Addis Ababa University

School of Graduate Studies

Faculty of Computer and Mathematical Sciences

Department of Mathematics

Project

On

Ranked k-shortest paths in network

By: - Tsegay Brhane

Advisor: - Berhanu Guta (PhD)

A Project submitted to the Office of Graduate Programs of Addis Ababa University in Partial fulfillment of the requirements for the Degree of Master of Science in Mathematics

January, 2011

AA, Ethiopia

Addis Ababa University
school of Graduate Studies
Faculty of Computer and Mathematical Science
Department of Mathematics

Approved by the Board of Examiners:

_____	_____
Department Head	**Signature**
_____	_____
Examiner	**Signature**
_____	_____
Examiner	**Signature**

Declaration

“I declare that this project has been composed by Tsegay Brhane and that no part of the project has formed the basis for the award of any degree, diploma, associate ship, fellowship or any other similar title to me.

_____”

Author’s Signature

Permission

“This is to certify that this project is compiled /a recorded of the research work done / by Tsegay Brhane in the Department of mathematics, Addis Ababa University, under my supervision.

_____”

Advisor’s Signature

ABSTRACT

The complete set of this paper focuses on k-shortest path problem and algorithms which solve k-shortest path in a graph $G=(V,E)$ where V is set of n nodes and E is set of m arcs. and I have discussed how to compute the first, second, third, - - - , k^{th} shortest path (in order) from the source node to every other nodes by using labeling algorithm which is label setting and label correcting algorithm and path removing algorithm in a given network.

ACKNOWLEDGEMENT

First and foremost, it is my pleasure to express my heart-felt appreciation and special gratitude to my advisor Dr. Berhanu Guta for his patience in repeatedly reading the draft manuscript of this project and for making constructive comments and suggestions from which I have immeasurably benefited in sharpening my understanding, predominantly, on the area I study.

Secondly, I would like to extend grateful acknowledgement to my family who support me all the time.

Last but not least, I am also indebted for my friends and to all those who helped me one way or another for the accomplishment of my study.

Tsegay Brhane

tsegay.ab@gmail.com

A.A.U.

2011

Table of content

Title page

Glossary of symbols and notations

d_j → distance value of node j.

C_{ij} → length (cost) of arc (i, j).

$d(j)$ → set of distance value of node j.

$d_j(r)$ → r^{th} distance value of node j.

p_j^r → r^{th} shortest path from node 1 to node j.

$l(p)$ → length of path p.

Introduction

The shortest path problem is one of the most fundamental network optimization problems; this problem comes up in practice and arises as sub problem in many network optimization problems. Algorithm for this problem have been studied for long time, however advances in the method and theory of shortest path algorithm are still being made. k-shortest path problem is advanced and long studied generalization of the shortest path problem in which the objective of k-shortest path problem is not only to find the first shortest path as shortest path problem but also to find the second , third ,- - - , k^{th} shortest path (in order) from the source node to every other nodes in the network. The first work on k-shortest paths appeared in 1959, more or less at the same time with the first papers on the shortest path problem ,the resulting problem when $k = 1$, have been published. Since then shortest path problem has merited much more attention from the researcher than k-shortest path problem because k-shortest path problem manipulate great amount of data. However with the development of computer and data structures there has been an increasing interest from the researcher on k-shortest path problem, as larger problem resulting from real world application.

This paper contains two chapters. The first Chapter devoted some basic ideas of network system, shortest path problem and algorithm solving shortest path problem such as Dijakstra's and bellman ford algorithm. Moreover the shortest path problems with their solution methodology are discussed. And the second Chapter gives a briefly description of the k shortest path and algorithm solving k shortest paths such as labeling algorithm (label setting and label correcting algorithm) and path removing algorithm with their examples.

CHAPTER ONE

1 PRELIMINARY CONCEPT

1.1 Basic definitions

In this section we introduce some of the basic definitions relating to network, graph and related notions.

Network:

A triple (V,E,C) is said to be network if $G=(V,E)$ is directed graph without directed cycle and passes exactly one source and at least one sink.

The elements of V are called nodes or vertices.

The elements of E are called arc or edge.

Directed graph:

A graph $G=(V,E)$ consists of a set V of nodes and a set E of pairs of distinct nodes from V called arcs. The numbers of nodes and arcs are denoted by N and A, and it is assume throughout that $1 \leq \mathrm{N} < \infty$ and $0 \leq \mathrm{A} < \infty$. An arc (i,j) is viewed as an ordered pair, and is to be distinguished from the pair (j,i). If (i,j) is an arc, we say that (i,j) is outgoing from node i and incoming to node j ; we also say that j is an outward neighbor of i and that i is an inward neighbor of j. We say that arc (i,j) is incident to i and to j, and that i is the start node and j is the end node of the arc. We also say that i and j are the end nodes of arc (i,j).

Undirected graph: where an arc is associated with a pair of nodes regardless of order.

Source (initial) node: is anode which has only outgoing arrows or edges.

Terminal (sink) node: is anode which has only incoming arrows or edges.

Intermediate node: is anode which has both incoming and outgoing arrows (edges).

Path: a path p in a directed graph is a sequence on nodes $(v_1,v_2,...,v_k)$ with $k \geq 2$ a corresponding sequence of $k-1$ arcs such that the i^{th} arc in the sequence is either

(v_i, v_{i+1}) (in such case it is called a forward arc of path) or (v_{i+1}, v_i) (in which case it is called a back ward arc of the path) nodes v_1 and v_k are called the starting node (origin) and the end node (destination) of p respectively.

Forward path: a path is said to be a forward path if all of its arcs are forward arc.

Backward path: a path is said to be a backward path if all of its arcs are backward arc

Cycle: is a path for which the starting and end nodes are the same.

Cyclic network: is a network consisting of directed cycle.

A cyclic network: is a network without any directed cycle.

Connected network: a network is said to be connected if for each pair of node i and j there is a path starting at i and ending at j.

Tree: - is a connected partial sub network that contain no cycle

Strongly connected network: a network is said to be strongly connected if for each pair of node i and j there is a forward path starting at i and ending at j.

Sub graph: we say that a graph $G' = (V', E')$ is a sub path of a graph $G = (V, E)$ if $V' \subset V$ and $E' \subset E$.

Weighted graph: is a graph in which each edge has an associated numerical value.

Edge weight may represent distance, cost, time etc.

Complexity of an algorithm; - is the number of computational steps taken by an algorithm to solve a particular type of model (if n, represents the size of our model we wish to know the maximum number of operations needed to solve it as a function $f(n)$ then we can be shore that for any instance of a model of size n the computational steps will not be greater than $f(n)$).

1.2 Shortest path problem

The shortest path problem is a classical and important combinatorial problem that arises in many contexts .we are given a weighed graph $G=(V,E)$ with a node set V, edge set E and weight set C.each arc $(i,j)\in E$ has a cost or "length" C_{ij} associated with it. We are also given a starting node $S\in V$ and terminal node $t\in V$.The length of a path $P=(V_1,V_2,...,V_{k-1},V_k)$ is the sum of the arcs length in the path (P) and it will be denoted by $l(P)$ so $l(P)=\sum_{(i,j)\in p} C_{ij}$.

A path is said to be shortest path if it has minimum length over all paths with the same origin and destination node. i.e. P^* said to be a shortest path over all paths with the same source and destination node if $P^*\in R$ such that $l(P^*)\le l(q)$ $\forall q\in R$ where R is set of paths with the same source and destination node .the length of the shortest path is also called the shortest distance.

Types of shortest path problem

Single –source (one-to-all) shortest path

Find shortest path from source node S to every other nodes.

Single –destination (all-to-one) shortest path

Find shortest path from every node to destination node t in the network.

Single –pair (one-to-one) shortest path

Find shortest path from specific node u to specific node v.

All-pair shortest path

Find shortest path between every pair of nodes.

Note

Since there is no algorithm is known for computing a single pair shortest path better than solving single source shortest path problem and also all pair shortest path con be solving by running single source shortest path $|V|$ times .we will focus on single source shortest path problem.

Shortest path property

Property 1: a sub path of a shortest path is itself a shortest path.

Property 2: there is a tree of shortest path from a start vertex to all other vertices.

Lemma: shortest path can't contain cycles.

1.3 Algorithms solving shortest paths

In this section we develop abroad class of shortest path algorithms for single source (one –to-all) shortest path problem. Those algorithms maintain and adjust a vector $(d_1, d_2, ..., d_N)$ where each d_j called the distance value of node j, is either a scalar or ∞ .those broad class of shortest path algorithms classified in to two broad categories based on how they select the node to be removed from the candidate list of nodes. Those are

- Label setting algorithm
- Label correcting algorithm

A, Label setting algorithm

In this algorithm the node i removed from the candidate list is anode with minimum distance value under the assumption that all arc length are non negative $(C_{ij} \geq 0)$.this algorithm has a remarkable property: each node will enter in to candidate list at most once, as we will show shortly its distance value has its permanent or final value at the first time it removed from the candidate list .the most time consuming part of this algorithm is calculating the minimum distance value associated to each nodes of candidate list.

B, Label correcting algorithm

In this section the choice of a node i to remove from the candidate list is the same as label setting algorithm however anode may enter to candidate list multiple times and distance value of anode become permanent if the candidate list of nodes is empty. But if we use FIFO queue of label the choice of node i removed from the candidate list is if anode i is the first element of the candidate list.

Both group of algorithm employ the labeling method in computing one-to-all shortest path problem.

Those two groups of algorithm differ in the way in which they update the estimate of the shortest path distance associate with each node of each iteration and the way in which they converge of the final optimal one-to-all shortest path.

In label setting algorithm the final optimal shortest path distance from the source node to the destination node determined once the destination node remove from the candidate list .while in label correcting algorithm the final optimal shortest path distance from the source node to destination node is determined when the candidate list of node is empty.

Whereas Dijakstra's algorithm is representative of label setting algorithm and Bellman ford algorithm is representative of label correcting algorithm.

1.3.1 Dijakstra's algorithm

Dijakstra's algorithm is one of the most important algorithms from the group of label setting algorithm and it solves the single source shortest path problem (find the shortest path from the source node S to all other nodes) on weighted directed graph $G = (V, E)$ for the case in which all edge weights are non negative.

If there is an edges with negative weight the actual shortest path cannot obtained by using Dijakstra's algorithm because Dijakstra's algorithm adds vertices by increasing its distance value but if there is a negative weighted edge the distance value of a vertices cannot increase its value.

Algorithm description step by step

Step 1, Initialization

- Assign the zero distance value to node S, and label it as permanent.

  I.e. [The state of node S is $(0, p)$]

- Assign to every other node a distance value of ∞ and label them as temporary.

  I.e. [The state of every other node is (∞, t)]

- Designate the node S as the current node.
- Candidate list of node = { S }

Step 2, Distance value update & current node designation update

Let i be the index of the current node

- Find the set J of nodes with temporary label that can be reached from the current node i by a link (i, j).update the distance value of these nodes. for each $j \in J$ the distance value d_j of node j is update as follows

  $$\text{New } d_j = \min\{d_j, d_i + C_{ij}\}$$

  Where C_{ij} is the cost of link (i, j) as given in the network problem

- Candidate list of node (L)={S}u{J}
- Determine a node j that has the smallest distance value d_j among all node $j \in J$ .find j^* such that $\min_{j \in L} d_j = d_{j^*}$
- Change the label of node j^* to permanent and designate this node as the current node. and remove j^* from the candidate list of node

Step 3, termination criteria

- If all nodes that can be reached from node S have been permanently labeled then stop .we are done
- If we can not reach any temporary labeled node from the current node then all the temporary labels become permanent .we are done .otherwise go to step 2

Algorithm

Step 1: Initialization

begin

$d_s = 0$ and $pred(s) = 0$;

$d_j = \infty$ for each node $j \neq s$;

LIST = $\{ s \}$;

Step 2: Main step

While LIST $\neq \varnothing$ **do**

begin

Remove the node which has minimum associated distance value from LIST;

for each arc $(i, j) \in A(i)$ **do**

if $d_j > d_i + c_{ij}$ **then**

begin

$d_j = d_i + c_{ij}$; $pred(j) = i$;

if $j \notin$ LIST **then** append j to the end of LIST;

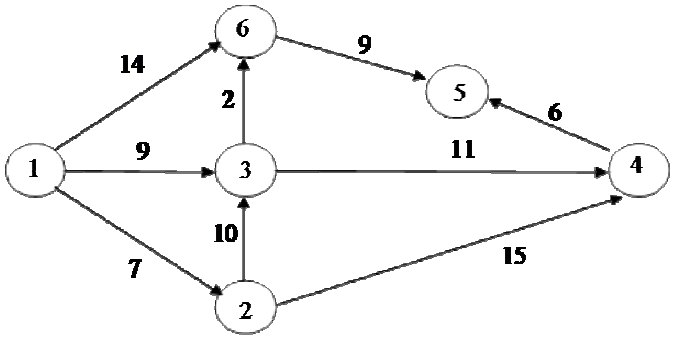
end;

end;

end;

Example 1

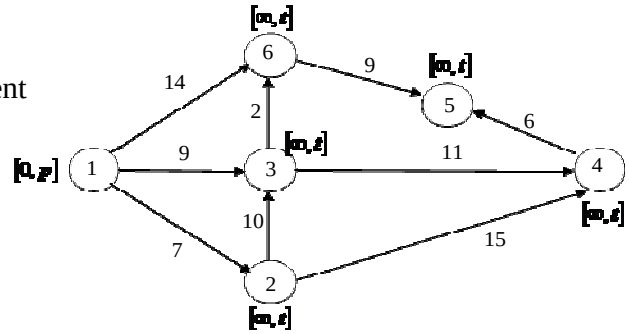
We want to find the shortest path from node 1 to all other nodes using Dijakstra's algorithm



Solution

Step 1 Initialization

- Node 1 is designated as the current Node
- The state of node 1 is $(0, p)$.
- Evert other node has state (∞, t)



Step 2 Distance value update & current node designation

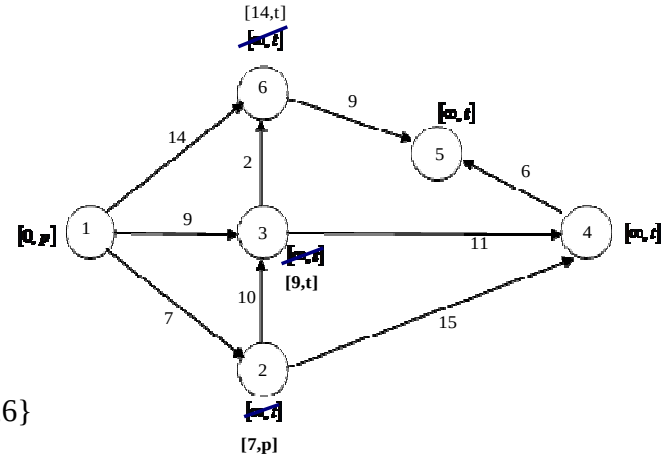
- Node 2,3and 6 can be reached from the current node 1.
- Update distance value for those nodes

$$d_2 = \min\{\infty, 0+7\} = 7$$

$$d_3 = \min\{\infty, 0+9\} = 9$$

$$d_6 = \min\{\infty, 0+14\} = 14$$

- Candidate list of node (L) = {2,3,6}



- Now among all elements of L node 2 has the smallest distance value
- The status label of node 2 changes to permanent. So its state is $[7, p]$ while the status of node 3 and 6 remains temporary.
- Node 2 become the current node and remove it from L.

Step 3, Termination criteria

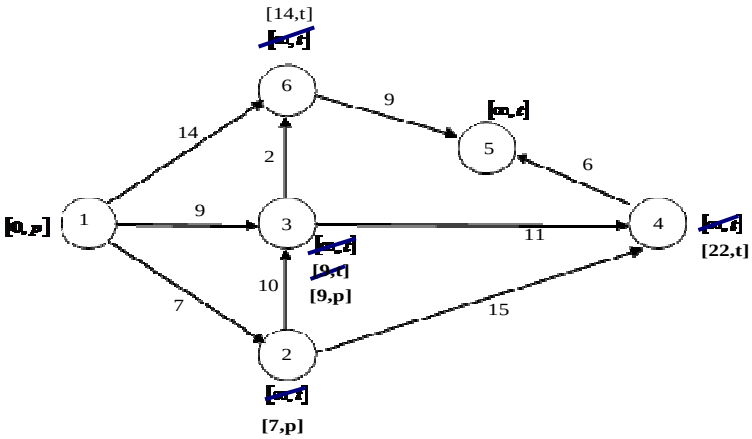
- We are not done .because all nodes have not been reached from the source node 1. So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 3and 4 can be reached from the current node 2.
- Update distance value for those nodes

$$d_3 = \min\{9, 7+10\} = 9$$

$$d_4 = \min\{\infty, 7+15\} = 22$$



- Candidate list of node(L) = {3,6,4}
- Now among all elements of L node 3 has the smallest distance value
- The status label of node 3 changes to permanent. So its state is $[9, p]$ while the status of node 4 remains temporary.
- Node 3 becomes the current node and remove it from L.

Step 3, Termination criteria

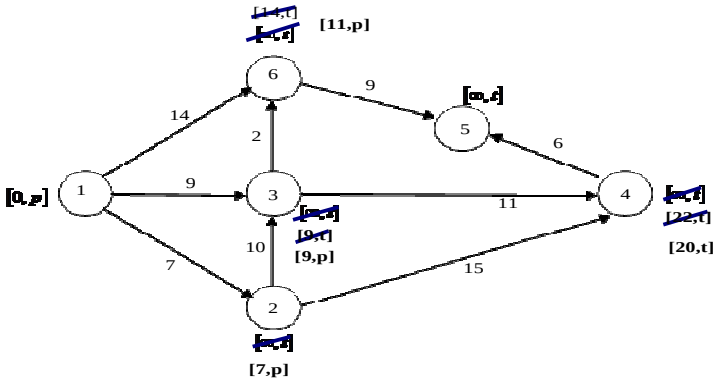
- We are not done .because all nodes have not been reached from the source node 1. So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 6 and 4 can be reached from the current node 3.
- Update distance value for those nodes

$$d_4 = \min\{22,9+11\} = 20$$

$$d_6 = \min\{14,9+2\} = 11$$



- Candidate list of node(L) = {6,4}
- Now among all elements of L node 6 has the smallest distance value
- The status label of node 6 changes to permanent. So its state is $[11, p]$ while the status of node 4 remains temporary.
- Node 6 becomes the current node and remove it from L.

