Design and Analysis of Algorithm

Lecture- 13:
Greedy Algorithm

Contents

1. Single source shortest Path
2. Optimal Storage on tapes

Graph

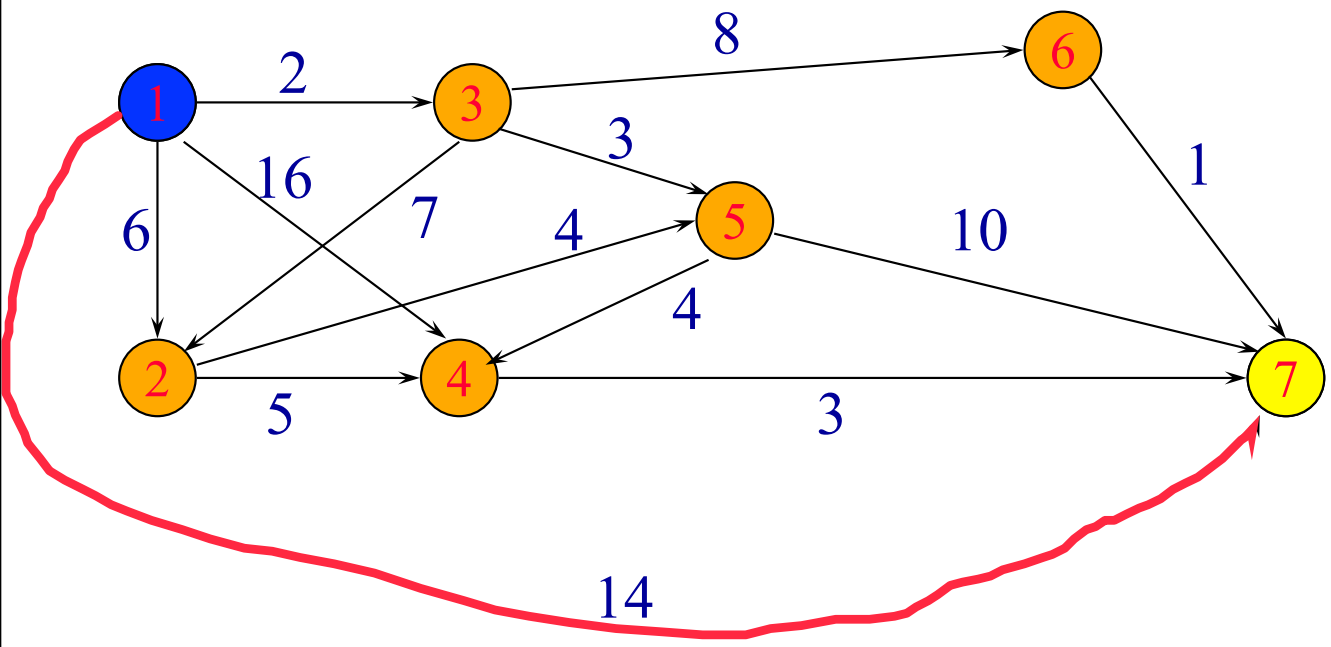
Graphs can be used to represent the connecting structure,

e.g. road of a state or country with vertices representing cities and edges representing sections of roads. The edges can then be assigned weights which may be either the distance between the two cities or the cost of moving from one city to another

Suppose you are to attend a party and for that your friend has invited you to the famous restaurant of city. Now you are interested in following answers

- If there is a path from your home to restaurant?
- If more than one path available ?
- Which is the shortest path?

Path between nodes

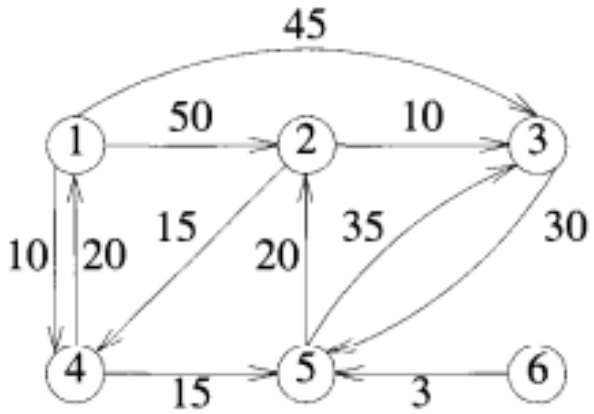


- Consider Source node 1 and destination node 7
- A direct path between Nodes 1 and 7 will cost 14
- A shorter path will cost only 11

Path construction

The greedy way to generate the shortest paths from v_0 to the remaining vertices is to generate the paths in non decreasing order of path length.

