Shortest paths

Shortest path problems

- Let G=(X,U,v) with:
  - $X=\{x_0, x_1, x_2,\ldots, x_{n-1}\}$ et $v : U \rightarrow \Re$
- Length of a path: number of arcs composing the path
- Weight(value) of a path : sum of weights of its arcs
- Some path from x_i to x_k is of minimal weight if its weight is the smallest one (<= to all others paths from x_i to x_k.)
  - We call this the shortest path

Ps. all paths and cycles are assumed directed.

The shortest path problems

Three types of problems:

- Given two vertices x_i and x_k, find the shortest path (when such a path exists);
- Given a vertex x_s, find all shortest paths (if they exist) from x_s to any other vertex x_i;
- Find the shortest paths for all couples of vertices in the graph.

Applications

- Subproblem for numerous optimisation problems.
- Applications to transport:
  - Vehicle routing problem;
- Applications in telecommunications, ATM…
- *etc.*

Some properties of shortest paths (I)

- Lemma 1. Any subpath of a shortest path is as well a shortest path.

We assume below that there exists at least a path from x_0 to x_i for any i.

- Lemma 2. A necessary and sufficient condition such that, for any i, there exists a shortest path from x_0 to x_i is that graph G doesn't contain a negative cycle.

Some properties of shortest paths (II)

- Theorem 1. Let G be a graph without negative cycles and λ_i values of paths from x_0 to x_i. A necessary and sufficient condition such that $\{\lambda_i / 0 \leq i \leq n-1\}$ be the set of shortest paths values from x_0 is that:

  1- $\lambda_0 = 0$;

  2- $\lambda_j - \lambda_i \leq v_{ij}$, for all arc $(x_i, x_j) \in U$.

Proof (hint):

NC. If for some arc $(x_i, x_j) \in U$. $\lambda_j - \lambda_i > v_{ij}$, **we have a shorter path than λ_j to x_j.**

SC: Let μ be a shortest path to x_j, then we write down the eq. for all arcs composing it and sum on them, we obtain $\lambda_j \leq$ value(μ).

Corollary. The set of arcs (x_i, x_j) such that $\lambda_j - \lambda_i = v_{ij}$ is the set of arcs involved in shortest paths.

Shortest path algorithms

- FORD (or Bellman-Ford) algorithm:
  - Works for all weights given to arcs
  - O(n m)
  - Label correcting algorithm
- DIJKSTRA algorithm:
  - Works for all non negative weights given to arcs
  - $O(n^2)$
  - Label setting algorithm
- BELLMAN algorithm:
  - Works in acyclic graphs
  - O(m)
  - Label setting algorithm

FORD Algorithm

Algorithm:

(i) *Initialization*

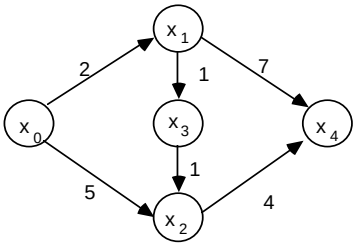
Poser $\lambda_0 = 0$ et $\lambda_i = +\infty$ pour $i > 0$.

(ii) *Edges examination*

for each vertex x_i, check all (x_i,x_j) from x_i and substitute λ_j with $\lambda_\iota + v_{ij}$ when $\lambda_i + v_{ij} < \lambda_j$.

(iii) *Stop Test*

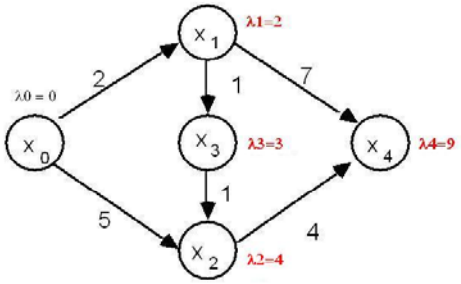
Iterate (ii) until some λ_j is updated in (ii).



