Part II: Applications of Network Models

Lecture 1: Shortest Path Problem

Junlong Zhang

zhangjunlong@tsinghua.edu.cn

- Lecturer
  - Junlong Zhang, Assistant Professor, Dept. of Industrial Engineering
  - Email: zhangjunlong@tsinghua.edu.cn
  - Office: South 603, Shunde Building
  - Office hour: 14:00-15:00 Wednesday
  - Research interests: stochastic integer programming, bilevel programming, logistics
- Grading
  - Homework: 30% (*late submission will not be accepted*)
  - Final exam: 70% (*16th week, June 9, in class*)

Introduction

- Presence of a transportation network
  - travel is restricted to take place along the highways, avenues, streets, rail links, waterways, …
- Network-based models and their applications
  - Urban travel distances and travel times
  - Vehicle-routing and collection and distribution
  - Site selection for the location of facilities
- Methodology: graph theory

In this class

- Objective: learn the model formulation and solution algorithms of the *shortest path problem*.
- Content
  - 1. Notation
  - 2. Model formulation
  - 3. Solution algorithm
    - 3.1 Dijkstra algorithm
    - 3.2 Bellman-Ford algorithm
    - 3.3 Floyd algorithm
    - 3.4 Complexity

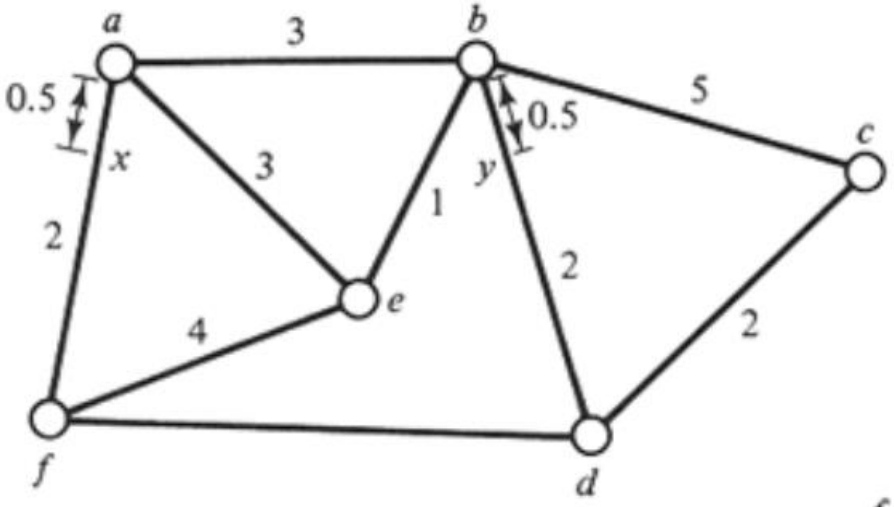
1. Some definition and notation

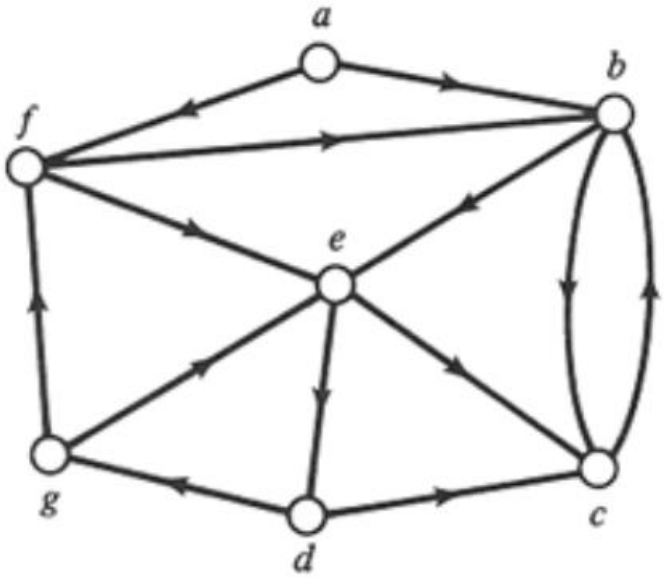
- N: set of nodes (vertices), $N = \{1, \ldots, n\}$
- A: set of edges (arcs, links), $A = \{(i, j) \mid i, j \in N\}$
- Network (graph): $G(N, A)$
- Directed (oriented) graph: every edge has a specified orientation
- Degree of a node
  - Undirected graph: number of edges incident on it
  - Directed graph:
    - Indegree: number of edges leading into that node
    - Outdegree: number of edges leading away from the node