- First, a shortest path to the nearest vertex is generated.
- Then a shortest path to the second nearest vertex is generated , and so on



Path	Length
1,4	10
1,4,5	25
1,4,5,2	45
1,3	45

The length of a path is defined to be the sum of the weights of the edges on that path. The starting vertex of the path is referred to as the *source* ,and the last vertex the *destination*

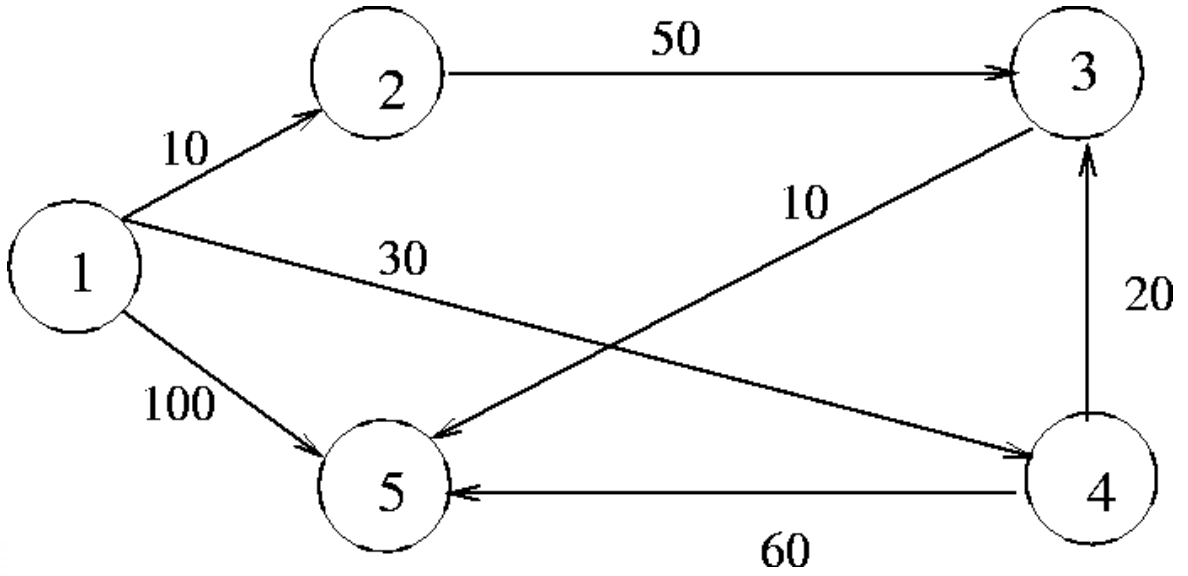
Algorithm

```
Algorithm ShortestPaths(v, cost, dist, n)
// dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n
// vertices. dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1 : n, 1 : n].
{
    for i := 1 to n do
    { // Initialize S.
        S[i] := false; dist[i] := cost[v, i];
    }
    S[v] := true; dist[v] := 0.0; // Put v in S.
    for num := 2 to n − 1 do
    {
        // Determine n − 1 paths from v.
        Choose u from among those vertices not
        in S such that dist[u] is minimum;
        S[u] := true; // Put u in S.
        for (each w adjacent to u with S[w] = false) do
            // Update distances.
            if (dist[w] > dist[u] + cost[u, w])) then
                dist[w] := dist[u] + cost[u, w];
    }
}
```