An example

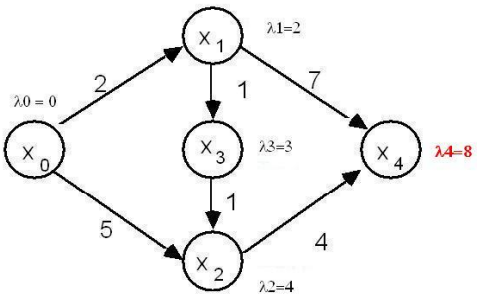
FORD Algorithm
an example

End of first iteration



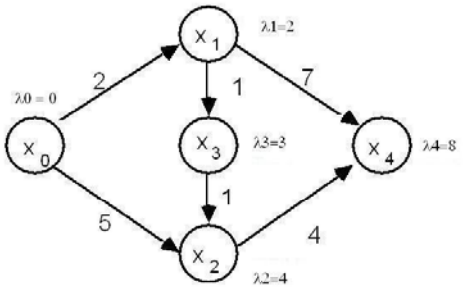
FORD Algorithm
an example

End of second iteration



FORD Algorithm
an example

Last iteration



Validity et complexity of Ford algorithm

Theorem 2: Ford algorithm computes values of the shortest path from x_0 when the graph is without negative circuits.

Proof by recurrence (hint):

- Set λ_i^{k*}, the min value of a path from x_0 to x_i containing at most k arcs.
- Set λ_i^k, the value λ_i after k steps in the loop while.

Invariant:

At the end of k^{th} step, λ_i^k gives the value of a path from x_0 to x_i s.t. $\lambda_i^k \leq \lambda_i^{k*}$.

Theorem 3: The complexity of Ford algorithm is in O(nm) where n = |X| and m = |U|.

DIJKSTRA Algorithm

Algorithm

(i) Set $S = \{x_0\}$, $\lambda_0 = 0$, $\lambda_i = v_{0i}$, if $(x_0, x_i) \in U$, and $\lambda_i = +\infty$, otherwise.

(ii) While $S \neq X$ do:

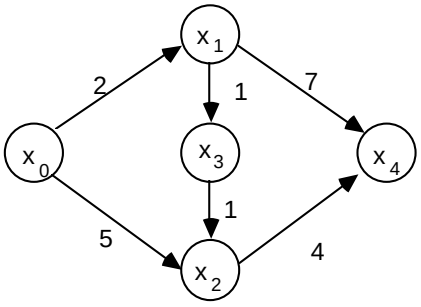
choose $x_i \in X - S$ of λ_i minimum.

set $S = S + \{ x_i \}$.

For any $x_j \in (X - S)$, successor of x_i,

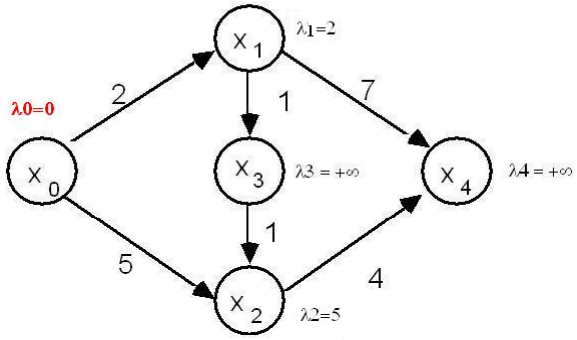
set: $\lambda_j = \min(\lambda_i + v_{ij}, \lambda_j)$.

