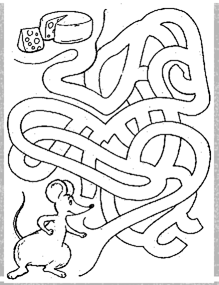
FIRST PART: CABLE NETWORKS

THE ROUTING PROBLEM I.E. THE SHORTEST PATH PROBLEM AND THE LEAST COST PATH PROBLEM

Prof. Tiziana Calamoneri

Network Algorithms

A.A. 2019/20



THE PROBLEM

(already studied!)

THE ROUTING PROBLEM (1)

Given a network:

- When packets are sent from a computer to another one through the network, each computer has to route data on a path passing through intermediate computers.
- This is the very general routing problem.

THE ROUTING PROBLEM (2)

- Case 1. Not adaptive routing

  A routing algorithm could try to send packets through a network so that the length of the used path is minimized. Such length can be measured in terms of number of hops between pairs of computers.

  If the network is modeled as a graph (node = computer and edge = link), the problem reduces to the shortest path problem between two nodes.

THE ROUTING PROBLEM (3)

- Case 2. Adaptive Routing

  It takes into account the traffic conditions: in order to decide next step, the traffic is estimated, so the packet is sent toward the zones of the network not affected by traffic.

  If the network is modeled by an edge-weighted graph (node = computer, edge = link, weight = dynamic value proportional to the traffic on the connection), the problem reduces to the (dynamic) least cost path problem.

THE ROUTING PROBLEM (4)

cases 1 and 2. Adaptive and non adaptive routings (cntd)

- Non adaptive routing:
  - Good results with consistent topology and traffic
  - Poor performance if traffic volume or topologies change over time
  - Information about the entire network has to be available
  - Each packet is routed through an outgoing edge in a fixed way
  - Routing tables are **used**

THE ROUTING PROBLEM (5)

cases 1 and 2. Adaptive and non adaptive routings (cntd)

- Adaptive routing:
  - Decisions are based on current network state
  - Packets follow dynamically computed routes
  - Routers are able to communicate
  - Rather often re-calculations are necessary
  - Each router **creates** its own routing table

THE ROUTING PROBLEM (6)

- Case 3. Routing with faults

  When the network is modeled as a graph, the length (edge-weight) of an edge may also represent the probability of its failing (used, for instance, in networks of thelephone lines, or broadcasting systems in computer networks or in transportation routes). In all these cases, one is looking for the route having the highest probability not to fail.

  More precisely…

THE ROUTING PROBLEM (7)

case 3. Routing with faults (cntd)

- Let $p(e)$ be the probability that edge e does not fail. Under the -not always realistic- assumption that failings of edges occur independently of each other, $p(e_1)\cdot p(e_2)\cdot\ldots\cdot p(e_k)$ gives the probability that the path $P=(e_1,e_2,\ldots,e_k)$ can be used without any faults.
- We want to maximize this probability over all possible paths with starting point a and arrival point b…

THE ROUTING PROBLEM (8)

case 3. Routing with faults (cntd)

- Note. Since function *log* is monotonic increasing, the maximum of the product $p(e_1)\cdot p(e_2)\cdot\ldots\cdot p(e_k)$ is reached iff the logarithm of the product is maximum, i.e. iff:

  $$\log p(e_1)+\log p(e_2)+\ldots+\log p(e_k) \text{ is maximum.}$$

- $\log p(e) \leq 0$ for each e because $p(e) \leq 1$.
- Define $w(e)=-\log p(e)$, then $w(e)\geq 0$ for all e; furthermore, we have to find a path from a to b for which

  $$w(e_1)+w(e_2)+\ldots+w(e_k) \text{ becomes minimum.}$$

- Thus, our problem is reduced again to the least cost path problem.

THE ROUTING PROBLEM (9)

Why does routing matter?