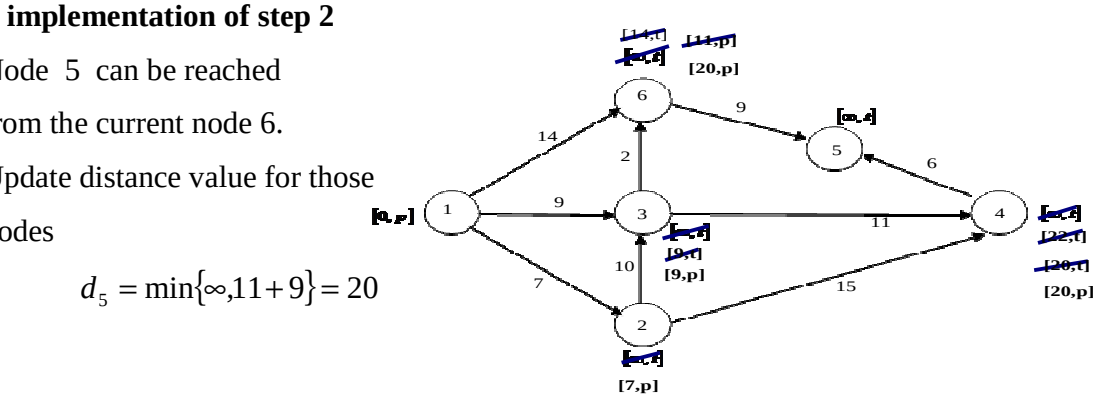
Step 3, Termination criteria

- We are not done .because all nodes have not been reached from the source node 1. So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 5 can be reached from the current node 6.
- Update distance value for those nodes

$$d_5 = \min\{\infty,11+9\} = 20$$



- Now node 5 is the only candidate so its status changes to permanent.
- Node 5 becomes the current node.
- From node 5 we cannot reach any other node .hence node 4 gets permanent labeled and we are done

1.3.2 Bellman ford algorithm

The bellman ford algorithm is an efficient algorithm to solve single source shortest path problem in the more general case in which edge weight can be negative. Given a weighted graph $G=(V,E)$.with source node S and weight function $W:E\to R$.the bellman ford algorithm returns a Boolean value indicating whether or not there is a negative weight cycle reachable from the source node.

If there is such cycle the algorithm indicates that no solution exists if there is no such cycle the algorithm produces paths and their weight.

Algorithm description step by step

Step 1, Initialization

- Assign the zero distance value to node S, and label it as permanent.

  I.e. [The state of node S is $(0,p)$]

- Assign to every other node a distance value of ∞ and label them as temporary.

  I.e. [The state of every other node is (∞,t)]

- Designate the node S as the current node.
- Candidate list of node = { S }

Step 2, Distance value update & current node designation update

Let i be the index of the current node

- find the set J of nodes with temporary label that can be reached from the current node i by a link (i,j).update the distance value of these nodes. for each $j\in J$ the distance value d_j of node j is update as follows

  $$\text{New } d_j=\min\{d_j,d_i+C_{ij}\}$$

  Where C_{ij} is the cost of link (i,j) as given in the network problem

- Candidate list of node (L)={S}u{J}
- Determine a node j that has the smallest distance value d_j among all node $j\in J$ .find j^* such that $\min\limits_{j\in L} d_j = d_{j^*}$

- Designate the node j^* as the current node. and remove j^* from the candidate list of node

Step 3, termination criteria

- If candidate list of node becomes empty then all the temporary labels become permanent. then Stop because we are done otherwise go to step 2

Remark

Bellman ford algorithm is the same as Dijakstra's algorithm while in bellman ford algorithm one node can enter in to list of nodes more than once .but if we use FIFO queue of label its algorithm is stated us follow.

Algorithm;

Step 1: Initialization

begin

$d_s = 0$ and $pred(s) = 0$;

$d_j = \infty$ for each node $j \neq s$;

LIST = $\{ s \}$;

Step 2: Main step

While LIST $\neq \varnothing$ **do**

begin

Remove the first element from LIST;

for each arc$(i, j) \in A(i)$ **do**

if $d_j > d_i + c_{ij}$ **then**

begin

$d_j = d_i + c_{ij}$; $pred(j) = i$;

if $j \notin$ LIST **then** append j to the end of LIST;

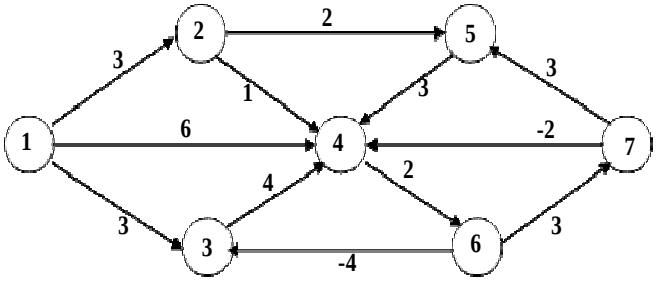
end;

end;

end;

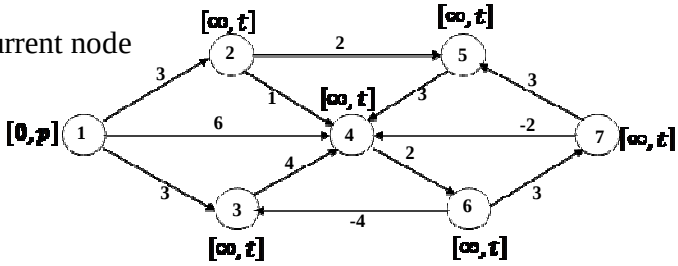
Example 2

We want to find the shortest path from the source node to all other nodes using bellman ford algorithm



Initialization step one

- Node one is designated as the current node
- The state of node one is $[o, p]$
- Ever other node has state $[\infty, t]$
- List of node = $\{\ 1\ \}$



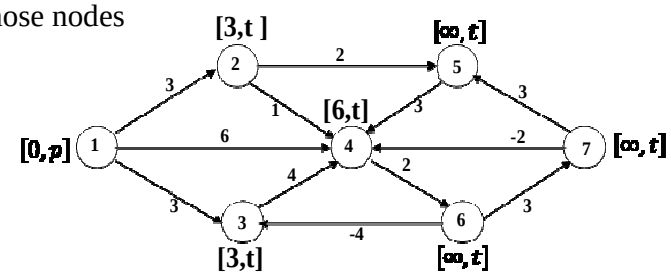
Step 2

- Node 2, 3 and 4 can be reached from the current node 1.
- Update the distance value for those nodes

$d_2 = \min\{\infty, 0+3\} = 3$

$d_3 = \min\{\infty, 0+3\} = 3$

$d_4 = \min\{\infty, 0+6\} = 6$



- List of node (L) = {2, 3, 4}
- Now among all elements of L node 2 and 3 have the smallest distance value
- Node 2 become the current node and removes it from list of nodes

Step 3

- We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

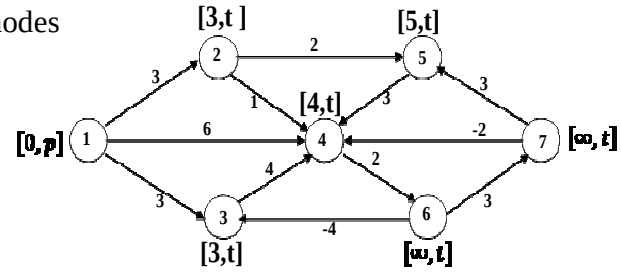
  So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 5 and 4 can be reached from the current node 2
- Update the distance value for those nodes

$d_5 = \min\{\infty, 3+2\} = 5$

$d_4 = \min\{\infty, 3+1\} = 4$



- List of nodes = $\{3, 4, 5\}$
- Now among all element of list of nodes node 3 has the smallest distance value
- Node 3 becomes the current node and remove it from the list of nodes

Step 3

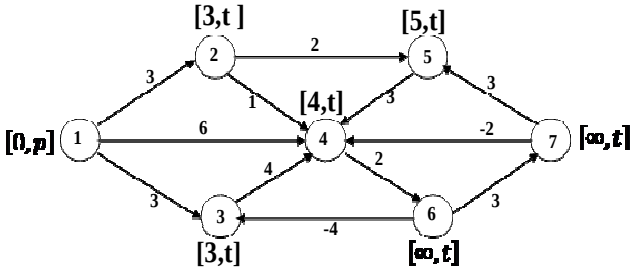
- We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

  So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 4 can be reached from the current node 3
- Update the distance value for those nodes

$d_4 = \min\{4, 3+4\} = 4$



- List of nodes = $\{\ 4\ ,\ 5\ \}$
- Now among all element of list of nodes node 4 has the smallest distance value.
- Node 4 becomes the current node and remove it from the list of nodes.

Step 3

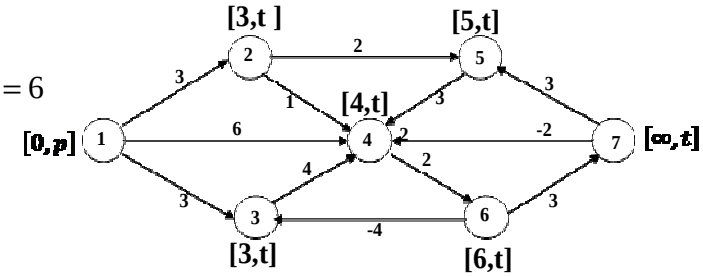
- We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

  So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 6 can be reached from the current node 4.
- Update the distance value of node 6

$$d_6 = \min\{\infty\ ,\ 2+4\} = 6$$



- List of nodes = $\{\ 5\ ,\ 6\ \}$
- Now among all element of list of nodes node 5 has the smallest distance value.
- Node 5 becomes the current node and remove it from the list of nodes.

Step 3

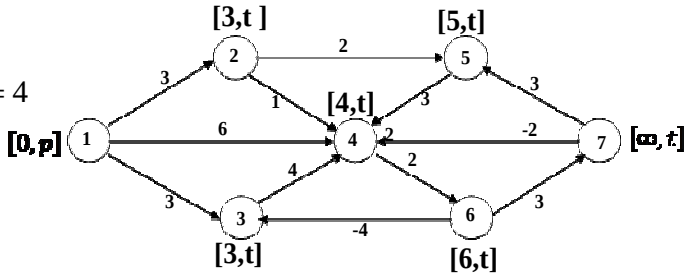
- We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

  So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 4 can be reached from the current node 5.
- Update the distance value of node 4

$$d_4 = \min\{4\ ,\ 5+3\} = 4$$



- List of nodes = $\{\ 6\ ,\ 4\ \}$

- ❖ Now among all element of list of nodes node 4 has the smallest distance value.
- ❖ Node 6 becomes the current node because we are work with node 4 as a current node and remove node 4 and 6 from the list of nodes.

Step 3

- ➢ We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

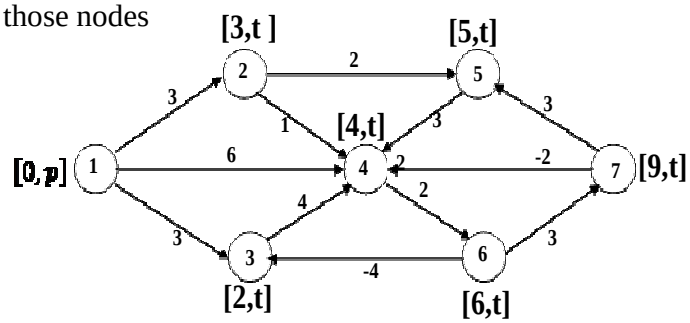
  So we perform other iteration (back to step 2)

Another implementation of step 2

- ❖ Node 3 and 7 can be reached from the current node 6.
- ❖ Update the distance value of those nodes

$$d_3 = \min\{3, 6-4\} = 2$$

$$d_7 = \min\{3, 6+3\} = 9$$



- ❖ List of nodes = { 3 , 7 }
- ❖ Now among all element of list of nodes node 3 has the smallest distance value.
- ❖ Node 3 becomes the current node and remove it from the list of nodes.

Step 3

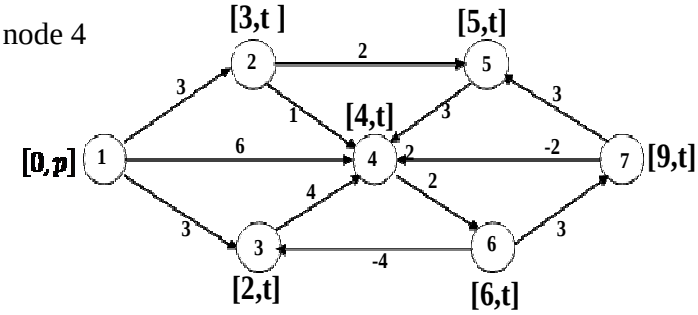
- ➢ We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

  So we perform other iteration (back to step 2)

Another implementation of step 2

- ❖ Node 4 can be reached from the current node 3.
- ❖ Update the distance value of node 4

$$d_4 = \min\{4, 2+4\} = 4$$



- List of nodes = $\{\ 7\ ,\ 4\ \}$
- Now among all element of list of nodes node 4 has the smallest distance value.
- Node 7 becomes the current node because we are work previously with node 4 as a current node and remove node 4 and 7 from the list of nodes.

Step 3

- We are not done .because all nodes have not been reached from the source node 1 and list of node is not empty.

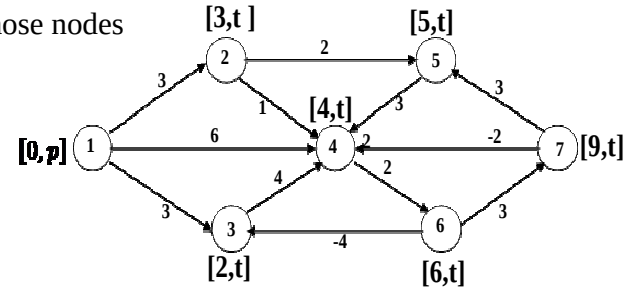
  So we perform other iteration (back to step 2)

Another implementation of step 2

- Node 4 and 5 can be reached from the current node 7.
- Update the distance value of those nodes

$$d_4 = \min\{\ 4\ ,\ 9-2\} = 4$$

$$d_5 = \min\{\ 5\ ,\ 9+3\} = 5$$



- List of nodes = $\{\ 4\ ,\ 5\ \}$
- Since the distance value of node 5 and 4 are not changed and we are work previously with node 4 and 5 as current node associated with the distance values 4 and 5 respectively those node can't chose as a current node therefore remove those nodes from the list of nodes.

Step 3

Stop because we are done

CHAPTER TWO

2. RANKED K- SHORTEST PATHS

k-shortest path problem is a natural and long studied generalization of the shortest path problem in which the objective of k-shortest path problem is not only to find the first shortest path as shortest path problem but also to find the second , third ,- - - , k^{th} shortest path (in order) from the source node to every other nodes in the network. Where each P_i should have cost greater or equal than P_{i-1} for $1 < i \leq k$ and the remain path between the source node and the destination node should have cost at least equal to p_k .We are given a weighted network (V, E, C) with node set V , edge set E and the weight set C specifying weight C_{ij} for the edges $(i, j) \in E$, we are also given a starting node $S \in V$ and terminal node $t \in V$.

To determine the set of ranked k-shortest paths, we select first a desired integer $k \geq 1$.then starting from node 1, step by step; we find a set of ranked k-shortest paths p_j^1 , p_j^2 ,…, p_j^k to each node j in the network. Here $p_j^r, r = 1, 2, \ldots, k$ refers to the r^{th} shortest path from node 1 to a node j . In general, to each node j we will assign appropriately a k-vector of distance label $d(j) = (d_j(1), d_j(2), \ldots, d_j(k))$ where r^{th} component label $d_j(r) = l(p_j^r)$ and $d_j(1) \leq d_j(2) \leq \ldots \leq d_j(k)$. To produce this, our procedure is essentially stated in the following algorithms as follow.

2.1 Algorithms solving k-shortest paths

There are many algorithms which solves k-shortest paths problem those algorithm can be studied based on their complexity and detailed description. In this paper we deal on the details of those algorithms rather than on their complexity.

In order to find k-shortest path it is important for the algorithm to choose different route throughout the network .This can be done by labeling node and edge or by

removing node and edge .Based on labeling node and edge or removing node and edge we classify those algorithm which solves k-shortest path problem in to two class

- The first class is labeling algorithm and
- The second class is path removing (deviation) algorithm.

2.1.1 Labeling algorithm

These algorithms behave much like classical labeling algorithm for shortest path problem. The difference is that labeling algorithm for k-shortest path problem keeps a list of label at each node while labeling algorithm for shortest path problem keeps only one label at each node.

There are two subclass of labeling algorithm namely

- Label setting algorithm and
- Label correcting algorithm

2.1.1.1 Label setting algorithm for k-shortest paths

Label setting algorithm keeps a list of labels at each node and the length of the list is K in each iteration the label with the lowest associated distance value (cost) in the network is chosen although there are many label at each node and this label update other label list such a label is treated as permanent label as in Dijakstra's algorithm. This implies that the k-shortest paths are ranked in order. Making a label at node t permanent identifies the next shortest path this algorithm precedes until there is no more update and the destination labels permanently k times.

Algorithm description step by step

Step 1, Initialization

- Assign zero distance value to node S and predecessor node of S is zero**.** i.e. $d(s)=0$, $pred(s)=0$ and [The label of node S is $[S,0(1),0]$ and label it permanently]
- Assign to every other node a distance value of ∞, k-times and label them as temporary.
- List of nodes with their predecessor node (L_1) = { [S , $0(1)$, 0] }
- Designate the node S as the current node and [S , $0(1)$, 0] as current label

Step 2, Distance value update (list) and current node designate update

Let i be the index of current node .

- Remove the current label from the list of nodes with their predecessor node.
- Select the set of nodes J that can be reached for the current node i by a link (i, j) which doesn't make cyclic path and update (list) the distance value of those nodes. For each $j \in J$ the r^{th} distance value of node j ($d_j(r)$) is update(list) as follows

$$d_j(r) = d_i(r_1) + C_{ij} \quad , \quad pred\,(j(r)) = i(r_1)$$

Where C_{ij} is the cost of link (i, j) as given in the network problem and $r = 1, 2, 3, \ldots, k$.