DIJKSTRA Algorithm an example



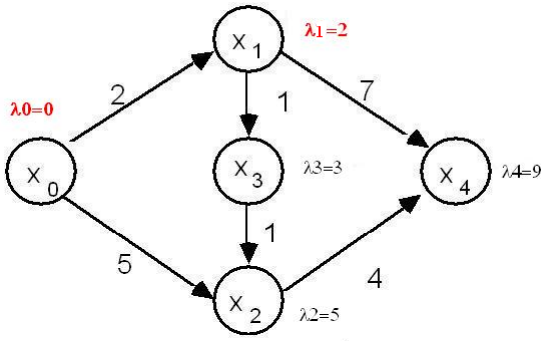
DIJKSTRA Algorithm
an example

End of the first iteration



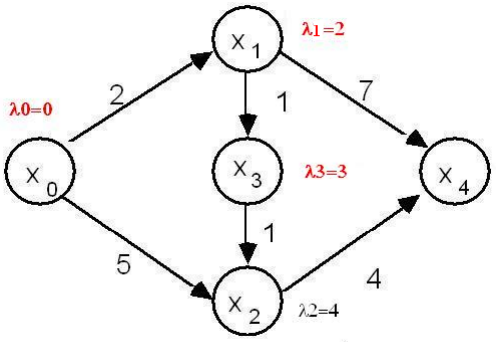
DIJKSTRA Algorithm
an example

End of the second iteration



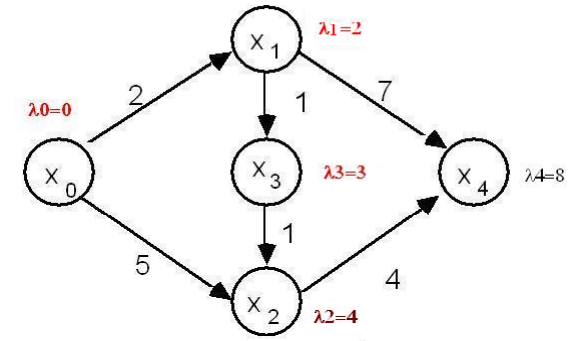
DIJKSTRA Algorithm
an example

End of the third iteration



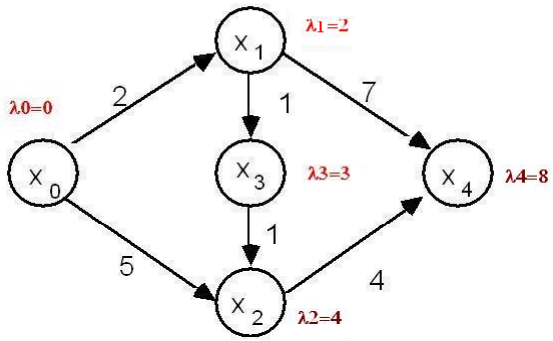
DIJKSTRA Algorithm
an example

End of the forth iteration



DIJKSTRA Algorithm
an example

End of the last iteration



Validity and complexity of Dijkstra algorithm

Theorem 4. λ_i obtained at the end of the algorithm are the shortest path values from x_0.

(valuations ≥ 0 : there are no negative cycles)

Proof by recurrence :

Invariant:

At the end of step **k**,

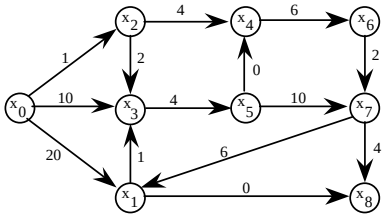
1- if $x_i \in S$: $\lambda_i = \lambda^*_i$.

2- if $x_i \notin S$: $\lambda_i = \min_{z \in U-(i) \cap S} \lambda_z + v_{zi}$

Lemma 3. Dijkstra algorithm is of complexity $O(n^2)$.

Exercise

We wish to find the values of the minimal paths from x0.



Apply DIJKSTRA algorithm. Write down the successive values of λi (if DIJKSTRA may be used), as well as a tree of minimal paths from x0.

Are the minimal paths unique? Justify your answer.

PROBLEM: SECOND SHORTEST (I)

Second shortest algorithm:

Begin

1) Apply the Dijkstra algorithm to obtain the tree A of the shortest paths from 0 to i and the potentials λ (i) from 0 to i for all the vertices i of G.

(Note: We shall note γ (r, s) the path, if it exists, from r to s in A)

2) Determine γ (0, n-1) = (y0 = 0, y1, ..., yp = n-1).

3) Set value: = +∞;

For j: = 1 to p do

for all k ∈ (U-(yj) - (yj-1)) do

Begin

α: = λ (k) + v (k, yj) + (λ(n-1) - λ(yj));

if α < value then

value: = α; pivot1: = k; pivot2: = yj;

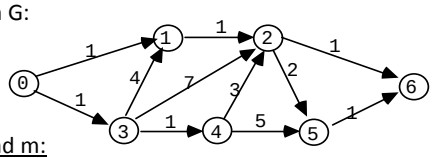
end;

4) **Second**: = γ (0, pivot1) + (pivot1, pivot2) + γ (pivot2, n-1).

end.

PROBLEM: SECOND SHORTEST (II)

1) Apply the algorithm to the following graph G:



2) Analysis of the complexity function of n and m:

It is assumed that the graph is coded by the queue of predecessors and successors.

2.1) What are the complexities of the phases 1, 2, 3 and 4 of the algorithm? Conclude as to the total complexity.

2.2) What improvements could be proposed to reduce this complexity?