- End-to-end performance
  - Quality of the path affects user performance (propagation delay, throughput, packet loss)
- Use of network resources
  - Balance of the traffic over the routers and the links
  - Avoidance of congestion by directing traffic to lightly-loaded links
- Transition
  - The periodical changes of the routing tables reduce the incidence of faults and increase load balancing
  - Limiting packet loss and delay during changes

THE SHORTEST PATH PROBLEM AND THE LEAST COST PATH PROBLEM

SHORTEST PATHS (1)

- Let $G=(V,E)$ be a graph; let $w(e)$ be the length of each edge e.
- Many versions of the shortest path problem:
  - All to all
  - One to one
  - One to all
  - All to one
- Lengths can be:
  - All equal (unit length)
  - Non negative
  - Possibly negative but without negative cycles
  - Creating possible negative cycles

SHORTEST PATHS (2)

- Algorithm designed by Moore ['59] for the one-to-all shortest path problem and unit lenghts:

  … Breadth First Search (BFS) …

- TH. G is connected iff at the end of the BFS starting from node a, $dist(a,b) < \infty$ for each node b, where $dist$ is the distance in terms of number of edges.
- Note. This claim is false if G is a digraph (indeed the notion of connected graph does not exist on digraphs: strong and weak connection…)

LEAST COST PATHS (1)

Let $G=(V,E)$ be a graph or a digraph and let $w: E \rightarrow \mathbf{R}$ be an edge-weight function.

- (G,w) is called network.
- $w(e)$ is called length (though including meanings such as cost, capacity, weight, probability, …)
- For each path $P=(e_1, e_2, \ldots, e_k)$ (if G is a digraph, P is a dipath), the length of P is defined as $w(P)=w(e_1)+w(e_2)+ \ldots +w(e_k)$.
- Note. If $w(e)=1$ for each edge, the least cost path problem reduces to the shortest path problem.

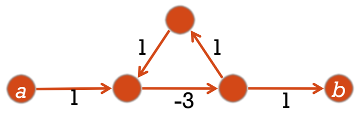
LEAST COST PATHS (2)

Given two nodes *a* and *b*, the distance *d(a,b)* is defined as the minimum, over all the paths *P* connecting *a* and *b*, of *w(P)*.

Two problems arise:

- PR.1: *b* could be unreachable from *a*
- SOL.: define $d(a,b)=\infty$ if *b* is unreachable from *a*

- PR.2: the minimum could not exist (cycles of negative length)
- SOL.: only networks without cycles of negative length are feasible



LEAST COST PATHS (3)

Negative lengths may occur!

Example:

- A ship travels from port *a* to port *b*, where the route (and possible intermediary ports) may be chosen freely.
- The length *w(x,y)* signifies the profit gained by going from *x* to *y*.
- For some edges, the ship might have to travel empty so that *w(e)* is negative for these edges: the profit is actually a loss.
- Replacing *w(e)* by –*w(e)* for all *e* in this network, the shortest path represents the route which yields the largest possible profit.

LEAST COST PATHS (4)

- In general, when *w* represents a gain, it seems natural to replace *w(e)* by –*w(e)* and look for least cost paths, but this could introduce cycles of negative weigth.
- There exist good algorithms that find minimum weight paths even when *G* contains cycles of negative weight.

LEAST COST PATHS (5)

OBSERVATIONS CONCERNING THE SOLUTION

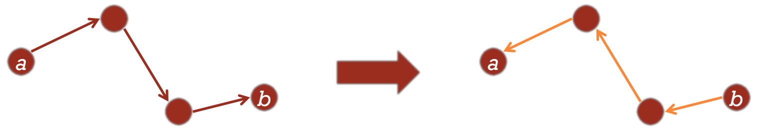
In any solution:

- Cycles having negative length cannot exist (we avoided them by hypthesis)
- Cycles having positive length cannot exist (by contradiction: if one of them is in the solution, the new solution without it has a lower cost)
- Cycles having null length do not exist without loss of generality: if one of them is in the solution, the new solution without it has the same cost and so is feasible, too
- So: our solution does not contain any cycles and hence it passes through at most *n-1* edges.

LEAST COST PATHS (6)

OBSERVATIONS CONCERNING THE SOLUTION (CNT.D)

- In order to univocally determine a path from *a* to *b* it is enough, for each node in such path, starting from *b* and coming back, to store its predecessor on the path.
- To do it: for each node *v* in *G* define a pointer *pt(v)*, initially equal to NULL; at the end, it points at the predecessor of *v* on the path.



SOME CLASSICAL ALGORITHMS

(already studied!)