$S = \{\phi\}$

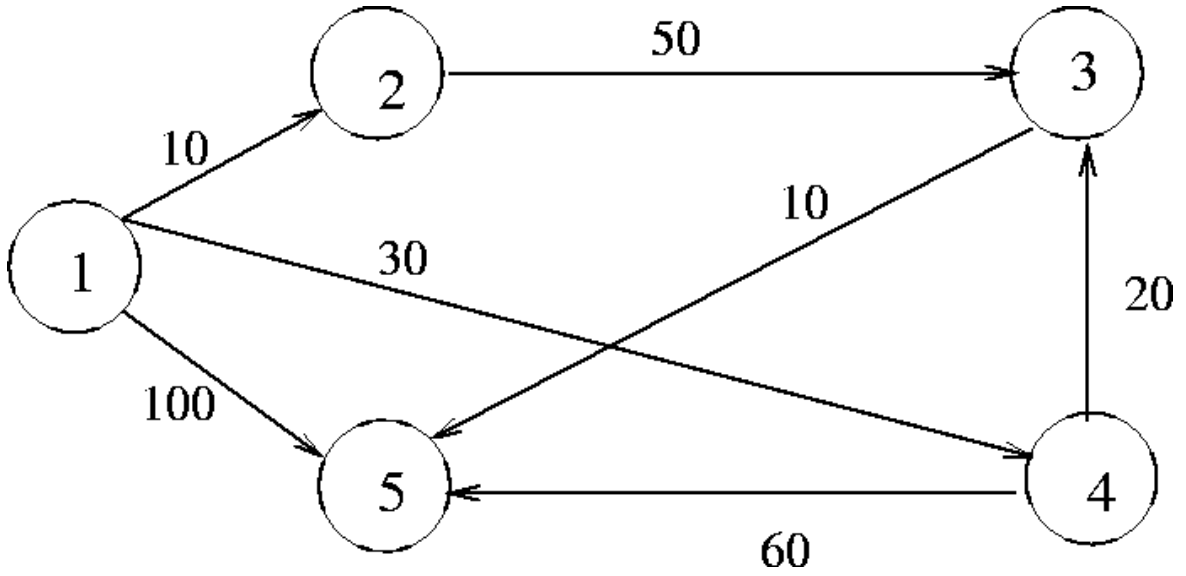
Algorithm

```
Algorithm ShortestPaths(v, cost, dist, n)
// dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n
// vertices. dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1 : n, 1 : n].
{
    for i := 1 to n do
    { // Initialize S.
        S[i] := false; dist[i] := cost[v, i];
    }
    S[v] := true; dist[v] := 0.0; // Put v in S.
    for num := 2 to n − 1 do
    {
        // Determine n − 1 paths from v.
        Choose u from among those vertices not
        in S such that dist[u] is minimum;
        S[u] := true; // Put u in S.
        for (each w adjacent to u with S[w] = false) do
            // Update distances.
            if (dist[w] > dist[u] + cost[u, w])) then
                dist[w] := dist[u] + cost[u, w];
    }
}
```



Initialization: $S = \{1\}$ $D(2) = 10, D(3) = \infty, D(4) = 30, D(5) = 100$

Algorithm



Select $w = 2$, so that $S = \{1, 2\}$

$$D[3] = \min(\infty, D[2] + C[2,3]) = 60$$
$$D[4] = \min(30, D[2] + C[2,4]) = 30$$
$$D[5] = \min(100, D[2] + C[2,5]) = 100$$

Select $w = 4$, so that $S = \{1, 2, 4\}$

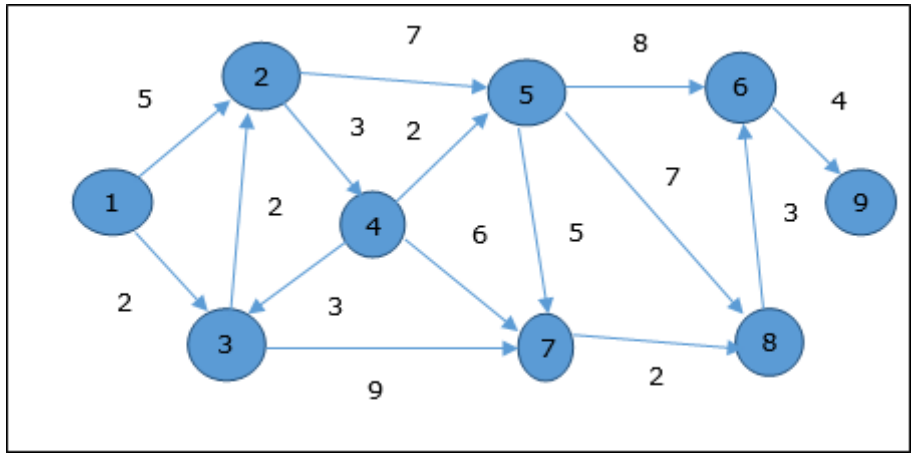
$$D[3] = \min(60, D[4] + C[4,3]) = 50$$
$$D[5] = \min(100, D[4] + C[4,5]) = 90$$