3) Proof of the algorithm:

We note **first** γ = (0, n-1) = (y0 = 0, y1, ..., yp = n-1) the shortest path obtained in the phase 1) of the algorithm and **second** = (z0 = 0, z1, ..., zq = n-1) a second shortest path from 0 to n-1.

3.1) Show that there is an integer r such that yq = zp, yp-1 = 1 ,..., yp-r = zq-r and yp-r-1 ≠ zq-r-1.

3.2) What is the remarkable property of the path (0 = z0, z1, ..., zq-r-1) ?

3.3) Deduce the validity of the algorithm.

Bellman algorithm

Algorithm:

(i) **enumerate** all vertex of the graph, set $\lambda_0 = 0$.

(ii) for j = 1 to n – 1 set : $\lambda_j = \min (\lambda_k + v_{kj})$ over the set of predecessors x_k of x_j.

Ps. Vertex numeration is a function f : {V}->N s.t. for any arc $(x_i x_j)$ $f(x_j) < f(x_j)$.

Proof by recurrence…

Theorem 5: Bellman algorithm computes the shortest path values λ_i from x_0 in O(m).

Some path problems

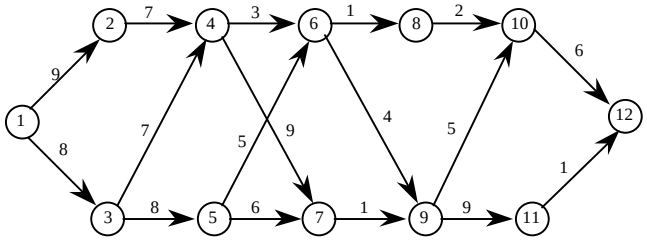
- The longest path computation problem;
- The maximum probability path;
- The maximum capacity path value;
  - Exercise : compute the shortest path among these of maximum capacity.

Exercise: The itinerary of Michel Strogoff

Leaving from Moscow, Michel STROGOFF, courier of the tsar, was supposed to reach IRKUTSK. Before leaving, he had consulted a fortune teller who told him, amongst other things : "After KAZAN beware of the sky, in OMSK beware of the tartars, in TOMSK beware of the eyes, after TOMSK beware of water and, above all, always be careful of a large brown-haired person with black boots. " STROGOFF had therefore written on a map his "chances" of success for each route between two towns : these chances were represented by a number between 1 and 10 (measuring the number of chances of success out of 10). Ignoring probability calculation, he had therefore chosen his route by maximising the total sum of the chances.

The numbers of the cities are: MOSCOW (1), KAZAN (2), PENZA(3), PERM (4), OUFA (5), TOBOLSK (6), NOVO-SAIMSK (7), TARA (8), OMSK (9), TOMSK (10), SEMIPALATINSK(11), IRKOUTSK (12).

Exercise: The itinerary of Michel Strogoff



1. Determine the route of Michel Strogoff.
2. What was the probability, with the assumption of the independence of the random variables, that Strogoff would succeed?
3. What would have been his route if he had known the principles of probability calculation?

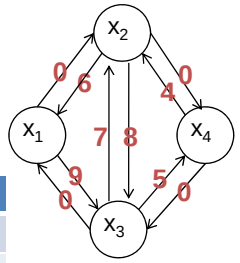
Matrix method (I)

```
for i←1 à n do {
    for j←1 à n do {
        if (j∈U⁺(i)) then V⁰[i][j] ← v_ij otherwise V⁰[i][j] ← ∞;
    }
}
for k←1 à n do {
    for i←1 à n do {
        for j←1 à n do {
            V^k[i][j] ← min(V^k-1[i][j] , V^k-1[i][k]+V^k-1[k][j])
        }
    }
}
```

Proof of validity of the algorithm by recurrence :
Hint : at the end of iteration **k**, $V^k[j][j]$ gives the value of the shortest path from **i** to **j** going through vertices **{1, 2, …, k}+{i, j}.**

Complexity : $O(n^3)$

Matrix method (II)




```
for i←1 à n do {
    for j←1 à n do {
            if (j∈U+(i)) then V0[i][j] ← vij
    otherwise V0[i][j] ← ∞;
    }
}
for k←1 à n do {
    for i←1 à n do {
            for j←1 à n do {
                    Vk[i][j] ← min(Vk-1[i][j] ,
    Vk-1[i][k]+Vk-1[k][j])
            }
    }
}
```

