	5(6)	5(4)	3	0	4(1)
6	1	1	1	3(3)	4(1)	0

The values in parenthesis are the non zero P values.

The call tree for Path(1, 4)



The intermediate nodes on the shortest path from 1 to 4 are v6, v3.
The shortest path is v1, v6, v3, v4.