1. Some definition and notation

- Path: a sequence of adjacent edges and nodes
- Simple path: each edge appears only once
- Elementary path: each node appears only once
- Cycle: a path whose initial and final nodes coincide
- Connected undirected graph: a path exists between every pair of nodes
- Strongly connected directed graph: a path exists between all ordered pairs of nodes
- Subgraph $G'(N', A')$ of $G(N, A)$: $N' \subset N$, $A' \subset A$

1. Some definition and notation





1. Some definition and notation

- Shortest path problem
  - Given a graph $G(N, A)$ and arc cost c_{ij} for $(i, j) \in A$, find a minimum-cost path between two nodes s and t.
- Elementary shortest path problem
  - Find a minimum-cost path between two nodes s and t such that each node of G is visited at most once (that is, the path does not contain subtours).
- When the costs c_{ij} induce no negative cycles on G, the shortest path problem can be solved efficiently via ad-hoc polynomial time algorithms.
- When there are negative cycles on G, subtours must be explicitly prevented, and an elementary shortest path problem must be solved.

2. Model formulation

- Consider a directed graph $G(N, A)$, denote by $\delta^+(i)$ and $\delta^-(i)$ the set of outgoing and incoming arcs of node i.

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$\sum_{(i,j)\in\delta^+(i)} x_{ij} - \sum_{(j,i)\in\delta^-(i)} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{else} \end{cases} \qquad \forall i \in V \tag{2}$$

$$\sum_{(i,j)\in\delta^+(i)} x_{ij} \leq 1 \qquad \forall i \in V \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A, \tag{4}$$

2. Model formulation

- When there are negative cycles on G, constraints (2)-(4) are not sufficient to guarantee that the solution is an elementary path. Why?
- In this case, subtour elimination constraints must be added, which will be introduced in following classes.

3. Solution algorithm

- Shortest path from a given node to all other nodes
  - 3.1 Dijkstra algorithm
  - 3.2 Bellman-Ford algorithm
- Shortest paths between all pairs of nodes
  - 3.3 Floyd algorithm
  - 3.4 Complexity

3.1 Dijkstra algorithm

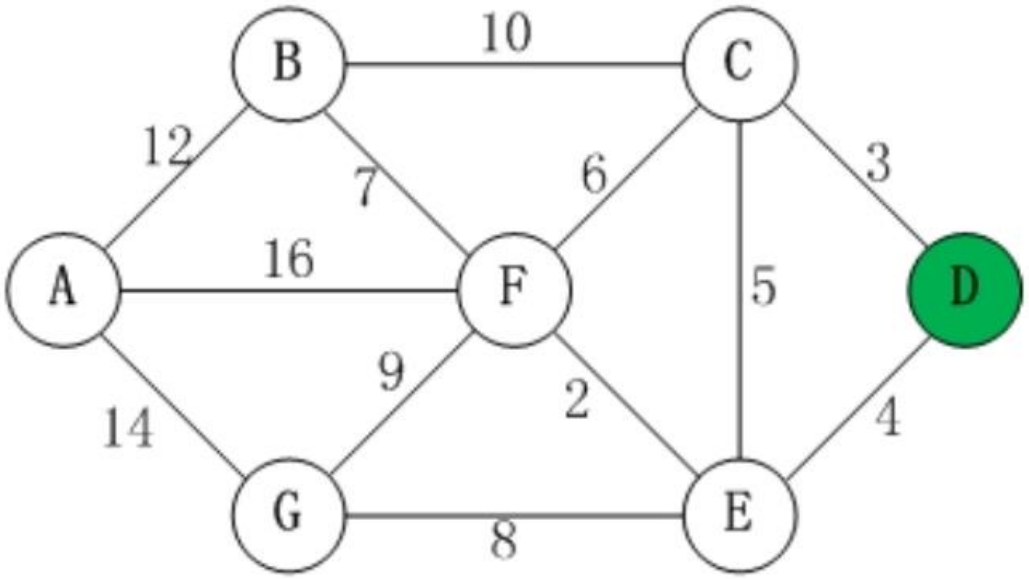
- Objective: to find the shortest paths from a source node s to all other nodes
- Length of link (i,j): $l(i,j)$
- Assumption: all link lengths are nonnegative
- Main idea: successively find its closest, second closest, third closest, and so on, node, one at a time, until all nodes in the network have been exhausted
- Two states for each node: open and closed
- Two label entries for each node:
  - $d(j)$: length of the shortest path from s to j discovered so far
  - $p(j)$: immediate predecessor node to j in the shortest path from s to j discovered so far