- Order and list associated distance values of each element of J , k - times in increasing order.

i.e. for each $j \in J$ $\quad d(j) = \{\, d_j(1)\,,\, d_j(2)\,,\, \ldots\,,\, d_j(k)\,\}$ where $d_j(1) \le d_j(2) \le, \ldots, \le d_j(k)$

- List of nodes with their predecessor node (L_1) = old L_1 $\cup$ { [reachable nodes from the current node i by a link (i, j). i.e. J , predecessor node of node J with its order , associated distance value of reachable nodes with their orders]}
- Determine a node j that has the smallest distance value $d_j(r)$ among all nodes.

Find j^* such that $\min_{j \in L_1} d_j(r) = d_{j^*}(*)$

Let $pred\,(j^*(*)) = i(\circ)$

- Change the label of node j^* with the distance value $d_{j^*}(*)$ to permanent and label node j^* with other distance value to temporary and designate

node j^* with the distance value $d_{j^*}(*)$ as a current node and designate the label $\lfloor\, j^*, i(\circ), d_{j^*}(*)\,\rfloor$ as a current label.

Step 3, Termination criteria

- If list of nodes with their predecessor node is not empty go to step 2.
- If list of nodes with their predecessor node is empty then

for j = 1 to n

for r = 1 to k

$$p_j^r = 1, i_1, i_2, \ldots, i_{s-1}, i_s, j$$

Where $\text{pred}(j(r)) = i_s(-)$, $\text{pred}(i_s(-)) = i_{s-1}(-)$, . . . , $\text{pred}(i_1(-)) = 1$

$l(p_j^r) = d_j(r)$

end;

end;

STOP

Remark

This algorithm check's whether the paths are cyclic or not for cyclic directed graph and undirected graph while it does not check for acyclic directed graph

Algorithm

Step 1: Initialization

begin

$d_s(1) = 0$ and $pred(s(1)) = 0$;

for each node $j \neq s$

for $r = 1$ *to* k

$d_j(r) = \infty$;

end:

end:

$old\ d_j(r) = d_j(r)$

LIST = { $[\, s, 0(1), d_s(1)\,]$ };

Step 2: Main step

Remove a label with minimum associated distance value from the LIST;

Current node = node associated with the above removed label.

Let i = current node and r_1 is order of i

for each arc $(i, j) \in A(i)$ **do**

if $p_j = 1, i_1, i_2, \ldots, i_s, i, j$ has no repeated node where $pred(j) = i(r_1)$, $pred(i(r_1)) = i_s(\bullet), \ldots, pred(i_1(\bullet)) = 1$ **then**

$d_j(r) = d_i(r_1) + c_{ij}$, $pred(j(r)) = i(r_1)$

for $q = 1$ *to* k

if $d_j(r) < old\ d_j(q)$ **then**

$r = q$;

if $r = 1$ **then**

for $p = r+1$ *to* k **do**

$d_j(r) = old\ d_j(p-1)$

end for;

end if ;

if $r = k$ **then**

for $p = 1$ *to* $r-1$ **do**

$d_j(r) = old\ d_j(p)$

end for;

end if ;

if $1 < r < k$ **then**

for $p = 1$ *to* $r-1$ **do**

$d_j(r) = old\ d_j(p)$

end for;

for $p = r+1$ *to* k **do**

$d_j(r) = old\ d_j(p-1)$

end for;

end if ;

$d(j) = \left(d_j(1), d_j(2), \ldots, d_j(k) \right)$

if $old\ d_j(k) \notin d(j)$ **then**

remove label which contain a distance value $old\ d_j(k)$ from a LIST;

end if;

end if;

break

else $r > k$ **end else;**

end for;

if $r \in \{ 1, 2, \ldots, k \}$ **then**

append $\lfloor j, i(r_1), d_j(r) \rfloor$ to the end of LIST;

end if ;

end for;

end if;

Step 3 : Termination criteria

if LIST $\neq \phi$ **then** go to step 2;

if LIST $= \phi$ **then**

for $j = 1$ *to* n

for $r = 1$ *to* k

$p_j^r = 1, i_1, i_2, \ldots, i_s, i, j$ where

$pred(j(r)) = i(r_1)$, $pred(i(r_1)) = i_s(\bullet), \ldots, pred(i_1(\bullet)) = 1;$

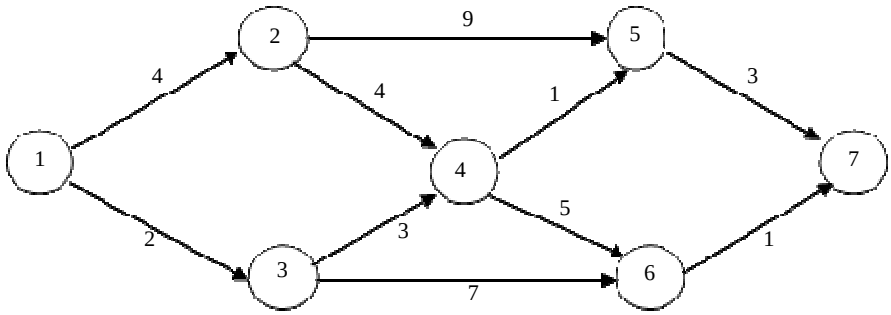
end for;

end for;

STOP

Example 3 for acyclic directed graph

We want to find ranked 3- shortest paths from the source node to all other nodes using label setting algorithm.



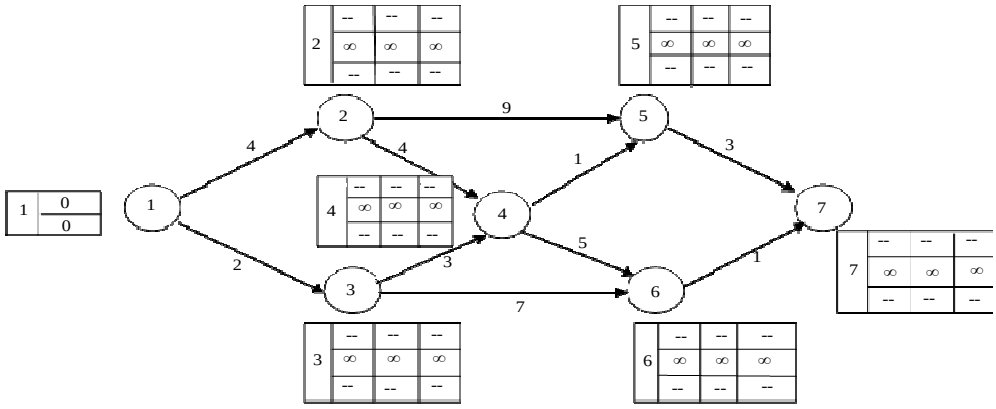
Solution

Step 1

- Node 1 with predecessory node zero and associated distance value zero is designated as the current node .

  i.e. $[\,1\,,\,0(1)\,,\,0\,]$ is the current node

- List of nodes with their predecessor node (L_1) = $\{[\,1\,,\,0(1)\,,\,0\,]\}$



Step 2

- Remove the label which has smallest associate distance value from the list of nodes with their predecessor node and find nodes that are reachable from the

current node. I.e. $[1,0(1),0]$ is removed from the list of nodes with their predecessor node and node 2 and 3 are reachable nodes from the current node 1.

- Update the distance value of those nodes

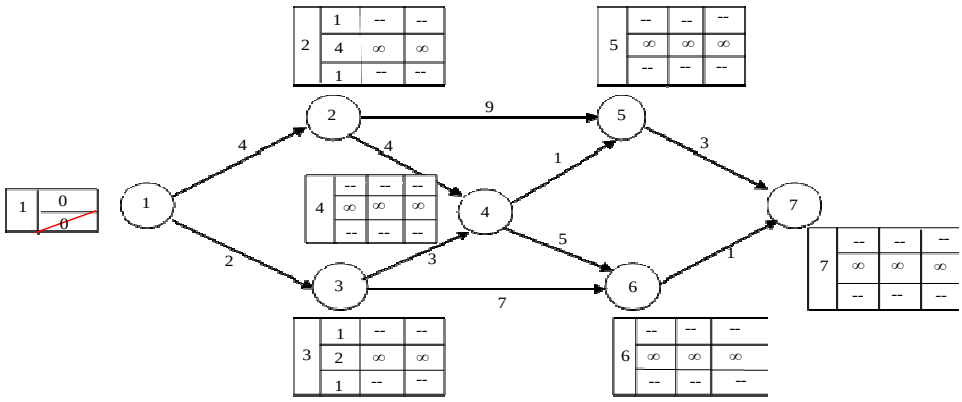
$$d_2(r) = d_1(1) + C_{12} = 0 + 4 = 4 \qquad r = 1 \quad , \quad pred(2(1)) = 1(1)$$

$$d_3(r) = d_1(1) + C_{13} = 0 + 2 = 2 \qquad r = 1 \quad , \quad pred(3(1)) = 1(1)$$

$$d(2) = \{\ d_2(1), -, -\} = \{4, -, -\}$$

$$d(3) = \{\ d_3(1), -, -\} = \{2, -, -\}$$

- Add the above nodes with their predecessor node which is the current node and their associated distance value at the end of list of nodes with their predecessor node.
  - $L_1 = \{[2, 1(1), 4], [3, 1(1), 2]\}$
- Designate the label which has small associated distance value from all elements of list of nodes with their predecessor node us a current label and the node associate with this label us current node.
  - I.e. label $[3, 1(1), 2]$ is the current label and node 3 is current node.



Step 3

- We are not done because list of nodes with their predecessor node is not empty so then go to step 2.

Another implementation of step 2

- Remove $[\,3,1(1),2\,]$ from the list of nodes with their predecessor node.
- Node 4 and 6 are reachable nodes from node 3.
- Update the distance value of those nodes

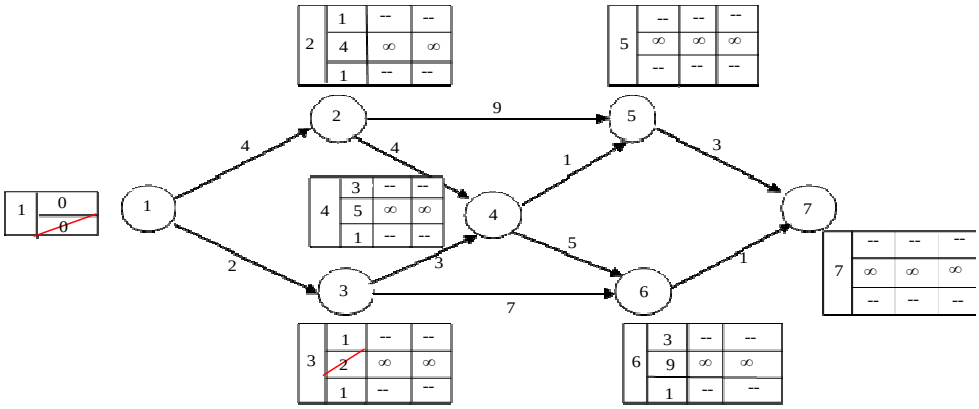
$$d_6(r)=d_3(1)+C_{36}=2+7=9 \qquad r=1 \quad , \quad pred(6(1))=3(1)$$

$$d_4(r)=d_3(1)+C_{34}=2+3=5 \qquad r=1 \quad , \quad pred(4(1))=3(1)$$

$$d(4)=\{\,d_4(1),-,-\}=\{5,-,-\}$$

$$d(6)=\{\,d_6(1),-,-\}=\{9,-,-\}$$

- $L_1 = \{[\,2\,,1(1),4\,],[\,4,3(1),5][\,6,3(1),9\,]\}$
- Label $[\,2,1(1),4\,]$ is current label and node 2 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

- Remove $[\,2,1(1),4\,]$ from the list of nodes with their predecessor node.
- Node 4 and 5 are reachable nodes from the current node 2.
- Update the distance value of those nodes.

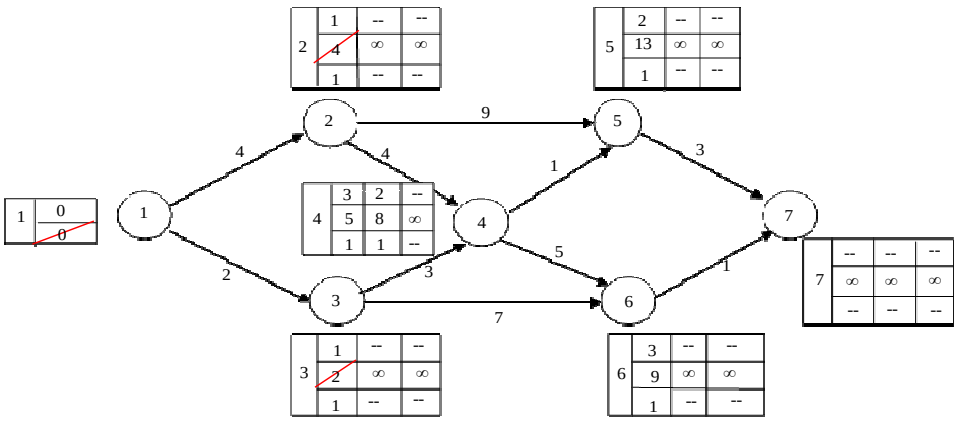
$$d_4(r) = d_2(1) + C_{24} = 4 + 4 = 8 \quad r = 2 \quad , \quad pred(4(2)) = 2(1)$$

$$d_5(r) = d_2(1) + C_{25} = 4 + 9 = 13 \quad r = 1 \quad , \quad pred(5(1)) = 2(1)$$

$d(4) = \{ d_4(1), d_4(2), - \} = \{ 5, 8, - \} \Rightarrow pred(4(1)) = 3(1)$

$d(5) = \{ d_5(1), -, - \} = \{ 13, -, - \}$

- $L_1 = \{ [4, 3(1), 5], [6, 3(1), 9], [4, 2(1), 8], [5, 2(1), 13] \}$
- Label $[4, 3(1), 5]$ is current label and node 4 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

- Remove $[4, 3(1), 5]$ from the list of nodes with their predecessor node.
- Node 5 and 6 are reachable nodes from the current node 4.
- Update the distance value of those nodes.

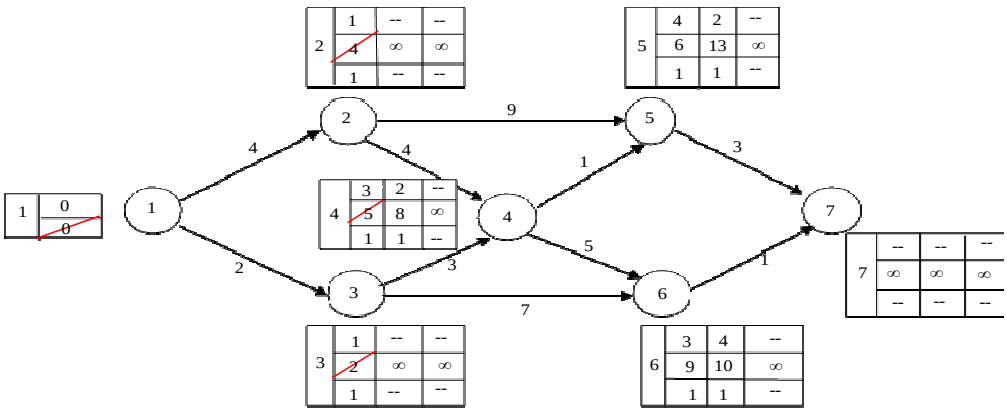
$$d_5(r) = d_4(1) + C_{45} = 5 + 1 = 6 \quad r = 1 \quad , \quad pred(5(1)) = 4(1)$$

$$d_6(r) = d_4(1) + C_{46} = 5 + 5 = 10 \quad r = 2 \quad , \quad pred(6(2)) = 4(1)$$

$d(5) = \{ d_5(1), d_5(2), - \} = \{ 6, 13, - \} \Rightarrow pred(5(2)) = 4(1)$

$d(6) = \{ d_6(1), d_6(2), - \} = \{ 9, 10, - \}$

- $L_1 = \{ [6, 3(1), 9], [4, 2(1), 8], [5, 2(1), 13] [5, 4(1), 6], [6, 4(1), 10] \}$
- Label $[5, 4(1), 6]$ is current label and node 4 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

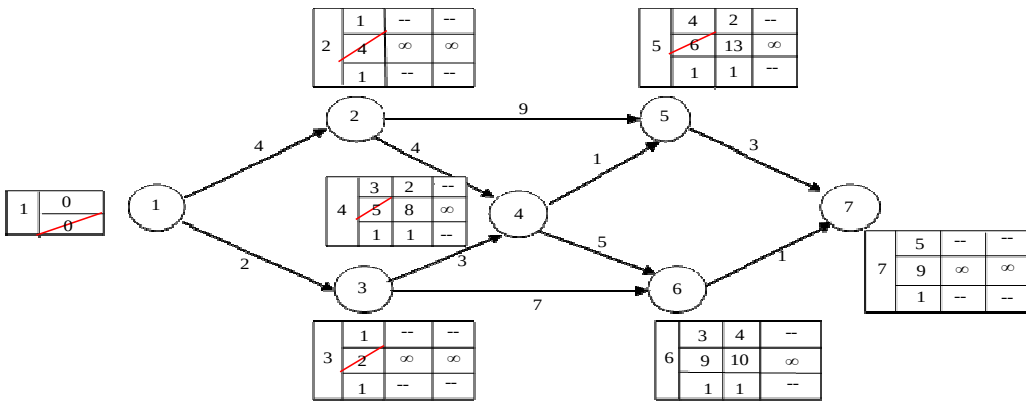
- Remove $[\,5\,,4\,(1)\,,6\,]$ from the list of nodes with their predecessor node.
- Node 7 is reachable node from the current node 5.
- Update the distance value of node 7.