	1	2	3	4
1	0	0	9	∞
2	6	0	8	0
3	0	7	0	5
4	∞	4	0	0

	1	2	3	4
1	0	0	9	∞
2	6	0	8	0
3	0	0	0	5
4	∞	4	0	0

	1	2	3	4
1	0	0	8	0
2	6	0	8	0
3	0	0	0	0
4	10	4	0	0

	1	2	3	4
1	0	0	8	0
2	6	0	8	0
3	0	0	0	0
4	0	0	0	0

	1	2	3	4
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

Shortest path algorithms and applications to networks

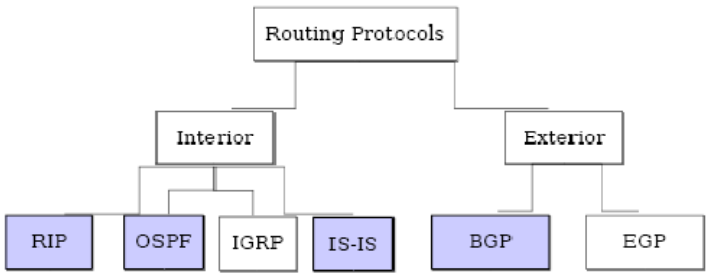
Routing protocols are implemented in a distributed way in IP networks.;

What is routing …

What is routing?

- The term **routing** corresponds to the mechanisms used by a host to transfer data to its destination by examining the information in the data.
- **Routing** is a key element of level **network** of TCP/IP stack. It uses information stocked in routing tables in each node-router.
  - The routing table stores the routes (and in some cases, metrics associated with those routes) to particular network destinations. It is frequent that in a routing table we find only the information about the gateway number toward the destination and not the entirely route.

Routing Protocols in Internet



Two main groups:
– Distance-Vector protocols: RIP, IGRP, BGP.
– Link-State protocols: OSPF, IS-IS

Distance-Vector routing protocols

A variant of Bellman-Ford algorithm:

```
boolean Bellman_Ford (G, s)
initialization (G, s) // all weights are set to +∝ except 0 for the origin node
    (0).
for i=1 to Number_of_nodes -1 do:
     for any edge (u, v) do:
          wx := weight(u) + weight(edge(u, v));
          if wx <weight(v) then
                    pred(v) := u;
                    weight(v) := wx;
for any edge (u, v) do:
     if weight(u) + weight(edge(u, v)) < weight(v) then
    return false;
return true.
```

Routing Information Protocol (RIP) (RFC 2453)

Let D(i,j) represent the metric of the best route from entity i to entity j. It should be defined for every pair of entities. d(i,j) represents the costs of the individual steps. Formally, let d(i,j) represent the cost of going directly from entity i to entity j. It is infinite if i and j are not immediate neighbors. Since costs are additive, it is easy to show that the best metric must be described by:

$D(i,i) = 0$, *all i*

$D(i,j) = \min_k [d(i,k) + D(k,j)]$, *otherwise*

and that the best routes start by going from i to those neighbors k for which d(i,k) + D(k,j) has the minimum value.

Implementing RIP

The **Routing Information Protocol** is a dynamic routing protocol used in local and wide area networks. As such it is classified as an interior gateway protocol (IGP) using the distance-vector routing algorithm.

Each router keeps a distance table for all destinations in the network. This table stores all shortest distance to any destination and the next neighbor to reach each of them according to the distance.

Periodically, each router announces its distance table to its direct neighbors;

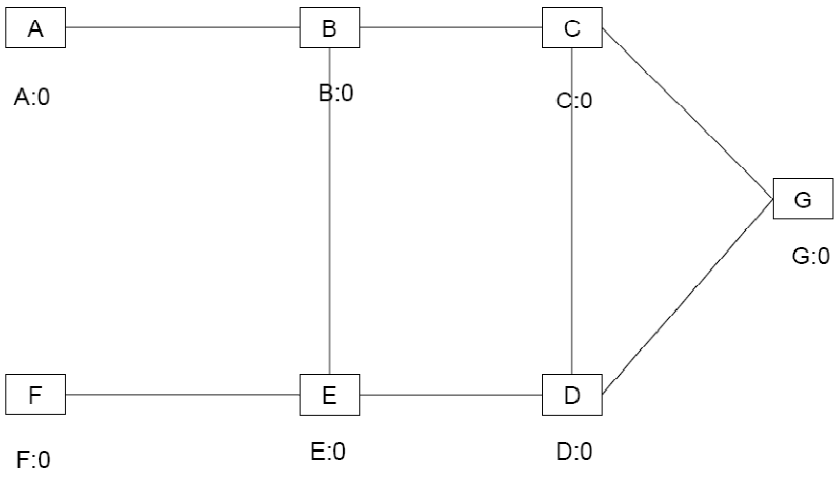
Any time some update is announced from a neighbor, do:

compute the new distance D';