3.1 Dijkstra algorithm

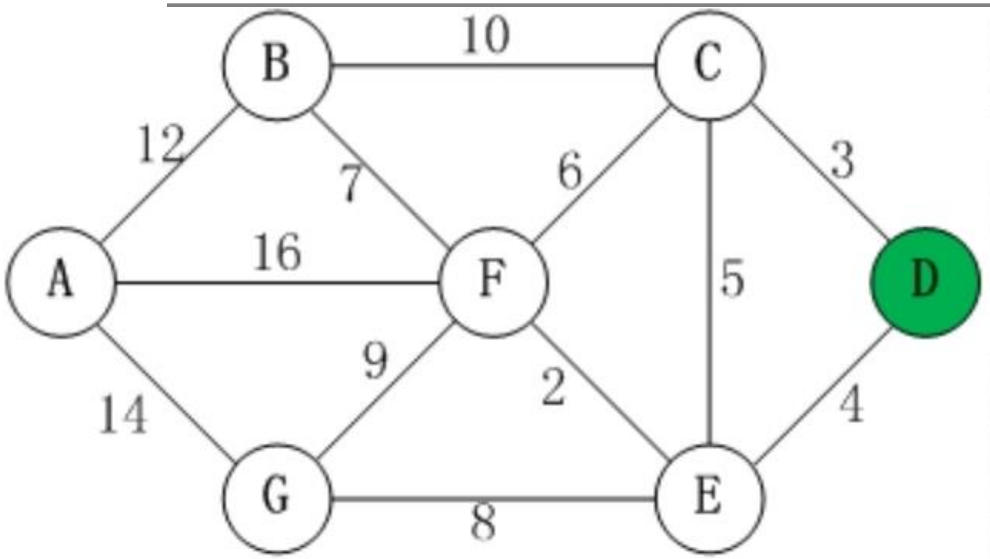
- Step 1: Initialize $d(s) = 0$, $p(s) = *$, s closed; $d(j) = \infty$, $p(j) = -$, j open, for $j \neq s$; set $k = s$ (i.e., s is the last closed node).
- Step 2: Examine all edges (k, j) with j open and update $d(j) = \min\{d(j), d(k) + l(k, j)\}$.
- Step 3: Among all open nodes, choose node i with the smallest $d()$ as the next node to close.
- Step 4: Find one closed node j^* such that $d(i) = d(j^*) + l(j^*, i)$, and set $p(i) = j^*$.
- Step 5: Set node i as closed. If all nodes are closed, stop; otherwise set $k = i$ and return to Step 2.