$$d_7(r)=d_5(1)+C_{57}=6+3=9 \qquad r=1 \quad , \quad pred(7\,(1))=5\,(1)$$

$$d\,(7)=\{\;d_7(1),-,-\,\}=\{\,9\,,-,-\,\}$$

- L_1 = $\{\;[\,6\,,3\,(1)\,,9\,],[\,4\,,2\,(1)\,,8\,],[\,5\,,2\,(1)\,,13\,],[\,6\,,4\,(1)\,,10\,]\;,[\,7\,,5\,(1)\,,9\,]\}$

  Label $[\,4\,,2\,(1)\,,8\,]$ is current label and node 4 is current node



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

- Remove $[\,4\,,2\,(1)\,,8\,]$ from the list of nodes with their predecessor node.
- Node 5 and 6 are reachable nodes from the current node 4.
- Update the distance value of those nodes.

$$d_5(r)=d_4(2)+C_{45}=8+1=9 \qquad r=2 \quad , \quad pred(5\,(2))=4\,(2)$$

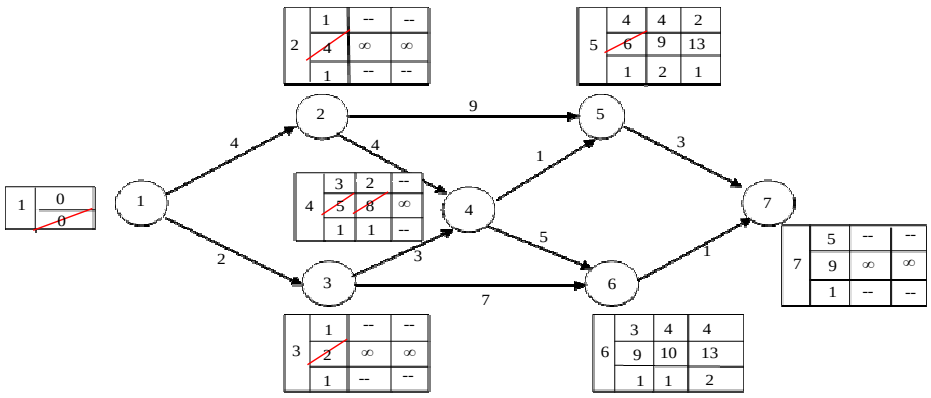
$$d_6(r)=d_4(2)+C_{46}=8+5=13 \qquad r=3 \quad , \quad pred(6\,(3))=4\,(2)$$

$$d\,(5)=\{\ d_5(1),\ d_5\,(2),d_5(3)\ \}=\{\,6\,,9\,,13\,\}\Rightarrow pred\,(5\,(3))=2\,(1)$$

$$d\,(6)=\{\ d_6(1),d_6(2),d_6(3)\,\}=\{\,9\,,10\,,13\,\}$$

- $L_1 = \left\{ \begin{matrix} [\,6\,,3\,(1)\,,9\,]\,,\ [\,5\,,2\,(1)\,,13\,],[\,6\,,4\,(1)\,,10\,]\,,[\,7\,,5\,(1)\,,9\,]\,, \\ [\,5\,,4\,(2)\,,9\,],[\,6\,,4\,(2)\,,13] \end{matrix} \right\}$

Label $[\,6\,,3\,(1)\,,9\,]$ is current label and node 6 is current node



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

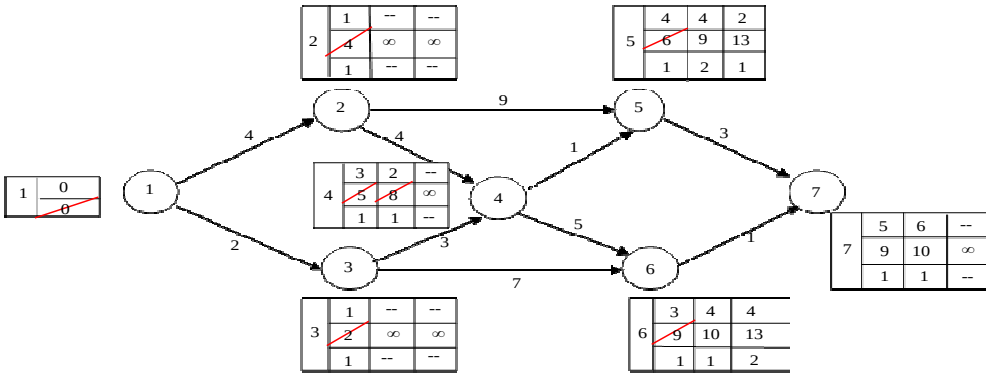
- Remove $[\,6\,,3\,(1)\,,9\,]$ from the list of nodes with their predecessor node.

- Node 7 is reachable node from the current node 6.
- Update the distance value of node 7.

$$d_7(r) = d_6(1) + C_{67} = 9 + 1 = 10 \quad r = 2 \ , \ pred(7(1)) = 6(1)$$

$$d(7) = \{ d_7(1), d_7(2), - \} = \{ 9, 10, - \}$$

- $L_1 = \left\{ \begin{matrix} [5, 2(1), 13], [6, 4(1), 10], [7, 5(1), 9], [5, 4(2), 9], \\ [6, 4(2), 13], [7, 6(1), 10] \end{matrix} \right\}$
- Label $[5, 4(2), 9]$ is current label and node 5 is current node



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

- Remove $[5, 4(2), 9]$ from the list of nodes with their predecessor node.
- Node 7 is reachable node from the current node 5.
- Update the distance value of node 7.

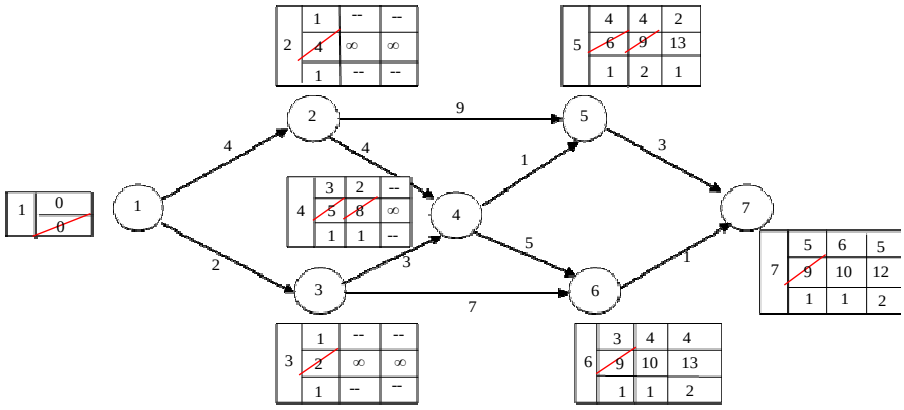
$$d_7(r) = d_5(2) + C_{57} = 9 + 3 = 12 \quad r = 3 \ , \ pred(7(3)) = 5(2)$$

$$d(7) = \{ d_7(1), d_7(2), d_7(3) \} = \{ 9, 10, 12 \}$$

- $L_1 = \left\{ \begin{matrix} [5, 2(1), 13], [6, 4(1), 10], [7, 5(1), 9], [6, 4(2), 13], \\ [7, 6(1), 10], [7, 5(2), 12] \end{matrix} \right\}$
- Label $[7, 5(1), 9]$ is current label and node 7 is current node since there is no node reachable from node 7 we remove this label from list of node with their

predecessor node and label $[\,6\,,4\,(1),10\,]$ is current label and node 6 is current node.

- $L_1 = \left\{ \begin{matrix} [\,5\,,2\,(1),13\,],[\,6\,,4\,(1),10\,],[\,6\,,4\,(2),13],\\ [\,7\,,6\,(1),10],[\,7\,,5\,(2),12\,] \end{matrix} \right\}$



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

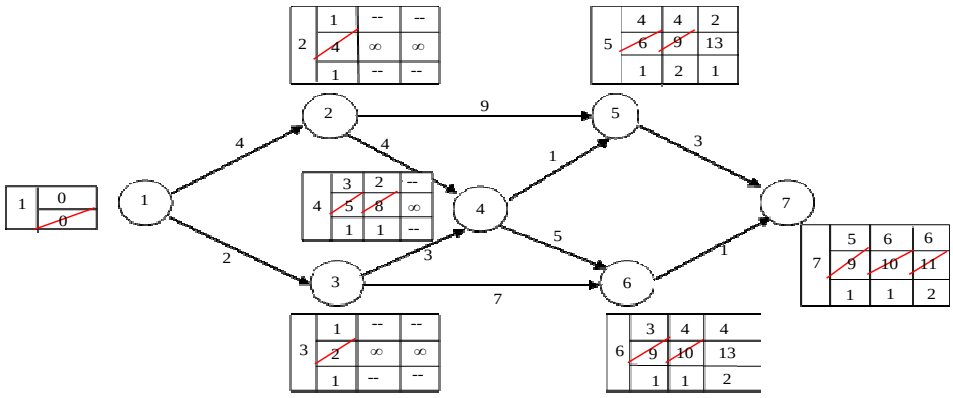
- Remove $[\,6\,,4\,(1),10\,]$ from the list of nodes with their predecessor node.
- Node 7 is reachable node from the current node 6.
- Update the distance value of node 7.

$$d_7(r) = d_6(2) + C_{67} = 10 + 1 = 11 \quad r = 3\ , \quad pred(7\,(3)) = 6\,(2)$$

$$d\,(7) = \{\ d_7(1), d_7(2), d_7(3)\,\} = \{\,9\,,10\,,11\,\}$$

- $L_1 = \{\ [\,5\,,2\,(1),13\,],[\,6\,,4\,(2),13],[\,7\,,6\,(1),10],\ [\,7\,,6\,(2),11\,]\,\}$
- Label $[\,7\,,6\,(1),10\,]$ and $[\,7\,,6\,(2),11\,]$ are current labels respectively and node 7 is current node since there is no node reachable from node 7 all labels of node 7 becomes permanent so we remove those labels from list of node with their predecessor node and label $[\,6\,,4\,(2),13\,]$ is current label and node 6 is current node.

- List of nodes with their predecessor node = $\{\ [5, 2(1), 13], [6, 4(2), 13]\ \}$



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

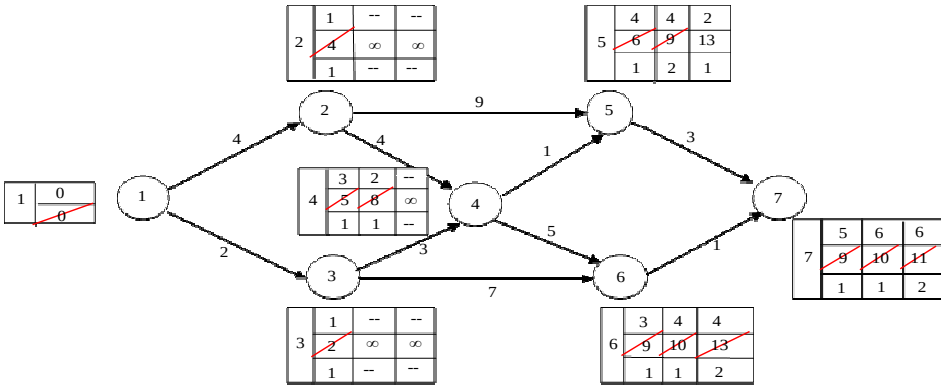
Another implementation of step 2

- Remove $[6, 4(2), 13]$ from the list of nodes with their predecessor node.
- Node 7 is reachable node from the current node 6.
- Update the distance value of node 7.

$$d_7(r) = d_6(3) + C_{67} = 13 + 1 = 14 \qquad r > 3$$

$$d(7) = \{\ d_7(1), d_7(2), d_7(3)\ \} = \{9, 10, 11\}$$

- L_1 = $\{\ [5, 2(1), 13]\ \}$
- Label $[5, 2(1), 13]$ is current label and node 5 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

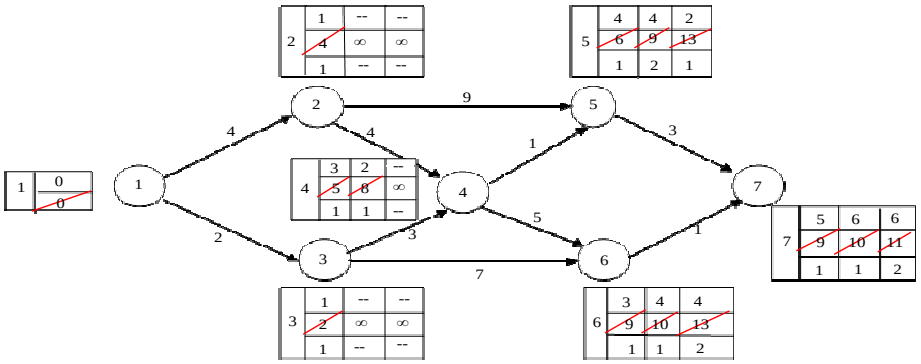
Another implementation of step 2

- Remove $[5, 2(1), 13]$ from the list of nodes with their predecessor node.
- Node 7 is reachable node from the current node 5.
- Update the distance value of node 7.

$$d_7(r) = d_5(3) + C_{57} = 13 + 3 = 16 \qquad r > 3$$

$$d(7) = \{ d_7(1), d_7(2), d_7(3) \} = \{ 9, 10, 11 \}$$

- List of nodes with their predecessor node = $\{ \phi \}$



Step 3

We are done because the list of nodes with their predecessor node is empty and all nodes are permanently label. List 3-shortest paths of node 7.

p_7^1 : 7 , pred$(7(1)) = 5(1)$, pred$(5(1)) = 4(1)$, pred$(4(1)) = 3(1)$, pred$(3(1)) = 1(1)$

$\Rightarrow p_6^1 = 1 - 3 - 4 - 5 - 7$ and $l(p_7^1) = 9$

p_7^2 : 7 , pred$(7(2)) = 6(1)$, pred$(6(1)) = 3(1)$, pred$(3(1)) = 1(1)$

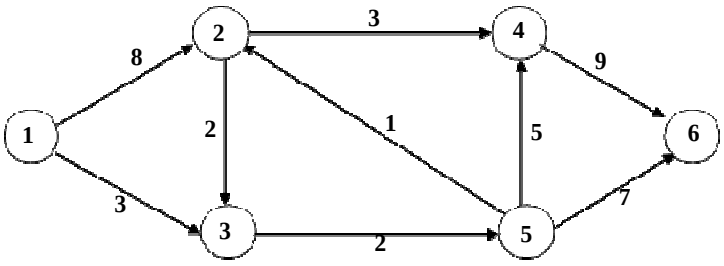
$\Rightarrow p_6^2 = 1 - 3 - 6 - 7$ and $l(p_7^2) = 10$

p_7^3 : 7 , pred$(7(3)) = 6(2)$, pred$(6(2)) = 4(1)$, pred$(4(1)) = 3(1)$, pred$(3(1)) = 1(1)$

$\Rightarrow p_7^3 = 1 - 3 - 4 - 6 - 7$ and $l(p_7^3) = 11$

Example 4 for cyclic directed graph

We want to find ranked 3 - shortest paths from the source node to all other nodes using label setting algorithm.



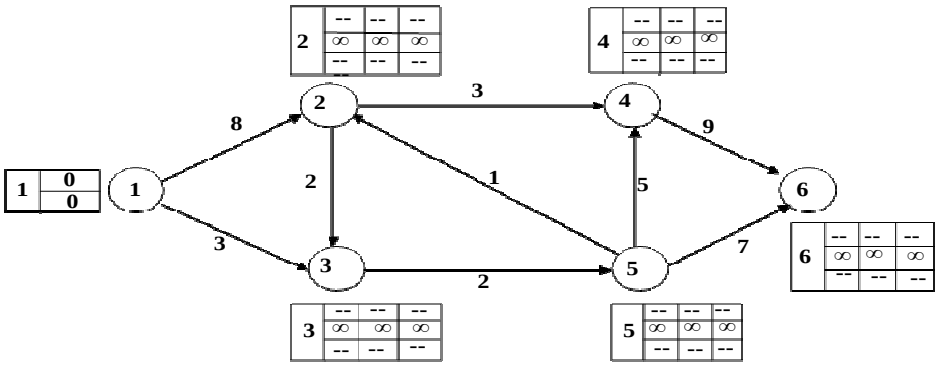
Solution

Step 1

- Node 1 with predecessory node zero and associated distance value zero is designated as the current label .

  i.e. $[1, 0(1), 0]$ is the current label and node 1 is current node.

- List of nodes with their predecessor node (L_1) = $\{[1, 0(1), 0]\}$



Step 2

- Remove the label which has smallest associate distance value from list of nodes with their predecessor node and find nodes that are reachable from the current node. I.e. $[1, 0(1), 0]$ is removed from the list of nodes with their predecessor node and node 2 and 3 can be reached from the current node 1.

- Update the distance value of those nodes

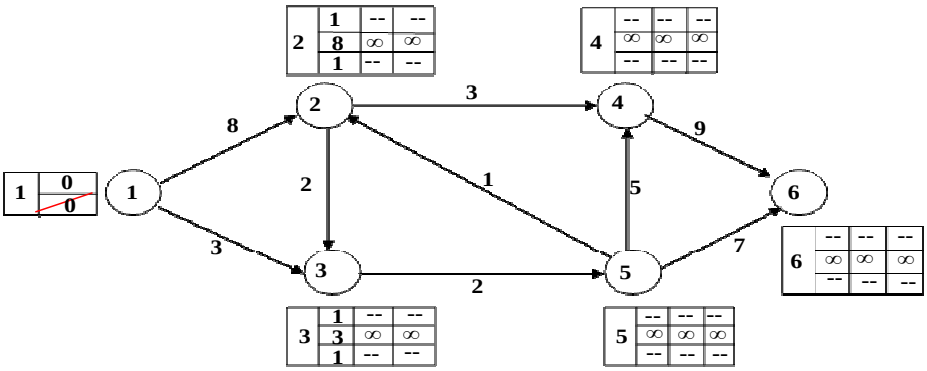
$$d_2(r) = d_1(1) + C_{12} = 0 + 8 = 8 \qquad r = 1 \quad , \quad pred(2(1)) = 1(1)$$

$$d_3(r) = d_1(1) + C_{13} = 0 + 3 = 3 \qquad r = 1 \quad , \quad pred(3(1)) = 1(1)$$

$$d(2) = \{\ d_2(1), -, -\} = \{8, -, -\}$$

$$d(3) = \{\ d_3(1), -, -\} = \{3, -, -\}$$

- Add the above nodes with their predecessor node which is the current node and their distance value to the end of list of nodes with their predecessor node.
  - $L_1 = \{[2, 1(1), 8], [3, 1(1), 3]\}$
- Designate the label which has small associated distance value from all elements of list of nodes with their predecessor node us a current label and the node associate with this label us current node.
  - I.e. label $[3, 1(1), 3]$ is the current label and node 3 is current node.