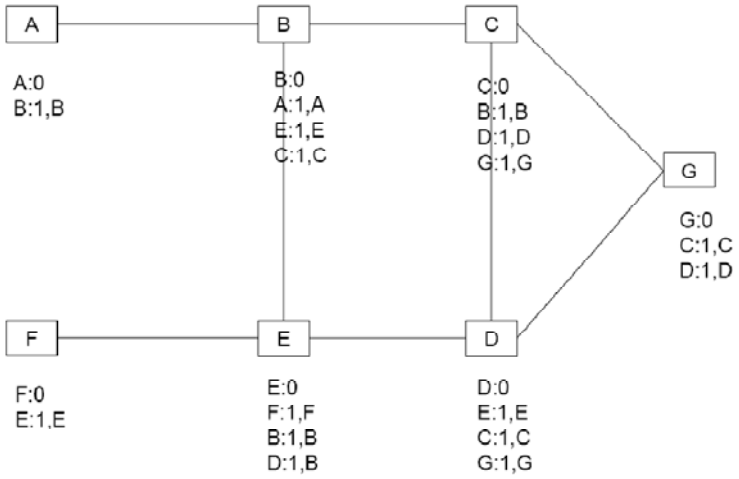
if D' < D keep the new value and the neighbor announcing it;

The update procedure is in origine of some limitations of the protocol…

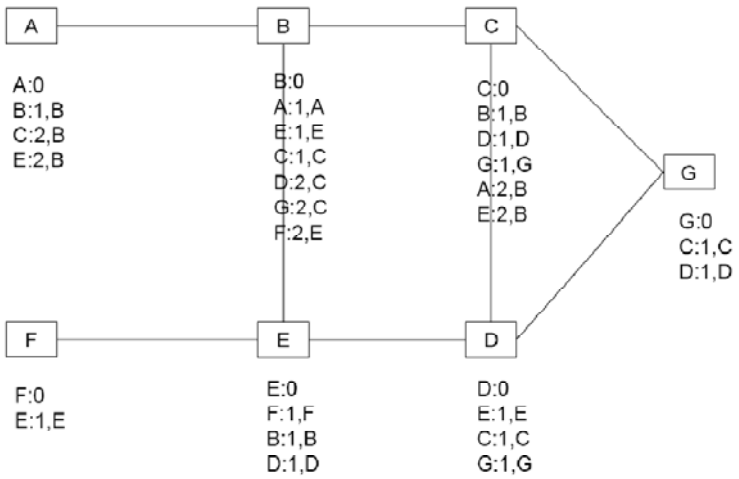
RIP: how it works?



RIP


A
B
C
G
F
E
D
A:0
B:1,B
B:0
A:1,A
E:1,E
C:1,C
C:0
B:1,B
D:1,D
G:1,G
G:0
C:1,C
D:1,D
F:0
E:1,E
E:0
F:1,F
B:1,B
D:1,B
D:0
E:1,E
C:1,C
G:1,G

RIP


A
B
C
G
F
E
D
A:0
B:1,B
C:2,B
E:2,B
B:0
A:1,A
E:1,E
C:1,C
D:2,C
G:2,C
F:2,E
C:0
B:1,B
D:1,D
G:1,G
A:2,B
E:2,B
G:0
C:1,C
D:1,D
F:0
E:1,E
E:0
F:1,F
B:1,B
D:1,D
D:0
E:1,E
C:1,C
G:1,G

Link-State protocols
OSPF (Open Shortest Path First)

- Principle:
  - **All nodes do have a map of the entire network**.
    - Determining the neighbors of each node
    - Distributing the information for the map (flooding)
    - Creating the map
  - **Computing the shortest paths**
    - Each node independently runs an algorithm (generally Dijkstra's algorithm is used) over the map to determine the shortest path from itself to every other node in the network.

END.