Example

- Find shortest path from node D to all other nodes



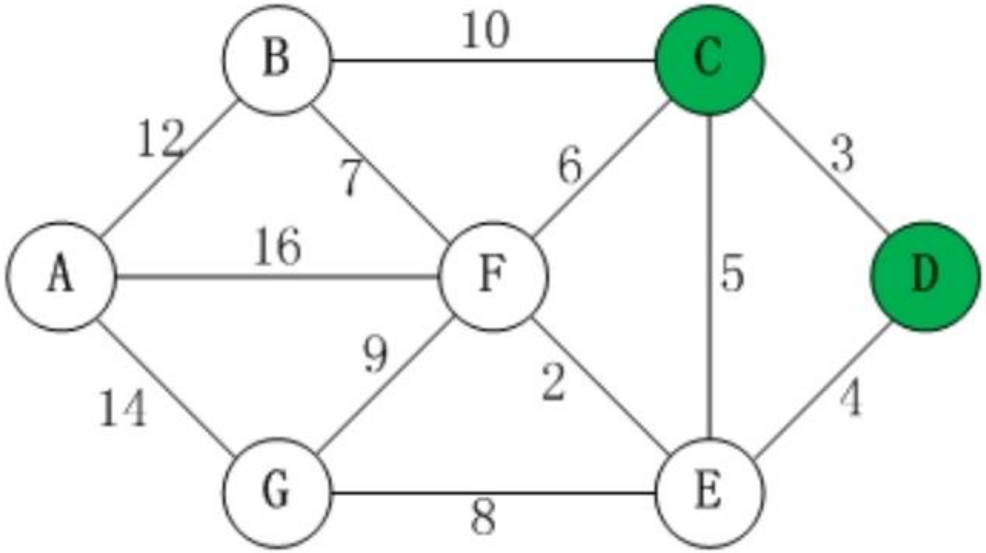
Example



第1步：
选取顶点D

S={D(0)}
U={A(∞),B(∞),C(3),E(4),F(∞),G(∞)}

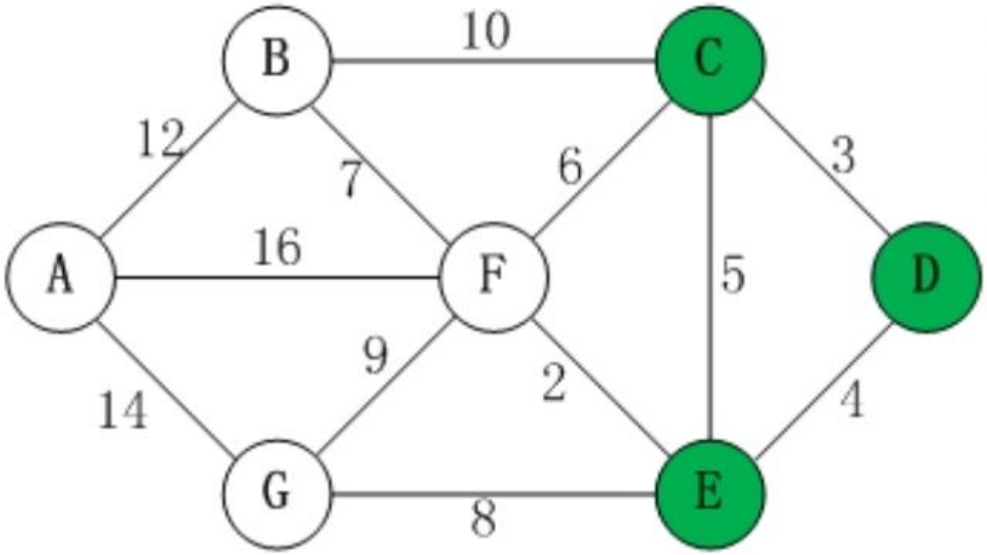
注：
(01) S是已计算出最短路径的定点的集合
(02) U是未计算出最短路径的定点的集合
(03) C(3)表示顶点C到起点D的最短距离是3