Step 3

- We are not done because list of nodes with their predecessor node is not empty then go to step 2.

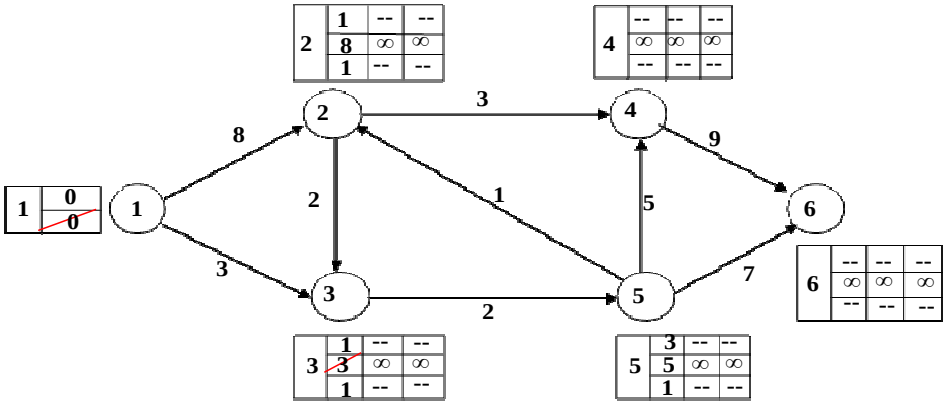
Another implementation of step 2

- Remove $[3, 1(1), 3]$ from the list of nodes with their predecessor node.
- Node 5 is reachable from node 3.
- Update the distance value of node 5

$$d_5(r) = d_3(1) + C_{35} = 3 + 2 = 5 \qquad r = 1 \quad , \quad pred(5(1)) = 3(1)$$

$$d(5) = \{\ d_5(1), -, -\} = \{5, -, -\}$$

- $L_1 = \{[2,1(1),8],[5,3(1),5]\}$
- Label $[5,3\ (1),5]$ is current label and node 5 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

Another implementation of step 2

- Remove $[5,3(1),5]$ from the list of nodes with their predecessor node.
- Node 2, 4 and 6 are reachable nodes from the current node 5.
- Update the distance value of those nodes.

$$d_2(r) = d_5(1) + C_{52} = 5+1 = 6 \quad r=1 \quad , \quad pred(2(1)) = 5(1)$$

$$d_4(r) = d_5(1) + C_{54} = 5+5 = 10 \quad r=1 \quad , \quad pred(4(1)) = 5(1)$$

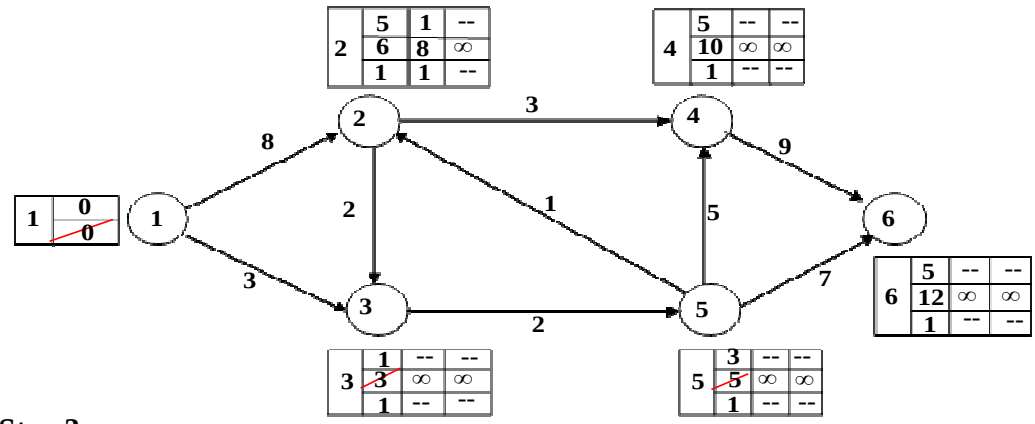
$$d_6(r) = d_5(1) + C_{56} = 5+7 = 12 \quad r=1 \quad , \quad pred(6(1)) = 5(1)$$

$$d(2) = \{d_2(1), d_2(2), -, -\} = \{6, 8, -\} \Rightarrow pred(2(2)) = 1(1)$$

$$d(4) = \{d_4(1), -, -\} = \{10, -, -\}$$

$$d(6) = \{d_6(1), -, -\} = \{12, -, -\}$$

- $L_1 = \{[2,1(1),8],[2,5(1),6],[4,5(1),10],[6,5(1),12]\}$
- Label $[2,5(1),6]$ is current label and node 2 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

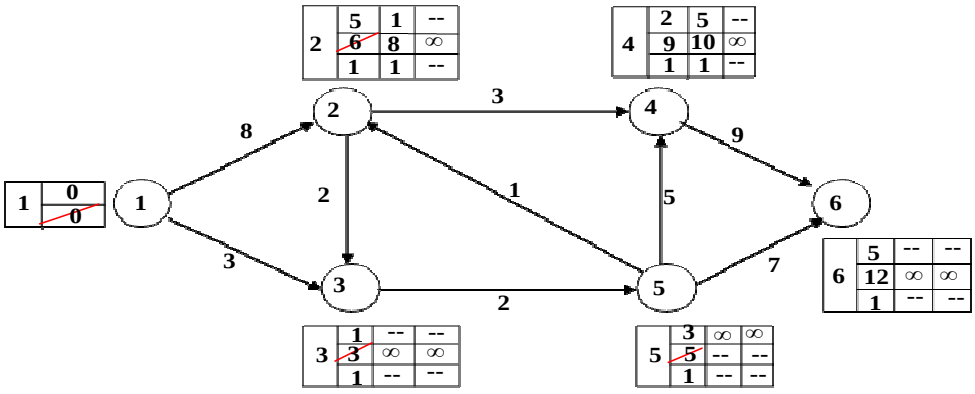
Another implementation of step 2

- Remove $[\,2\,,5\,(1)\,,6\,]$ from the list of nodes with their predecessor node.
- Node 4 is reachable node from the current node 2.
- Update the distance value of node 4.

$$d_4(r)=d_2(1)+C_{24}=6+3=9 \qquad r=1 \quad , \quad pred(4\,(1))=2\,(1)$$

$$d\,(4)=\{\ d_4(1),d_4(2),-\ \}=\{\,9\,,10\,,-\,\}\Rightarrow\ pred(4\,(2))=5\,(1)$$

- L_1 = $\{[\,2\,,1\,(1)\,,8\,]\,,\ [\,4\,,5\,(1)\,,10\,]\,,[\,6\,,5\,(1)\,,12\,]\,,[\,4\,,2\,(1)\,,9\,]\}$
- Label $[\,2\,,1\,(1)\,,8\,]$ is current label and node 2 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

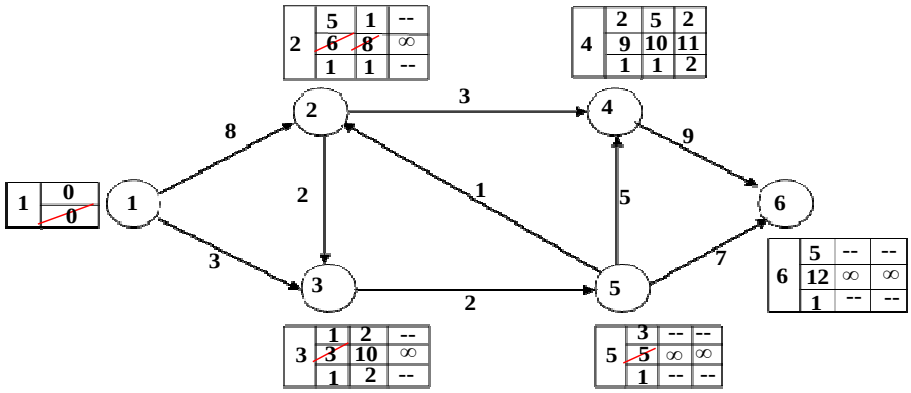
Another implementation of step 2

- Remove $[\,2\,,1\,(1)\,,8\,]$ from the list of nodes with their predecessor node.
- Node 3 and 4 are reachable nodes from the current node 2.
- Update the distance value of those nodes.

$$d_4(r)=d_2(2)+C_{24}=8+3=11 \quad r=3 \;,\; pred(4\,(3))=2\,(2)$$
$$d_3(r)=d_2(2)+C_{23}=8+2=10 \quad r=2 \;,\; pred(3\,(2))=2\,(2)$$

$$d\,(3)=\{\,d_3(1),d_3(2),-\}=\{3,10,-\}$$
$$d\,(4)=\{\,d_4(1),d_4(2),d_4(3)\}=\{9,10,11\}$$

- $L_1 = \left\{ \begin{matrix} [\,4\,,5\,(1)\,,10\,],[\,6\,,5\,(1)\,,12\,],[\,4\,,2\,(1)\,,9\,], \\ [\,3\,,2\,(2)\,,10\,],[\,4\,,2\,(2)\,,11\,] \end{matrix} \right\}$
- Label $[\,4\,,2\,(1)\,,9\,]$ is current label and node 4 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

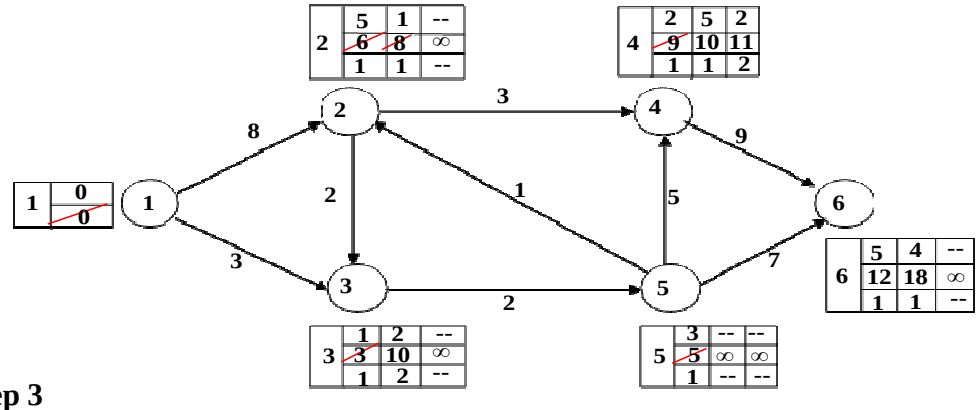
Another implementation of step 2

- Remove $[\,4\,,2\,(1)\,,9\,]$ from the list of nodes with their predecessor node.
- Node 6 is reachable node from the current node 4.
- Update the distance value of node 6.

$$d_6(r)=d_4(1)+C_{46}=9+9=18 \quad r=2 \;,\; pred(6\,(2))=4\,(1)$$

$$d\,(6)=\{\,d_6(1),d_6\,(2),-\}=\{12,18,-\}$$

- $L_1 = \left\{ \begin{matrix} [\,4,5(1),10\,],[\,6,5(1),12\,],[\,6,4(1),18\,], \\ [\,3,2(2),10\,],[\,4,2(2),11\,] \end{matrix} \right\}$
- Label $[\,4,5(1),10\,]$ is current label and node 4 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

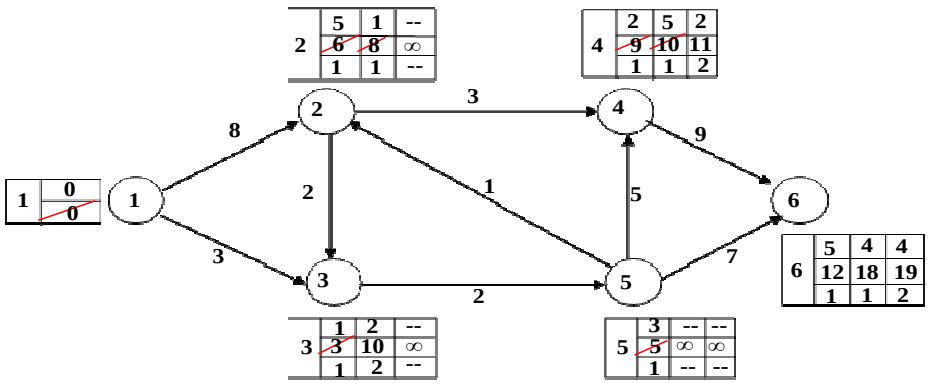
Another implementation of step 2

- Remove $[\,4,5(1),10\,]$ from the list of nodes with their predecessor node.
- Node 6 is reachable node from the current node 4.
- Update the distance value of node 6.

$$d_6(r) = d_4(2) + C_{46} = 10 + 9 = 19 \quad r = 3\ , \quad pred(6(3)) = 4(2)$$

$$d(6) = \{\, d_6(1), d_6(2), d_6(3) \,\} = \{12, 18, 19\}$$

- $L_1 = \left\{ \begin{matrix} [\,6,5(1),12\,],[\,6,4(1),18\,],[\,3,2(2),10\,], \\ [\,4,2(2),11\,],[\,6,4(2),19\,] \end{matrix} \right\}$
- Label $[\,3,2(2),10\,]$ is current label and node 3 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

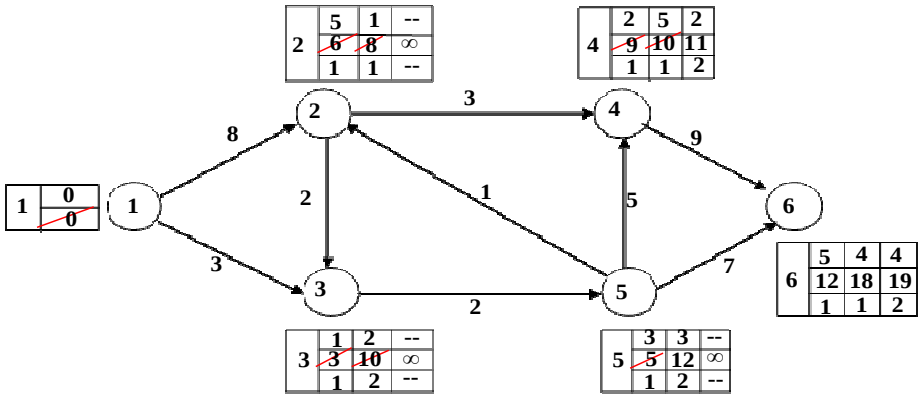
Another implementation of step 2

- Remove $[3, 2(2), 10]$ from the list of nodes with their predecessor node.
- Node 5 is reachable node from the current node 3.
- Update the distance value of node 5.

$$d_5(r) = d_3(2) + C_{35} = 10 + 2 = 12 \qquad r = 2\ , \quad pred(5(2)) = 3(2)$$

$$d(5) = \{\ d_5(1), d_5(2), -\} = \{5, 12, -\}$$

- $L_1 = \{\ [6, 5(1), 12], [6, 4(1), 18], [4, 2(2), 11]\ , [6, 4(2), 19], [5, 3(2), 12]\}$
- Label $[4, 2(2), 11]$ is current label and node 4 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

Another implementation of step 2

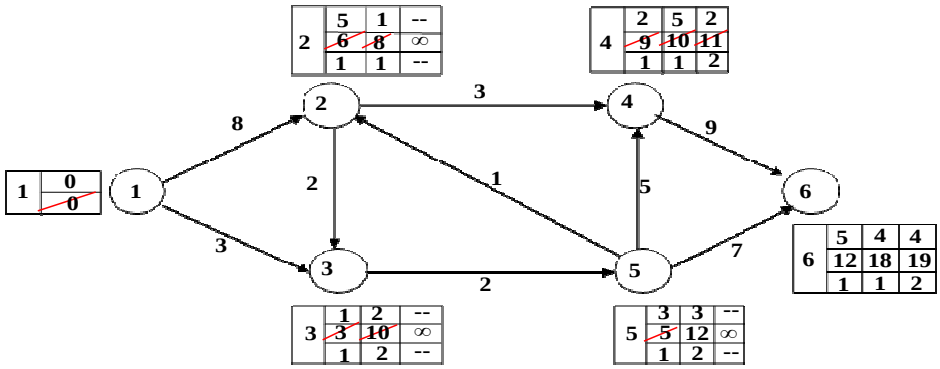
- Remove $[4, 2(2), 11]$ from the list of nodes with their predecessor node.
- Node 6 is reachable node from the current node 4.

- Update the distance value of node 6.

$$d_6(r) = d_4(3) + C_{46} = 11 + 9 = 20 \qquad r > 3$$

$$d(6) = \{ d_6(1), d_6(2), d_6(3) \} = \{12, 18, 19\}$$

- $L_1 = \{ [6, 5(1), 12], [6, 4(1), 18], [6, 4(2), 19], [5, 3(2), 12] \}$
- Label $[5, 3(2), 12]$ is current label and node 5 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

Another implementation of step 2

- Remove $[5, 3(2), 12]$ from the list of nodes with their predecessor node.
- Node 4 and 6 are reachable node from the current node 5.
- Update the distance value of node 4 and 6.