BELLMAN-FORD ALGORITHM ['58]

- *G=(V,E)* directed with edge-weights possibly negative
- It solves the problem of the shortest path from single source, hence it outputs the distances from the (single) source to each node
- It assumes that *G* does <u>not</u> contain any <u>cycles of negative length</u>
- It is based on the principle of <u>relaxation</u>
- Time complexity: $O(nm)$

DIJKSTRA ALGORITHM ['59]

- *G=(V,E)* directed with <u>non negative</u> edge-weights
- It solves the problem of the shortest path from single source, hence it outputs the distances from the (single) source to each node
- It is based on the principle of <u>relaxation</u>
- Time complexity: either $O(n^2)$ or $O(m \log n)$
- The time complexity of the Dijkstra Algorithm is better than the time complexity of the Bellman-Ford Algorithm, but it is less versatile, as it requires not negative edge weight edges.

FLOYD-WARSHALL ALGORITHM ['62]

- $G=(V,E)$ directed with edge-weights possibly negative
- It solves the problem of the all pairs shortest path, hence it outputs a matrix with the distances from each node to each other node
- Repeatedly applying the algs treated before, varying the source over all nodes in V :
  - Bellman-Ford: $n\ O(nm)=O(n^2m)$
  - Dijkstra: $n\ O(n^2)=O(n^3)$ o $n\ O(m \log n)=O(mn \log n)$
- Time complexity: $O(n^3)$ and negative edge weights are allowed

THE RELAXATION

- For each node v, let $d(v)$ be a function representing an estimate of the weight of the shortest path from s to v.
- At the beginning $d(v)=\infty$ for each v
- One relaxation step is performed as follows:
  - Given an edge (u,v)
  - If $d(u)+w(u,v)<d(v)$

    $d(v)=d(u)+w(u,w)$

    $pt(v)=u$
- Time complexity of one relaxation step: $O(1)$

BELLMAN-FORD ALGORITHM (1)

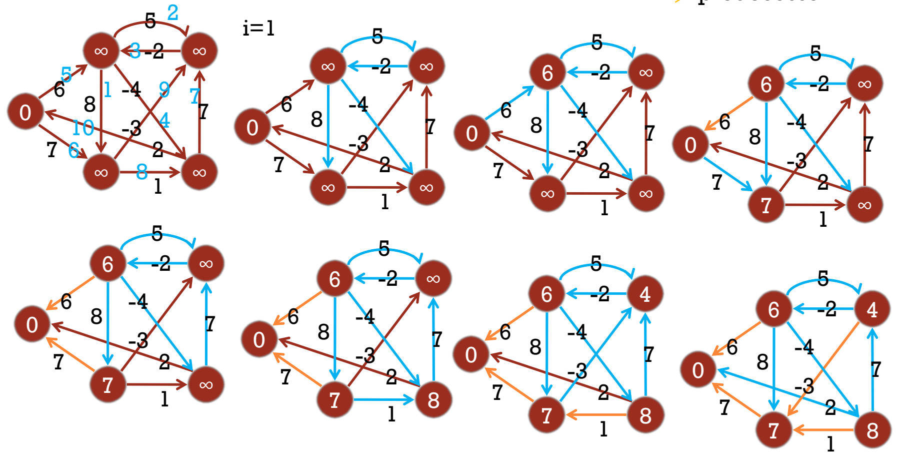
- Assume that G does not contain any cycles of negative length

```
For each v initialize d(v) and p(v)          O(n)+
For i=1 to n-1 do                            |n-1 times
   For each (u,v) relax v w.r.t. (u,v)         | mO(1)
```

- Time Complexity: $O(nm)$

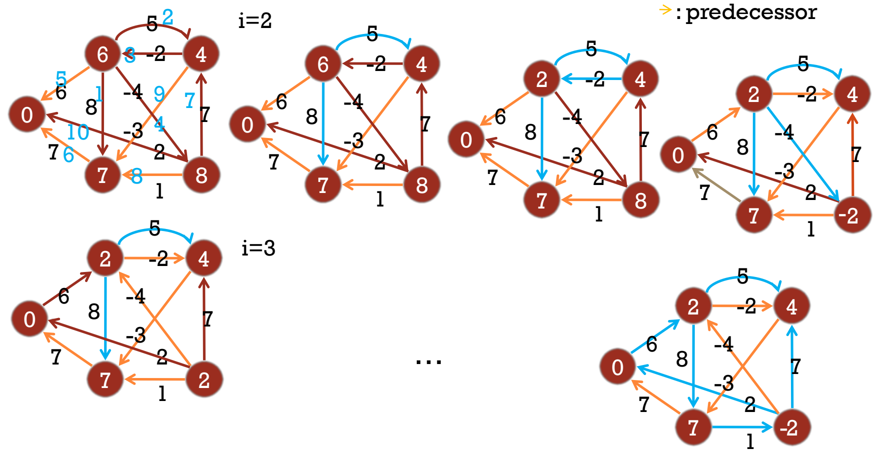
BELLMAN-FORD ALGORITHM (2)