第2步：
选取顶点C

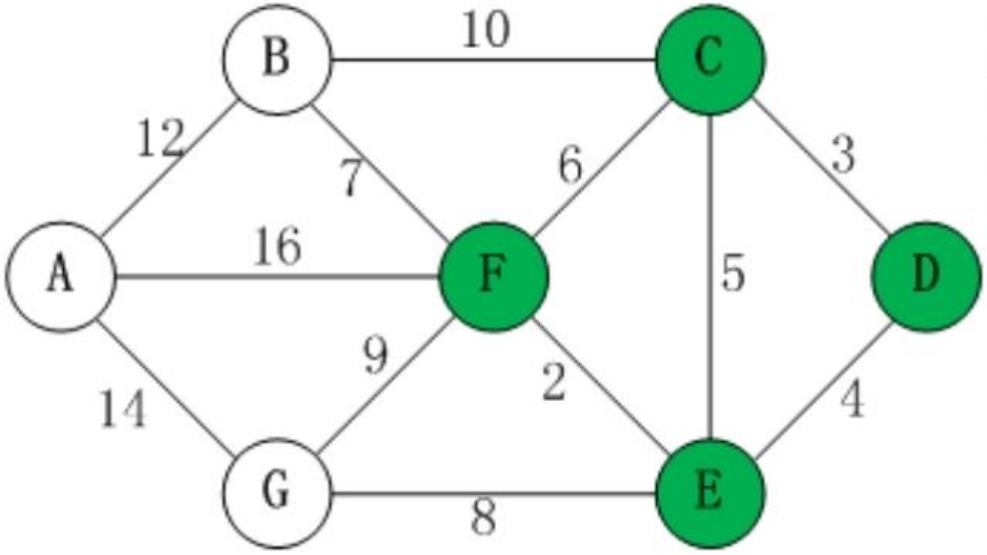
S={D(0),C(3)}
U={A(∞),B(23),E(4),F(9),G(∞)}

Example



第3步：
选取顶点E

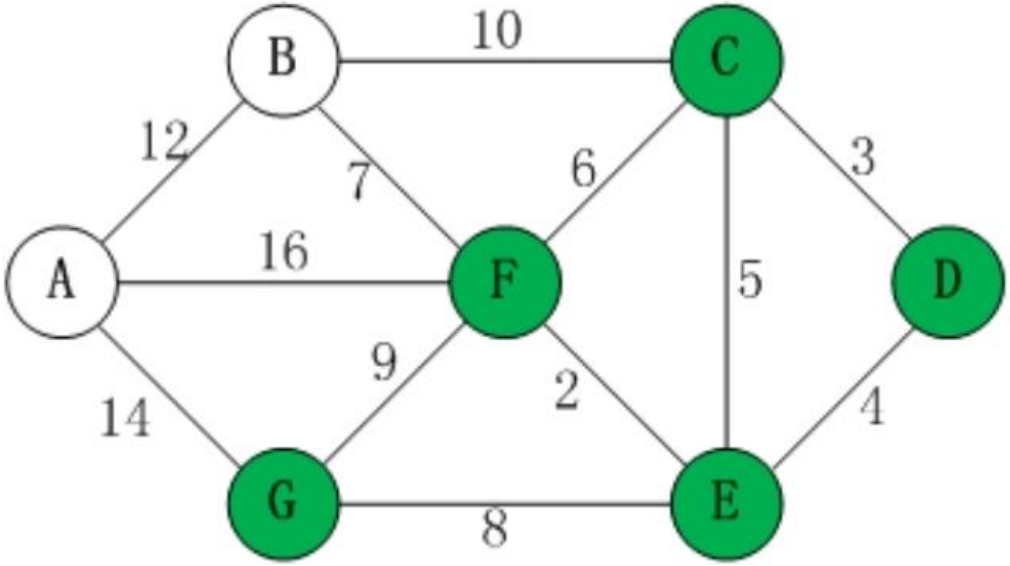
S={D(0),C(3),E(4)}
U={A(∞),B(23),F(6),G(12)}