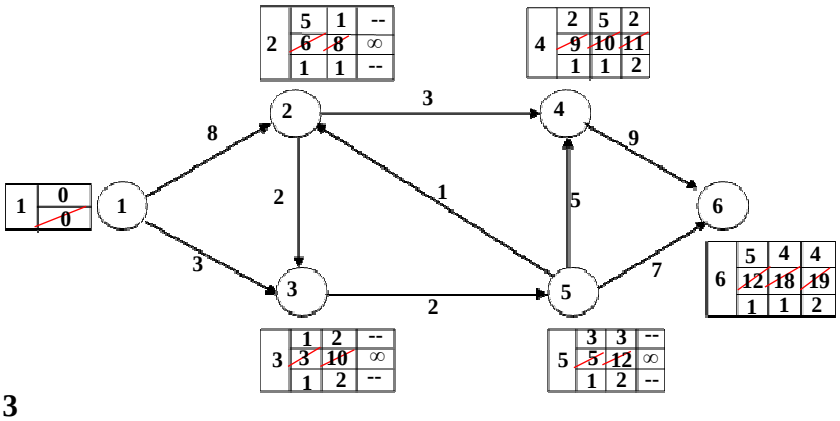
$$d_4(r) = d_5(2) + C_{54} = 12 + 5 = 17 \qquad r > 3$$

$$d_6(r) = d_5(2) + C_{56} = 12 + 7 = 19 \qquad r \geq 3$$

$$d(4) = \{ d_4(1), d_4(2), d_4(3) \} = \{9, 10, 11\}$$

$$d(6) = \{ d_6(1), d_6(2), d_6(3) \} = \{12, 18, 19\}$$

- $L_1 = \{ [6, 5(1), 12], [6, 4(1), 18], [6, 4(2), 19] \}$
- Label $[6, 5(1), 12]$, $[6, 4(1), 14]$ and $[6, 4(2), 19]$ are current labels respectively and node 6 is current node since there is no node reachable from node 6 all labels of node 6 becomes permanent so we remove those labels from list of node with their predecessor node



Step 3

We are done because the list of nodes with their predecessor node is empty and all labels become permanent. List 3-shortest paths of node 6

p_6^1: 6 , $\text{pred}(6(1)) = 5(1)$, $\text{pred}(5(1)) = 3(1)$, $\text{pred}(3(1)) = 1(1)$

$\Rightarrow p_6^1 = 1-3-5-6$ and $l(p_6^1) = 12$

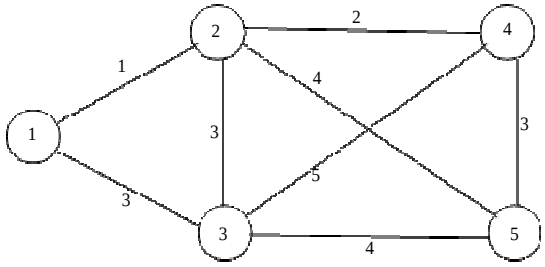
p_6^2 : $6, \text{pred}(6(2)) = 4(1), \text{pred}(4(1)) = 2(1), \text{pred}(2(1)) = 5(1), \text{pred}(5(1)) = 3(1),$ $\text{pred}(3(1)) = 1(1)$

$\Rightarrow p_6^2 = 1-3-5-2-4-6$ and $l(p_6^2) = 18$

p_6^3: 6 , $\text{pred}(6(3)) = 4(2)$, $\text{pred}(4(2)) = 5(1)$, $\text{pred}(5(1)) = 3(1)$, $\text{pred}(3(1)) = 1(1)$

$\Rightarrow p_6^3 = 1-3-5-4-6$

Example 5 for undirected graph



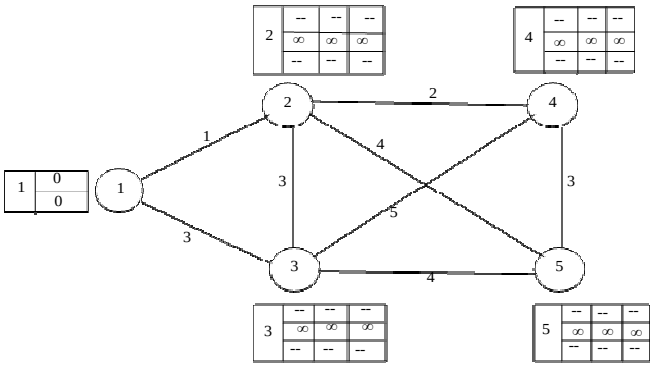
Solution

Step 1

- Node 1 with predecessory node zero and associated distance value zero is designated as the current node .

  i.e. $[1, 0(1), 0]$ is the current node

- List of nodes with their predecessor node (L_1) = $\{[1, 0(1), 0]\}$



Step 2

Remove $[1, 0(1), 0]$ from L_1 and $J = \{2, 3\}$

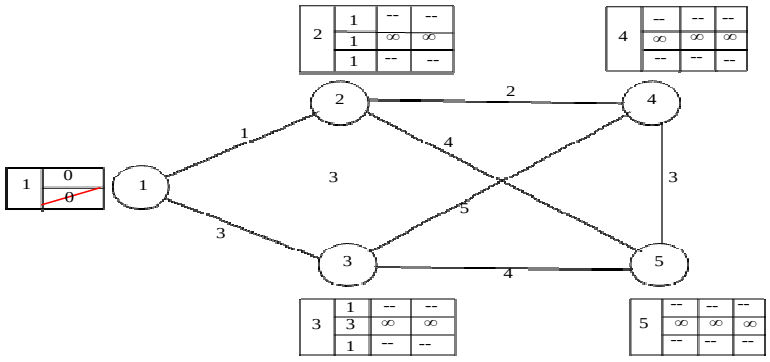
$d_2(r) = d_1(1) + C_{12} = 0 + 1 = 1 \qquad r = 1 \quad , \quad pred(2(1)) = 1(1)$,

$d_3(r) = d_1(1) + C_{13} = 0 + 3 = 3 \qquad r = 1 \quad , \quad pred(3(1)) = 1(1)$

$$d(2) = \{ d_2(1), -, - \} = \{1, -, -\}$$
$$d(3) = \{ d_3(1), -, - \} = \{3, -, -\}$$

$$L_1 = \{[2, 1(1), 1], [3, 1(1), 3]\}$$

Current label = $[\,2\,,1(1),1\,]$



Another implementation of step 2

Remove $[\,2\,,1(1),1]$ from L_1 and $J=\{3\,,4\,,5\,\}$

$d_3(r)=d_2(1)+C_{23}=1+3=4 \qquad r=2\ ,\quad pred(3(2))=2(1),$

$d_4(r)=d_2(1)+C_{24}=1+2=3 \quad r=1\ ,\ pred(4(1))=2(1)$

$d_5(r)=d_2(1)+C_{25}=1+4=5 \quad r=1\ ,\quad pred(5(1))=2(1)$

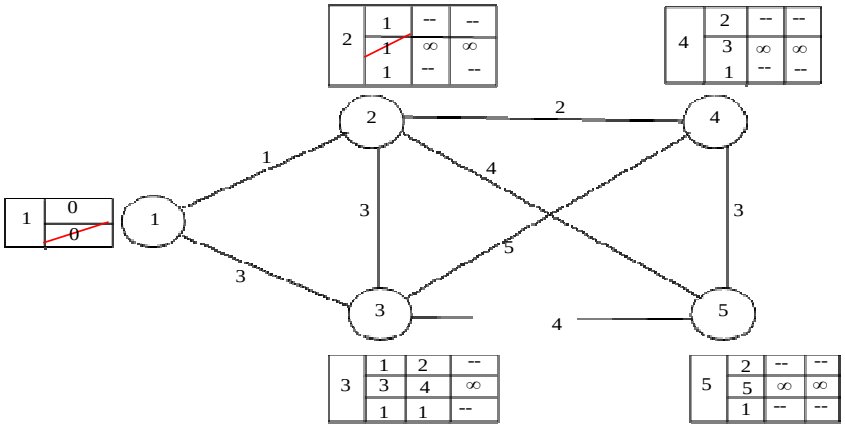
$d(3)=\{\ d_3(1),d_3(2),-\}=\{3\,,4\,,-\}$

$d(4)=\{\ d_4(1),-,-\}=\{3\,,-,-\}$

$d(5)=\{\ d_5(1),-,-\}=\{5\,,-,-\}$

$L_1 = \{\,[\,3\,,1(1),3\,]\,,[\,3\,,2(1),4\,],[4\,,2(1),3\,],[\,5\,,2(1),5\,]\}$

Current label = $[\,3\,,1(1),3\,]$



Another implementation of step 2

Remove $[3,1(1),3]$ from L_1 and $J=\{2,4,5\}$

$d_2(r)=d_3(1)+C_{32}=3+3=6 \quad r=2 \quad , \quad pred(2(2))=3(1),$

$d_4(r)=d_{13}(1)+C_{34}=3+5=8 \quad r=2 \quad , \quad pred(4(2))=3(1)$

$d_5(r)=d_3(1)+C_{35}=3+4=7 \quad r=2 \quad , \quad pred(5(2))=3(1)$ and

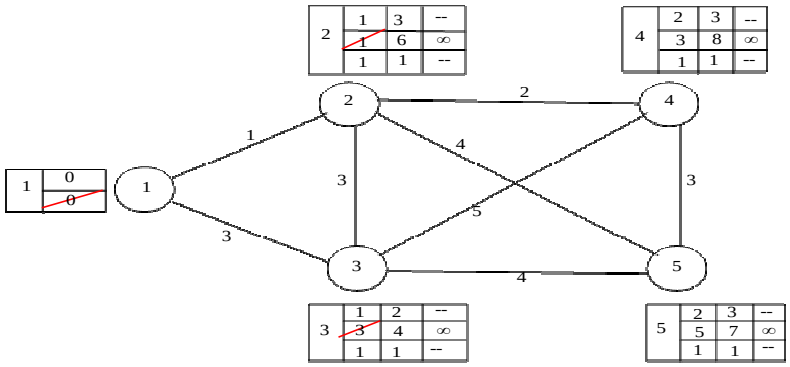
$$d(2)=\{d_2(1),d_2(2),-\}=\{1,6,-\}$$

$$d(4)=\{d_4(1),d_4(2),-\}=\{3,8,-\}$$

$$d(5)=\{d_5(1),d_5(2),-\}=\{5,7,-\}$$

$$L_1=\{[3,2(1),4],[4,2(1),3],[5,2(1),5],[2,3(1),6],[4,3(1),8],[5,3(1),7]\}$$

Current label = $[4,2(1),3]$



Another implementation of step 2

Remove $[4,2(1),3]$ from L_1 and $J=\{3,5\}$

$d_3(r)=d_4(1)+C_{43}=3+5=8 \quad r=3 \quad , \quad pred(3(3))=4(1),$

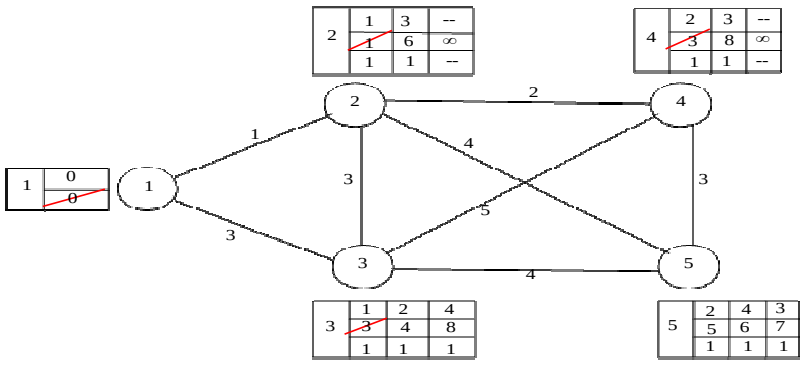
$d_5(r)=d_4(1)+C_{45}=3+3=6 \quad r=2 \quad , \quad pred(5(2))=4(1)$

$$d(3)=\{d_3(1),d_3(2),d_3(3)\}=\{3,4,8\}$$

$$d(5)=\{d_5(1),d_5(2),d_5(3)\}=\{5,6,7\} \Rightarrow pred(5(3))=3(1)$$

$$L_1 = \begin{Bmatrix} [3,2(1),4],\ [5,2(1),5],[2,3(1),6],[4,3(1),8],[5,3(1),7], \\ [3\,4(1)\,8,],[5,4(1),6] \end{Bmatrix}$$

Current label = $[3,2(1),4]$



Another implementation of step 2

Remove $[3,2(1),4]$ from L_1 and $J = \{4,5\}$

$d_4(r) = d_3(2) + C_{34} = 4+5 = 9 \quad r = 3 \quad , \quad pred(4(3)) = 3(2),$

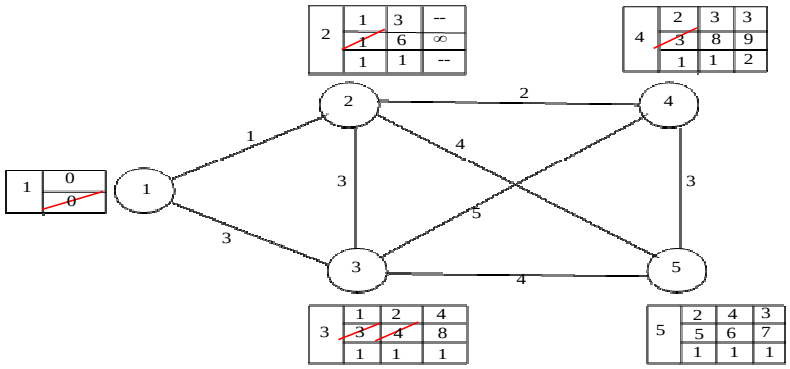
$d_5(r) = d_3(2) + C_{35} = 4+4 = 8 \qquad r > 3$

$d(4) = \{d_4(1), d_4(2), d_4(3)\} = \{3,8,9\}$

$d(5) = \{d_5(1), d_5(2), d_5(3)\} = \{5,6,7\}$

$$L_1 = \begin{Bmatrix} [5,2(1),5],[2,3(1),6],[4,3(1),8],[5,3(1),7], \\ [3,4(1)\,8,],[5,4(1),6],[4,3(2),9] \end{Bmatrix}$$

Current label = $[5,2(1),5]$



Another implementation of step 2

Remove $[5,2(1),5]$ from L_1 and $J=\{3,4\}$

$d_3(r)=d_5(1)+C_{53}=5+4=9 \qquad r>3,$

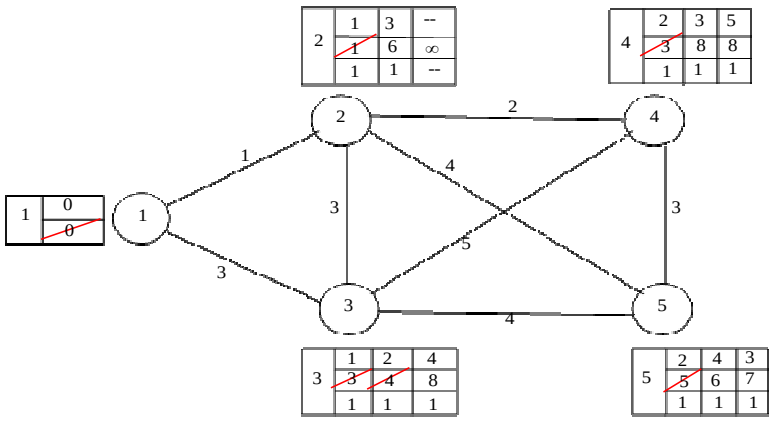
$d_4(r)=d_5(1)+C_{54}=5+3=8 \qquad r=3 \quad , \quad pred(4(3))=5(1)$

$d(3)=\{d_3(1),d_3(2),d_3(3)\}=\{3,4,8\}$

$d(4)=\{d_4(1),d_4(2),d_4(3)\}=\{3,8,8\}$

$$L_1=\left\{\begin{matrix}[2,3(1),6],[4,3(1),8],[5,3(1),7],\\ [3,4(1)\ 8,],[5,4(1),6],[4,5(1),8]\end{matrix}\right\}$$

Current label = $[2,3(1),6]$



Another implementation of step 2

Remove $[2,3(1),6]$ from L_1 and $J=\{4,5\}$

$d_4(r)=d_2(2)+C_{24}=6+2=8 \qquad r\geq 3$

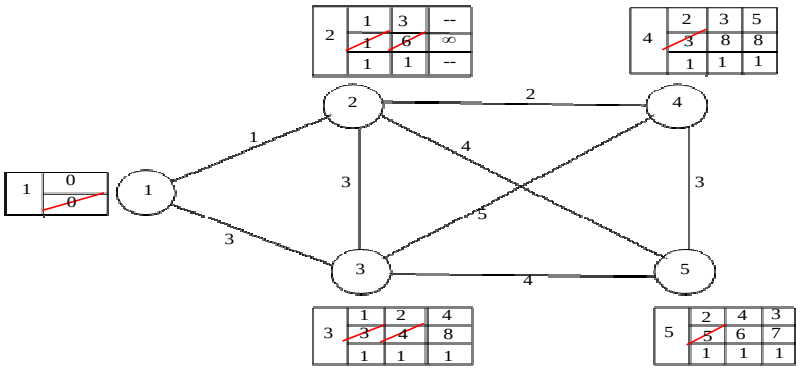
$d_5(r)=d_2(2)+C_{25}=6+5=10 \qquad r>3$

$d(4)=\{d_4(1),d_4(2),d_4(3)\}=\{3,8,8\}$

$d(5)=\{d_5(1),d_5(2),d_5(3)\}=\{5,6,7\}$

$$L_1=\left\{\begin{matrix}[4,3(1),8],[5,3(1),7],[3,4(1),8],\\ [5,4(1),6],[4,5(1),8]\end{matrix}\right\}$$

Current label = $[5,4(1),6]$



Another implementation of step 2

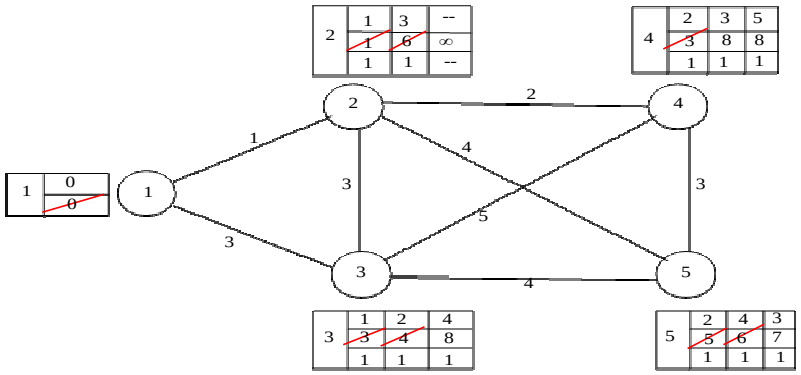
Remove $[5, 4(1), 6]$ from L_1 and $J = \{3\}$