Select $w = 3$, so that $S = \{1, 2, 4, 3\}$

$$D[5] = \min(90, D[3] + C[3,5]) = 60$$

D[2]=10
D[4]=30
D[3]=50
D[5]=60

Example



Node	V_1	V_3	V_2	V_4	V_5	V_7	V_6	V_8
1	0	0	0	0	0	0	0	0
2	5	4	4	4	4	4	4	4
3	2	2	2	2	2	2	2	2
4	∞	∞	7	7	7	7	7	7
5	∞	∞	11	9	9	9	9	9
6	∞	∞	∞	∞	17	17	17	17
7	∞	11	11	11	11	11	11	11
8	∞	∞	∞	∞	16	13	13	13
9	∞	∞	∞	∞	∞	∞	20	20

Optimal Storage on Tapes

The problem:

There are n programs that are to be stored on a computer tape of length L. The lengths of these n programs are $l_1, l_2, \ldots, l_n$ respectively.

All programs can be stored on the tape if and only if the sum of the lengths of the programs is at most L.

We assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front.

Mean Retrieval Time

If the programs are stored in the order $I = i_1, i_2, i_3 \ldots \ldots, i_n$ the time t_j needed to retrieve program i_j is proportional to $\sum_{1 \le k \le j} l_{i_k}$

If all programs are retrieved equally often then the expected or mean retrieval time is $\frac{1}{n}\left(\sum_{1 \le j \le n} t_j\right)$

Objective:

We are required to find a permutation for the n programs so that when they are stored on the tape in this order the MRT is minimize

Minimizing MRT

Minimizing mean retrieval time means minimizing

$$d(I) = \sum_{1 \le k \le j} \sum_{1 \le j \le n} l_{i_k}$$

Example: Let n = 3 and $(l_1, l_2, l_3) = (5,10,3)$.

There are $n! = 6$ possible orderings.

Ordering	$d(I)$	Sum
1,2, 3	5+ (5+10) + (5+10+3)	38
1, 3,2	5+ (5+3) + (5+3+10)	31
2,3,1	10+ (10+3) + (10+3+5)	41
2,1,3	10+ (10+3) + (10+3+5)	41
3,1,2	3+ (3+5) + (3+5+10)	29
3,2,1	3+ (3+10) + (3+10+5)	34

Greedy Algorithm

In greedy approach to find optimal solution the approach is

To choose the next program on the basis of d value of the permutation constructed so far.

The greedy method simply requires us to store the programs in non-decreasing order of their lengths

The complexity of the approach is $O(n\ log\ n)$

Theorem

If $l_1 \leq l_2 \leq \cdots \ldots \ldots \ldots \leq l_n$, then the ordering $i_j = j$, $1 \leq j \leq n$ minimizes,

$$\sum_{k=1}^{n} \sum_{j=1}^{n} l_{i_j}$$

over all possible permutations of the i_j

Multiple Tapes

The tape storage problem can be extended to several tapes

If there are $m > 1$ tapes, $T_o, T_1 \ldots, T_{m-1}$, then the programs are to be distributed over these tapes

```
Algorithm Store(n, m)
// n is the number of programs and m the number of tapes.
{
    j := 0; // Next tape to store on
    for i := 1 to n do
    {
        write ("append program", i,
            "to permutation for tape", j);
        j := (j + 1) mod m;
    }
}
```

Example: No. of programs is 13, length of each programs to be stored on three different tapes 5, 2, 6, 8, 9, 12, 6, 14, 19, 20, 5, 2, 1.

1	5	6	12	20					
2	5	8	14						
2	6	9	19						