x: order of the edges
→: visited edges
→: predecessor

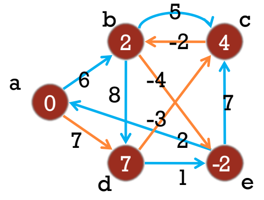


BELLMAN-FORD ALGORITHM (3)

x: order of the edges
→: visited edges
→: predecessor



BELLMAN-FORD ALGORITHM (4)



(Let us consider predecessors in the same direction than original edges)

Forwarding table of *a*:

b	*(a,d)* as the shortest path is *a-d-c-b*
c	*(a,d)* as the shortest path is *a-d-c*
d	*(a,d)*
e	*(a,d)* as the shortest path is *a-d-c-b-e*

BELLMAN-FORD ALGORITHM (5)

Bellman-Ford Algorithm is used for *Distance Vector routing*, an iterative, asynchronous and distributed protocol:

- $c(x,v)$=cost for (directed) link from x to v
- $D_x(y)$=estimate of least cost from x to y; x mantains distance vector $D_x=[D_x(y)$ for each y neighbor of $x]$; for each neighbor y, x mantains also D_y
- Each node x periodically sends D_x to its neighbors
- Neighbors update their own distance vector: $D_x(y)=min\{c(x,v)+D_v(y)\}$
- x notifies neighbors when its distance vector changes
- Over the time, D_x converges

- An exemple: Routing Information Protocol (RIP)

DIJKSTRA ALGORITHM (1)

- $G=(V,E)$ directed with non negative edge-weights
- It partitions the nodes of G: nodes whose shortest path from s has already been found (S) and all the other nodes (V-S)
- Greedy algorithm

- At each step, let u be the node in V-S with minimum value of d; add u to S and relax all the edges outcoming from u
- Keep V-S in a priority queue (e.g. min heap)

DIJKSTRA ALGORITHM (2)

- For each v initialize $d(v)$ and $pt(v)$
- S=empty set
- $Q=V$
- While Q is not empty
  - u=ExtractMin(Q)
  - $S=S \cup \{u\}$
  - For each edge (u,v) outcoming from u
    relax v w.r.t. (u,v)
    Update Q

The time complexity depends on the data structure used to implement Q:

Queue: $O(n^2)$
Heap: $O(m \log n)$
Fibonacci Heap: $O(m+n \log n)$

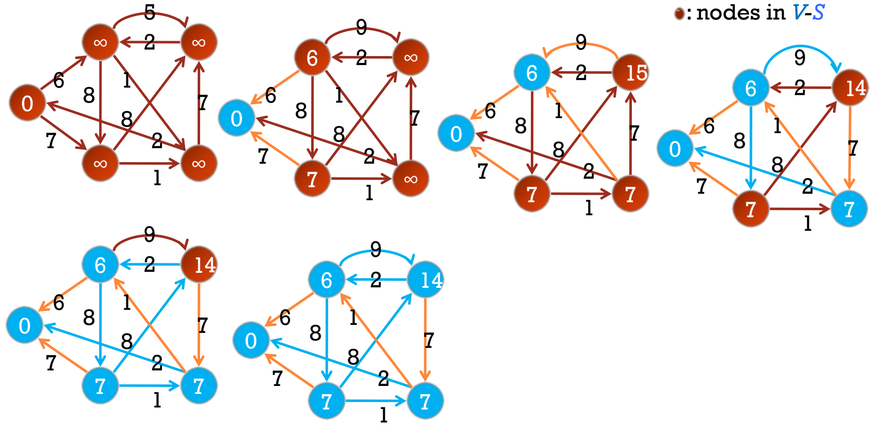
DIJKSTRA ALGORITHM (3)

- For each v initialize $d(v)$ and $pt(v)$
- S=empty set
- $Q=V$
- While Q is not empty
  - u=ExtractMin(Q)
  - $S=S \cup \{u\}$
  - For each edge (u,v) outcoming from u
    relax v w.r.t. (u,v)
    Update Q