$d_3(r) = d_5(2) + C_{43} = 6 + 4 = 10 \qquad r > 3,$

$$d(3) = \{d_3(1), d_3(2), d_3(3)\} = \{3, 4, 8\}$$

$$L_1 = \{[4, 3(1), 8], [5, 3(1), 7], [3, 4(1), 8], [4, 5(1), 8]\}$$

Current label = $[5, 3(1), 7]$



Another implementation of step 2

Remove $[5, 3(1), 7]$ from L_1 and $J = \{2, 4\}$

$d_2(r) = d_5(3) + C_{52} = 7 + 4 = 11 \qquad r = 3\ ,\ pred(2(3)) = 5(3),$

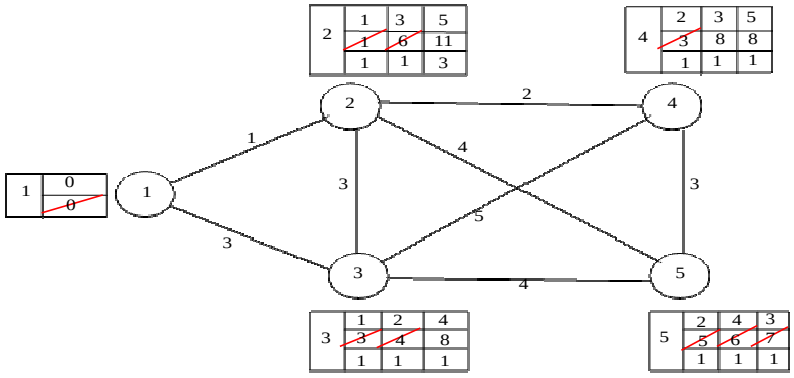
$d_4(r) = d_5(3) + C_{54} = 7 + 3 = 10 \qquad r > 3$

$d(2) = \{\ d_2(1), d_2(2), d_2(3)\ \} = \{1, 6, 11\}$

$d(4) = \{\ d_4(1), d_4(2), d_4(3)\ \} = \{3, 8, 8\}$

$L_1 = \{\ [4, 3(1), 8], [3, 4(1), 8], [4, 5(1), 8], [2, 5(3), 11]\}$

Current label = $[3, 4(1), 8]$



Another implementation of step 2

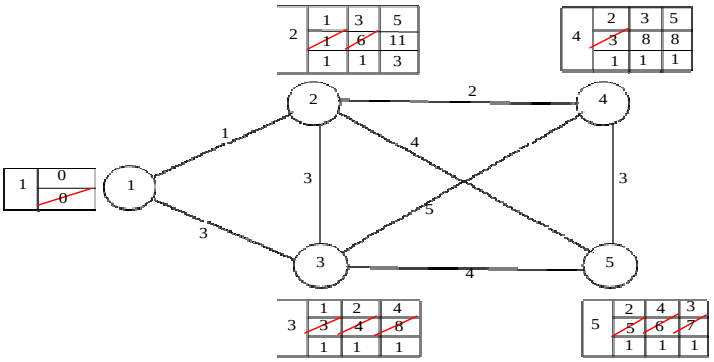
Remove $[3, 4(1), 8]$ from L_1 and $J = \{\ 5\ \}$

$d_5(r) = d_3(3) + C_{35} = 8 + 4 = 12 \qquad r > 3$

$d(5) = \{\ d_5(1), d_5(2), d_5(3)\ \} = \{5, 6, 7\}$

$L_1 = \{\ [4, 3(1), 8], [4, 5(1), 8], [2, 5(3), 11]\}$

Current label = $[4, 3(1), 8]$



Another implementation of step 2

Remove $[4,3(1),8]$ from L_1 and $J=\{2,5\}$

$d_5(r)=d_4(2)+C_{45}=8+3=11 \qquad r>3,$

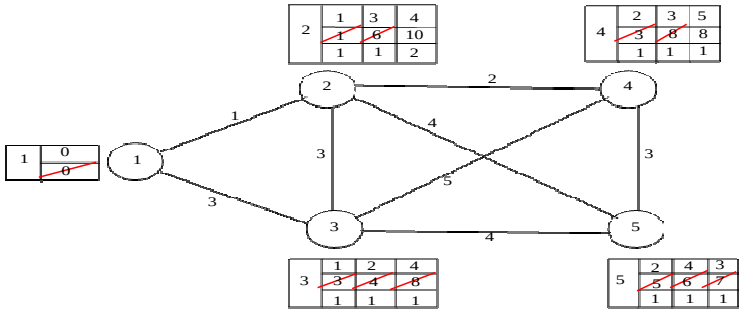
$d_2(r)=d_4(2)+C_{42}=8+2=10 \qquad r=3 \quad , \quad pred(2(3))=4(2)$

$d(2)=\{d_2(1),d_2(2),d_2(3)\}=\{1,6,10\}$

$d(5)=\{d_5(1),d_5(2),d_5(3)\}=\{5,6,7\}$

$L_1=\{[4,5(1),8],[2,4(2),10]\}$

Current label = $[4,5(1),8]$



Another implementation of step 2

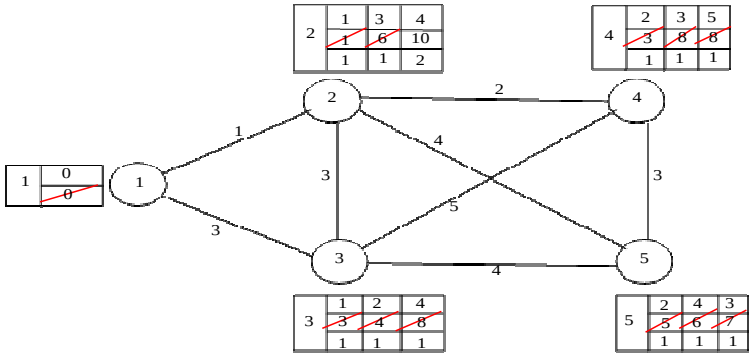
Remove $[4,5(1),8]$ from L_1 and $J=\{3\}$

$d_3(r)=d_4(3)+C_{43}=8+5=13 \qquad r>3$

$d(3)=\{d_3(1),d_3(2),d_3(3)\}=\{3,4,8\}$

$L_1=\{[2,4(2),10]\}$

Current label = $[2,4(2),10]$



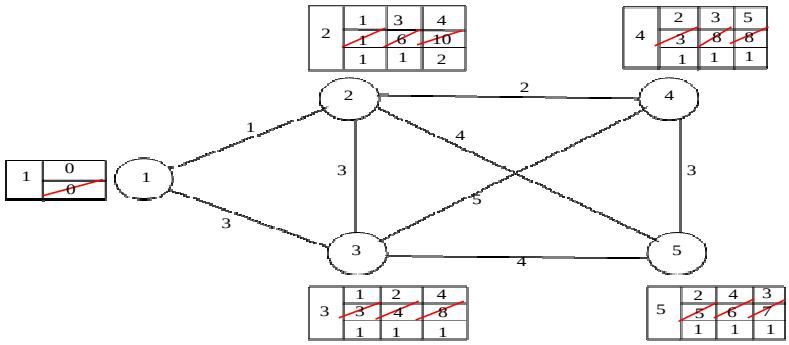
Another implementation of step 2

Remove $[\,2\,,4\,(\,2\,),10\,]$ from L_1 and $J=\{\ 5\ \}$

$\Rightarrow d_5(r)=d_2(3)+C_{25}=10+4=14 \qquad r>3$

$d\,(5)=\{\ d_5(1),d_5(2),d_5(3)\ \}=\{5\,,6\,,7\,\}$

$L_1=\{\ \phi\ \}$



Step 3

We are done because the list of nodes with their predecessor node is empty and all labels become permanent**.** List 3-shortest paths of node 5

$p_5^1:\ 5\ ,\ \text{pred}\,(5(1))=2(1),\text{pred}\,(2\,(1)\,)=1\,(1\,)$

$\Rightarrow\ p_5^1=1-2-5\quad\text{and}\quad l\,(p_5^{\,1})=5$

$p_5^{\,2}\ :\ 5\,,\text{pred}(5\,(2))=4(1),\text{pred}\,(4\,(1))=2\,(1),\text{pred}\,(2\,(1))=1\,(1)$

$\Rightarrow\ p_5^{\,2}=1-2-4-5\ \text{and}\ l\,(p_5^{\,2})=6$

$p_5^3:\ 5\ ,\ \text{pred}\,(5(3))=3(1),\text{pred}\,(3\,(1)\,)=1\,(1\,)$

$\Rightarrow\ p_5^{\,3}=1-3-5\quad\text{and}\quad l\,(p_5^{\,3})=7$

Complexity of k-shortest path problem

To determine the k-shortest paths

- For acyclic directed graph at most $k|V|$ comparisons and $k|E|$ addition are made by the algorithm to extract minimum distance value and update the distance value of all vertices adjacent to V respectively.
- For cyclic directed graph at most $k|V|$ comparisons, $k|E|$ addition and $k|E|$ checking are made by the algorithm to extract minimum distance value ,update the distance value of all vertices adjacent to V and whether there is a repeated node or not on each path respectively.
- For cyclic directed graph at most $k|V|$ comparisons, $k|E|$ addition and $2k|E|$ checking are made by the algorithm to extract minimum distance value ,update the distance value of all vertices adjacent to V and whether there is a repeated node or not on each path respectively.

2.1.1.2 Label correcting algorithm

A label correcting algorithm keeps a list of labels at each node. The length of the list is K.one node is selected in each iteration and the associated label list is used to update other label lists. The algorithm proceeds until there is no more update.

A more flexible way to perform label correcting is to separate labels and nodes. Rather than treating the whole label list at once. One single label is treated in each iteration. An obvious way to do this to have a FIFO queue of labels.

Algorithm description step by step

Step 1, Initialization

- Assign zero distance value to node S and predecessor node of S is zero**.** i.e. $d(s)=0$, $pred(s)=0$ and [The label of node S is $[S,0(1),0]$]
- Assign to every other node a distance value of ∞, k-times and label them as temporary.
- Enter the source node to the list of nodes with their predecessor node

i.e. List of nodes with their predecessor node (L) = { $[S,0(1),0]$ }

- Designate the node S as the current node and label $[S,0(1),0]$ as current label

Step 2, Distance value update (list) and current node designate update

Let i be the index of current node.

- Remove the first element of list of nodes with their predecessor node (current label).
- Select the set of nodes J that can be reached for the current node i by a link (i, j) which doesn't make cycle and update (list) the distance value of those nodes. For each $j \in J$ the r^{th} distance value of node j ($d_j(r)$) is update(list) as follows

$$d_j(r) = d_i(r_1) + C_{ij} \quad , \quad pred\,(j(r)) = i(r_1)$$

Where C_{ij} is the cost of link (i, j) as given in the network problem and $r = 1, 2, 3, \ldots, k$.

- Order and list associated distance values of each element of J, k -times in increasing order.

I.e. for each $j \in J$ $d\,(j) = \{\ d_j(1)\ ,\ d_j(2)\ ,\ \ldots\ ,\ d_j(k)\ \}$

Where $d_j(1) \le d_j(2) \le, \ldots, \le d_j(k)$.

- List of nodes with their predecessor node (L_1) = old L_1 $\cup$ { [reachable nodes from the current node i. i.e. J, predecessor node with their order, associated distance value] }
- Designate the first element of list of nodes with their predecessor node as a current label and its associated node as a current node.

Step 3, Termination criteria

- If list of nodes with their predecessor node is not empty go to step 2.
- If there is a negative weighted cyclic path stop because there is no shortest path.

- If list of nodes with their predecessor node is empty then

  For j = 1 to n

  For r = 1 to k

$$p_j^r = 1, i_1, i_2, \ldots, i_{s-1}, i_s, j$$

Where $\text{pred}(j(r)) = i_s(-)$, $\text{pred}(i_s(-)) = i_{s-1}(-)$, ... , $\text{pred}(i_1(-)) = 1$

$l(p_j^r) = d_j(r)$

End

End

STOP

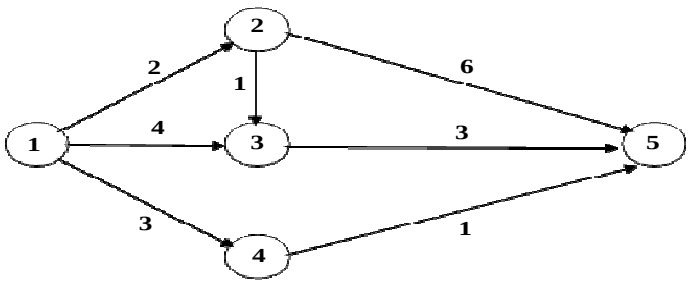
Algorithm

I state each steps of label correcting algorithm in a similar way as a label setting algorithm except remove a label with minimum associated distance value from the LIST because label correcting algorithm removes the first element of the LIST rather than removing the label with minimum associated distance value as label setting algorithm. Hence I didn't write each step of the algorithm because it will become a full of redundancy.

In order to illustrate the working of label correcting algorithm let us see the following

Example 6

We want to find 3-shortest paths from the source node to all other nodes using label correcting algorithm.



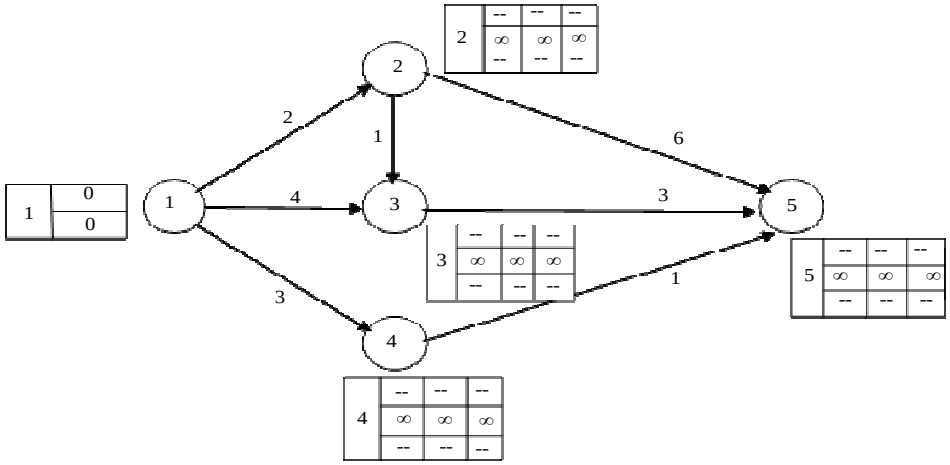
Solution

Step 1

- Node 1 with predecessory node zero associated distance value is designated as the current node .

  i.e. $[1, 0(1), 0]$ is the current node

- List of nodes with their predecessor node = $\{ [1, 0(1), 0] \}$



Step 2

- Remove the first element of the list of nodes with their predecessor node and find nodes that are reachable from the current label and current node. i.e. $[1, 0, 0]$ is removed from the list of nodes with their predecessor node and node 2,3 and 4 are reachable nodes from the current node 1.

- Update the distance value of those nodes

$$d_2(r) = d_1(1) + C_{12} = 0 + 2 = 2 \qquad r = 1 \ , pred\,(2(1)) = 1(1)$$

$$d_3(r) = d_1(1) + C_{13} = 0 + 4 = 4 \qquad r = 1 \ , pred(3(1)) = 1(1)$$

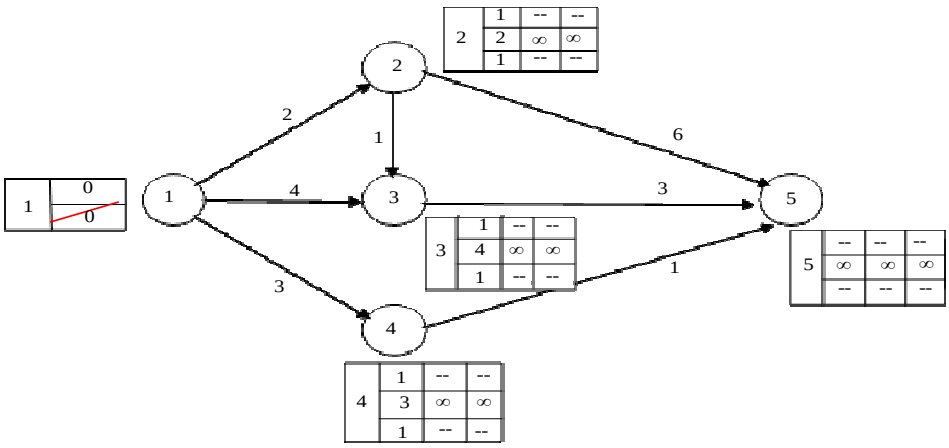
$$d_4(1) = d_1(1) + C_{14} = 0 + 3 = 3 \qquad r = 1 \ , pred(4(1)) = 1(1)$$

$$d\,(2) = \{\ d_2(1), -, -\} = \{2, -, -\}$$

$$d\,(3) = \{\ d_3(1), -, -\} = \{4, -, -\}$$

$$d\,(4) = \{\ d_4(1), -, -\} = \{3, -, -\}$$

- Add the above nodes with their predecessor node and their associated distance value to the end of list of nodes with their predecessor node.
  - $L_1 = \{[2, 1(1), 2], [3, 1(1), 4], [4, 1(1), 3]\}$
- Designate the first element of the list of nodes with their predecessor node us a current label and the node associate with this label us current node.
  - i.e. label $[2, 1(1), 2]$ is the current label and node 2 is current node.



Step 3

- We are not done because list of nodes with their predecessor node is not empty then go to step 2.

Another implementation of step 2

- Remove $[2, 1(1), 2]$ from the list of nodes with their predecessor node.
- Node 3 and 5 are reachable nodes from node 2.
- Update the distance value of those nodes

$$d_3(r) = d_2(1) + C_{23} = 2 + 1 = 8 \qquad r = 1 \;,\; pred(3(1)) = 2(1)$$

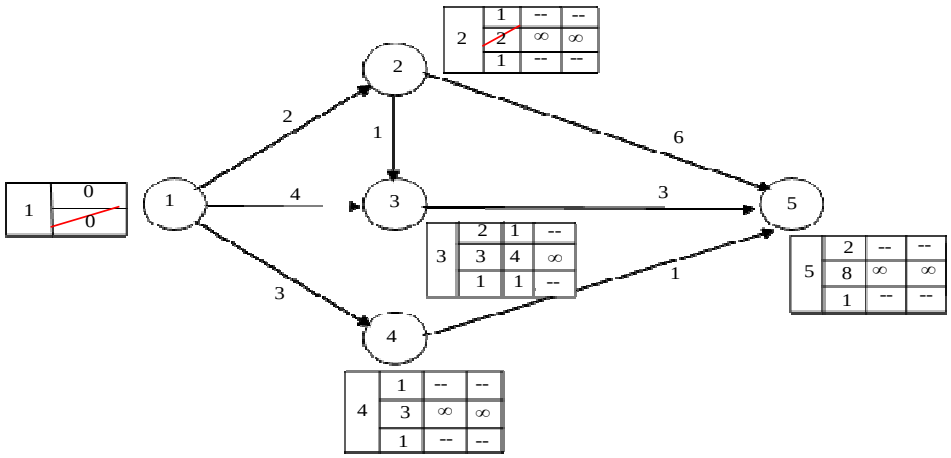
$$d_5(r) = d_2(1) + C_{25} = 2 + 6 = 8 \qquad r = 1 \;,\; pred(5(1)) = 2(1)$$

$$d(3) = \{d_3(1), d_3(2), -\} = \{3, 4, -\} \Rightarrow pred(3(2)) = 1(1)$$

$$d(5) = \{d_5(1), -, -\} = \{8, -, -\}$$

- $L_1 = \{[3, 1(1), 4], [4, 1(1), 3], [3, 2(1), 3], [5, 2(1), 8]\}$

- Label $[\,3,1(1),4\,]$ is current label and node 3 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2.

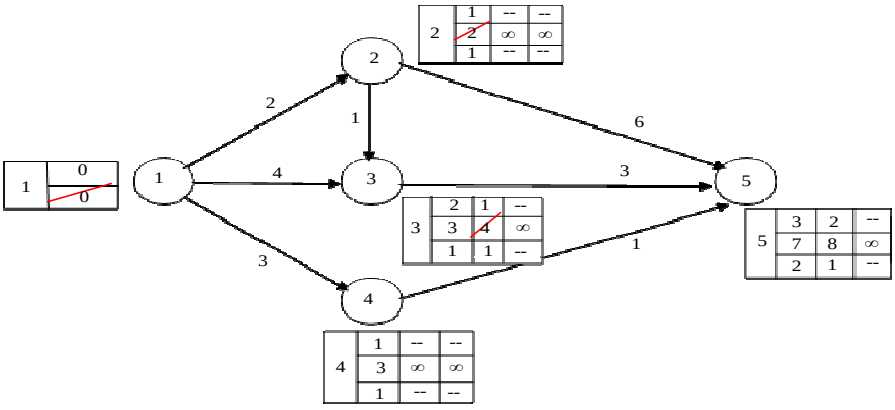
Another implementation of step 2

- Remove $[\,3,1(1),4\,]$ from the list of nodes with their predecessor node.
- Node 5 is reachable node from node 3.
- Update the distance value of node 5.

$$d_5(r)=d_3(2)+C_{35}=4+3=7 \qquad r=1 \quad , \quad pred(5(1))=3\,(2)$$

$$d\,(5)=\{\,d_5(1),d_5(2),-\,\}=\{\,7\,,8\,,-\,\}\Rightarrow pred(5(2))=2\,(1)$$

- $L_1=\{[\,4\,,1(1),3\,],[\,3\,\,,2(1),3\,],[\,5\,,2(1),8\,],[\,5\,,3(2),7\,]\,\}$
- Label $[\,4\,,1(1),3\,]$ is current label and node 4 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

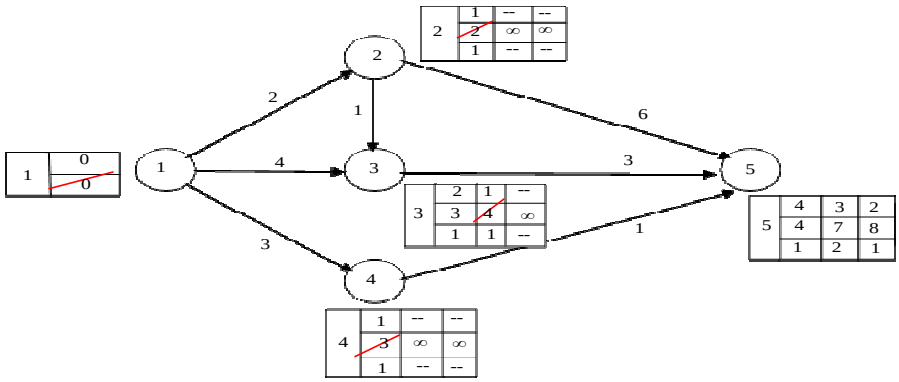
Another implementation of step 2

- Remove $[\,4\,,1(1),3\,]$ from the list of nodes with their predecessor node.
- Node 5 is reachable node from node 4.
- Update the distance value of node 5.

$$d_5(r)=d_4(1)+C_{45}=3+1=4 \qquad r=1 \quad , \quad pred\,(5(1))=4\,(1)$$

$$d\,(5)=\{\,d_5(1),d_5(2),d_5(3)\,\}=\{\,4\,,7\,,8\,\}\Rightarrow pred\,(5(2))=3(2)\ ,\quad pred(5(3))=2(1)$$

- $L_1=\{\ [\,3\ ,2(1),3\,],[\,5\,,2(1),8\,],[\,5\,,3(2),7\,],[5\,,4(1),4]\,\}$
- Label $[\,3\,,2(1),3\,]$ is current label and node 3 is current node.



Step 3

We are not done because list of nodes with their predecessor node is not empty. then go to step 2..

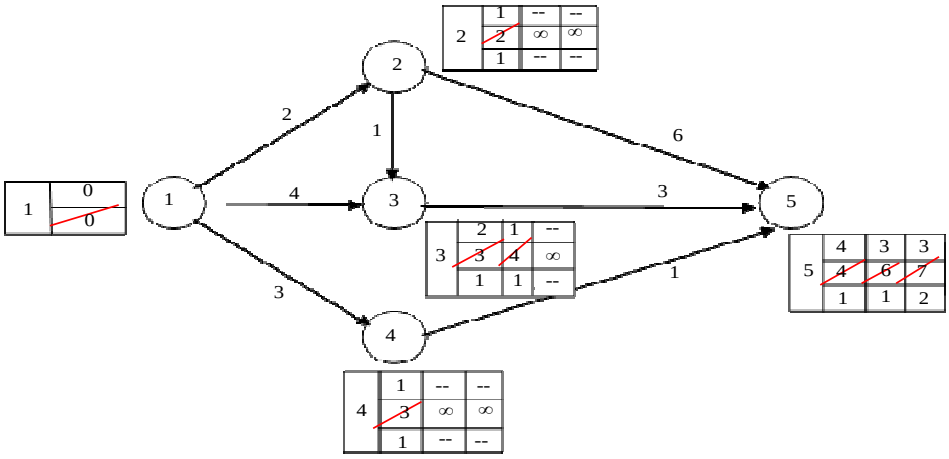
Another implementation of step 2

- Remove $[\,3\,,2(1),3\,]$ from the list of nodes with their predecessor node.
- Node 5 is reachable node from node 3.
- Update the distance value of node 5.

$$d_5(r) = d_3(1) + C_{35} = 3 + 3 = 6 \qquad r = 2 \quad , \quad pred(5(2)) = 3(1)$$

$$d(5) = \{ d_5(1), d_5(2), d_5(3) \} = \{ 4, 6, 7 \} \Rightarrow pred(5(1)) = 4(1) \ , \ pred(5(3)) = 3(2)$$

- $L_1 = \{ [5, 3(2), 7], [5, 4(1), 4], [5, 3(1), 6] \}$
- Label $[5, 3(2), 7]$, $[5, 4(1), 4]$ and $[5, 3(1), 6]$ are current labels respectively and node 5 is current node since there is no node reachable from node 5 all labels of node 5 becomes permanent so we remove those labels from list of node with their predecessor node.
- $L_1 =$ empty



Step 3

We are done because the list of nodes with their predecessor node is empty and all labels become permanent. List 3-shortest paths of node 5

p_5^1 : 5 , pred $(5(1)) = 4(1)$, pred $(4(1)) = 1(1)$

$$\Rightarrow p_5^1 = 1 - 4 - 5 \quad \text{and} \quad l(p_5^1) = 4$$

p_5^2 : 5, pred$(5(2)) = 3(1)$, pred $(3(1)) = 2(1)$, pred $(2(1)) = 1(1)$

$$\Rightarrow p_5^2 = 1 - 2 - 3 - 5 \text{ and } l(p_5^2) = 6$$

p_5^3 : 5 , pred $(5(3)) = 3(2)$, pred $(3(2)) = 1(1)$

$$\Rightarrow p_5^3 = 1 - 3 - 5 \quad \text{and} \quad l(p_5^3) = 7$$

2.1.2 path Removing algorithm

This algorithm was proposed by martin in 1984. The main idea takes in to account is the following property the second shortest path P_2 in G is the shortest path in a new network G^1, obtained from G removing the shortest path P_1.in addition the third shortest path P_3 in G corresponding to the shortest path in network G^2 obtained from G removing P_1 and P_2 , or removing P_2 from G^1.consequently the general step of the algorithm are

- Removing the shortest path in the current network.
- Determining the shortest path in the resulting network.

Remark: This algorithm is more applicable to find disjoint paths

Algorithm

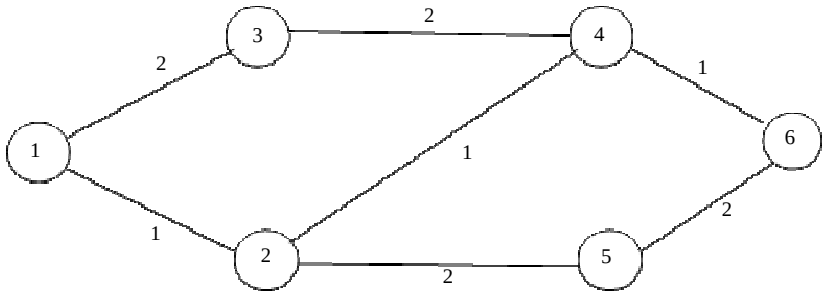
for $i = 0$ *to* k

- Find the shortest path tree T rooted at node S and shortest path (p_{i+1}) by running Dijkstra's algorithm in the graph G^i . This tree contains for every vertex j , a shortest path from S to j. Let p_{i+1} be the shortest cost path from S to t in the graph G^i. The edges in T are called tree edges and the remaining edges are called non tree edges.
- Modify the cost of each edge in the graph by replacing the cost C_{uv} of every edge (u, v) by $C'_{uv} = C_{uv} - d_v + d_u$. According to the resulting modified cost function, all tree edges have a cost of 0, and non tree edges have a non negative cost.
- Create a residual graph G^{i+1} formed from G^i by removing the edges of G^i that are directed into S and by reversing the direction of the zero length edges along path p_{i+1}
- **end for;**

- Find the shortest paths by combining the edges of p_1 , p_2 ,. . . , p_k and then discarding the common reversed edges of all paths.

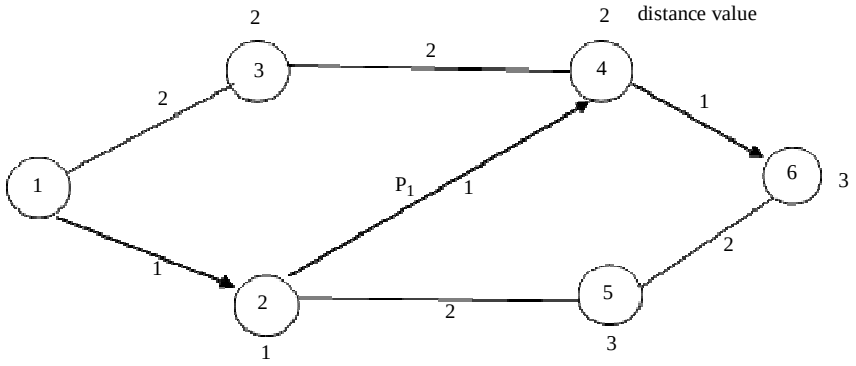
Example

The following example show how path removing algorithm find the shortest pair of disjoint paths from the source node 1 to destination node 6.

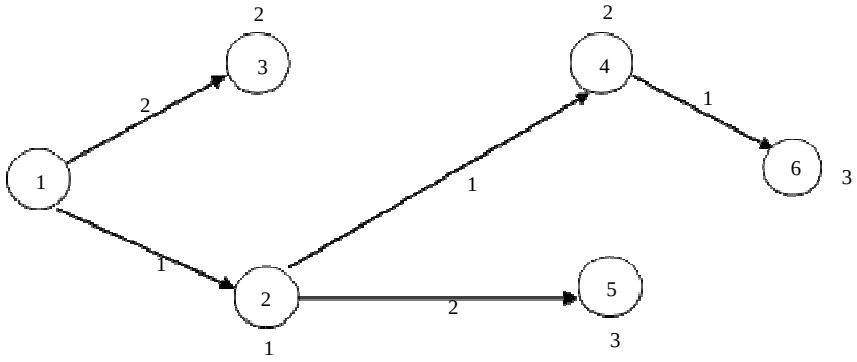


Solution

By using Dijakstra's algorithm we get the first shortest path (p_1 = 1-2-4-6)



And the shortest path tree T rooted at node 1 and the computed distance from node 1 to every vertex is



Now let us update cost of each arc and reverse the edge of path p_1 to obtain the new graph G^1 .

$C_{ij}^{'} = C_{ij} - d_j + d_i$

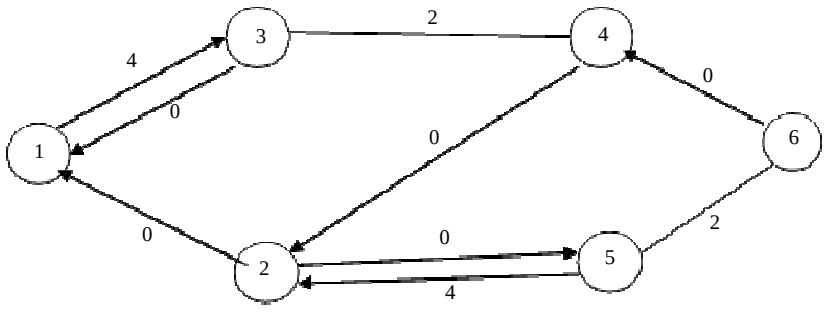
$C_{13}^{'} = C_{13} - d_3 + d_1 = 2-2+0 = 0$, $C_{31}^{'} = C_{31} - d_1 + d_3 = 2-0+2 = 4$

$C_{25}^{'} = C_{25} - d_5 + d_2 = 2-3+1 = 0$ $C_{52}^{'} = C_{52} - d_2 + d_5 = 2-1+3 = 4$

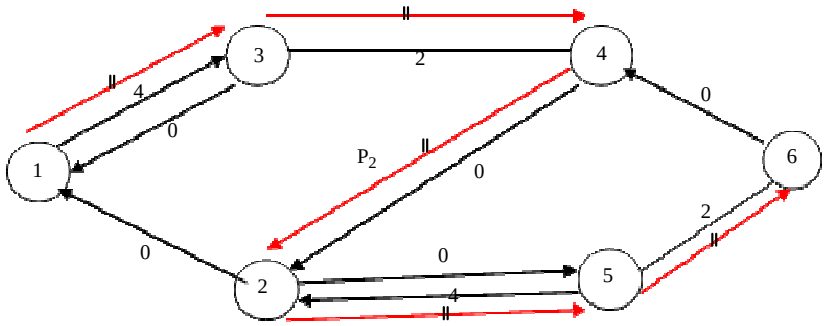
$C_{34}^{'} = C_{34} - d_4 + d_3 = 2-2+2 = 2$ $C_{43}^{'} = C_{43} - d_3 + d_4 = 2-2+2 = 2$

$C_{56}^{'} = C_{56} - d_6 + d_5 = 2-3+3 = 2$ $C_{65}^{'} = C_{65} - d_5 + d_6 = 2-3+3 = 2$

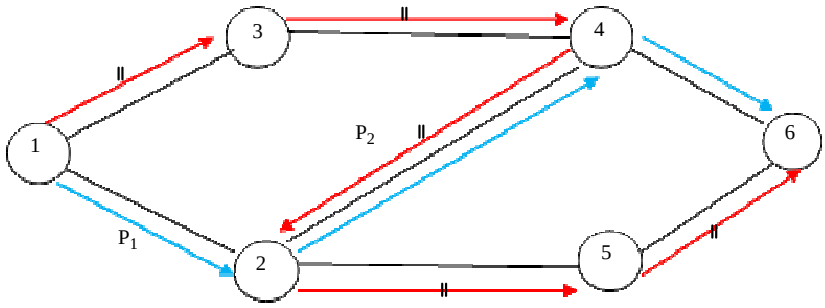
And the distance value of each arc of path p_1 is zero.



By applying Dijakstra's algorithm on G^1 we obtain the second shortest path p_2 (p_2 =1-3-4-2-5-6)



Now let us illustrate p_1 and p_2 in one graph.



Finally let us find the shortest paths by combining the edges of p_1 and p_2 and then discarding the common reversed edges between both paths (2-4), as a result we get the two shortest paths

$p_1 = 1-2-5-6$
$p_2 = 1-3-4-6$

REFERENCE:

[1] David Eppistein, finding the k shortest paths, tech, report 94-26, 1994

[2] D, Shier, computational experience with an algorithm for finding k shortest paths in a network, journal of the research of the NBS, 78:138-164, 1974

[3] D. Shier international methods of determining the k shortest paths in network, networks 6:151-159, 1976

[4] E.Q.V. MARTINS, M.M.B, Pascal, and J., L, E. Santos, A new shortest paths ranking algorithm research report, 1996

[5] E.Q.V. MARTINS, M.M.B, Pascal, and J., L, E. Santos .the k shortest paths problem, research report, 1998

[6] J, Y, Yen, finding the k shortest loop less paths in a network, management science, 17:712-716, 1971

[7] MARTA M, B PASCOAL, implementations and empirical comparison of k shortest loop less path algorithms, 2006

[8] Palmgren M. and Yuan D., 'A Short summary shortest path: Algorithms and applications.

[9] QIUJIN WU, JOANNA HARTILEY, using k shortest paths algorithm to accommodate user preference in the optimization of public transport.