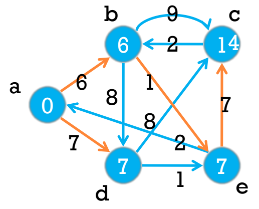
Using heap:
$O(n)$
$O(1)$
$O(n)$
n times
| $O(\log n)$
| $O(1)$
| $O(\deg u)$ times
| | $O(1)$
| | $O(\log n)$
tot.
$O(n \log n + m \log n) = O(m \log n)$

DIJKSTRA ALGORITHM (4)

→: visited edges
→: predecessor
●: nodes in S
●: nodes in V-S



DIJKSTRA ALGORITHM (5)



(Let us consider drawings of predecessors in the same direction than original edges)

Forwarding table of a:

b	(a,b)
c	(a,b) as the shortest path is a-b-e-c
d	(a,d)
e	(a,b) as the shortest path is a-b-e

DIJKSTRA ALGORITHM (6)

Dijkstra Algorithm is used for dynamic routing protocols.

- Each router:
  - Keeps trace of its incident links
    - Whether the link is up or down
    - The cost on the link (varying in time)
  - Broadcasts the link state (flooding)
    - So, every router has a complete view of the graph
  - Runs Dijkstra Algorithm
    - To compute the shortest paths…
    - …and construct the forwarding table
- An exemple: Open Shortest Path First (OSPF) used in the networks with Internet Protocol (IP)

FLOYD-WARSHALL ALGORITHM (1)

Sometimes, it is not enough to calculate the distances w.r.t. a certain node *s*: we need to know the distances between all pairs of nodes.

- $G=(V,E)$ directed with any edge-weight.
- Algorithm for the all-to-all shortest path problem
- **Trick 1**: all edges are in; the non-existing ones have $w=\infty$
- **Trick 2**: in order to go from *i* to *j* you can either go directly or passing through a third node *k*
- dynamic programming

FLOYD-WARSHALL ALGORITHM (2)

Algorithm:

```
For each node $i$, initialize $dist(i,i)=0$                      $O(n)$
For each edge $(i,j)$ initialize $dist(i,j)=w(i,j)$              $O(n^2)$
For each node $k$                                                $n$ times
    For each node $i$                                            |$n$ times
        For each node $j$                                        ||$n$ times
            $dist(i,j)=min\{dist(i,j), dist(i,k)+dist(k,j)\}$    |||$O(1)$
```

- Time Complexity $O(n^3)$

ANOTHER APPLICATION (1)

Difference Constraints:

- Let be given some tasks with precedence constraints and running lengths, and an unlimited (or limited by n=number of tasks) number of processors:
  - Each task *i* has:
    - Starting time s_i
    - Time to complete $b_i>0$
    - Constraint $s_j+b_j\leq s_i$ if task *i* can be started after that task *j* has been completed
  - First task can start at time 0
  - When can we finish last task?

ANOTHER APPLICATION (2)

This is the Shortest Path Problem on directed acyclic graphs:

- Define a graph having a node for each task
- Insert a dummy node v_0 (that models time 0)
- Insert an arc (v_0,i) for each task node i and let 0 be its weight
- For each precedence constraint $s_j+b_j\leq s_i$ insert an arc (j,i) with weight b_j
- Optimal Solution: start each task i at time equal to the length L_i of the longest path from v_0 to i. All the tasks are surely completed within time

$$max_i\ (L_i+b_i)$$

ANOTHER APPLICATION (3)

To transform to the shortest path problem, multiply all lengths and times by -1:

$b_1=2$

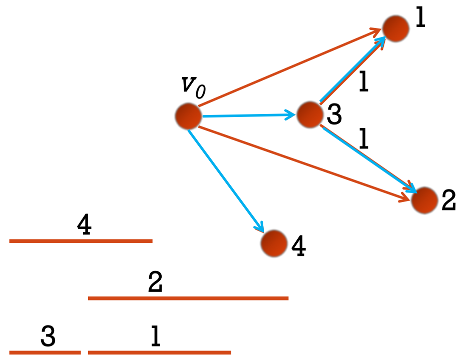
$b_2=3$

$b_3=1$

$b_4=2$

$s_3+b_3\leq s_2$

$s_3+b_3\leq s_1$



v_0
1
1
3
1
2
4
4
2
3
1