第4步：
选取顶点F

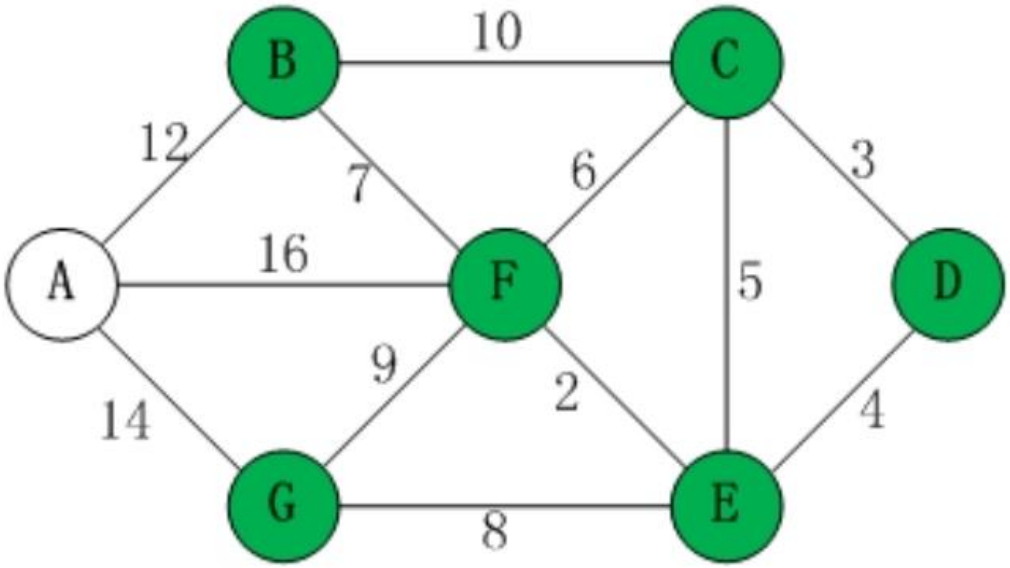
S={D(0),C(3),E(4),F(6)}
U={A(22),B(13),G(12)}

Example



第5步：
选取顶点G

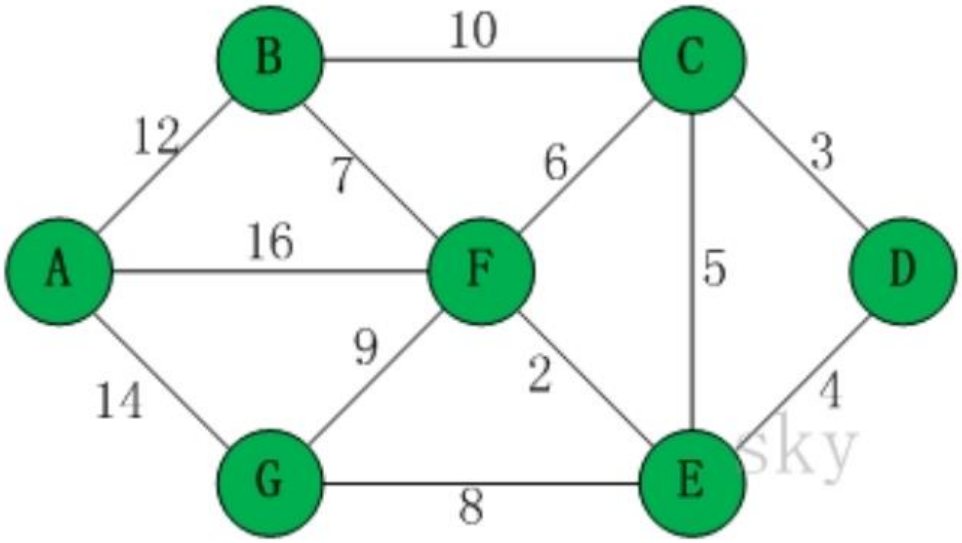
S={D(0),C(3),E(4),F(6),G(12)}
U={A(22),B(13)}



第6步：
选取顶点B

S={D(0),C(3),E(4),F(6),G(12),B(13)}
U={A(22)}

Example



第7步：
选取顶点A

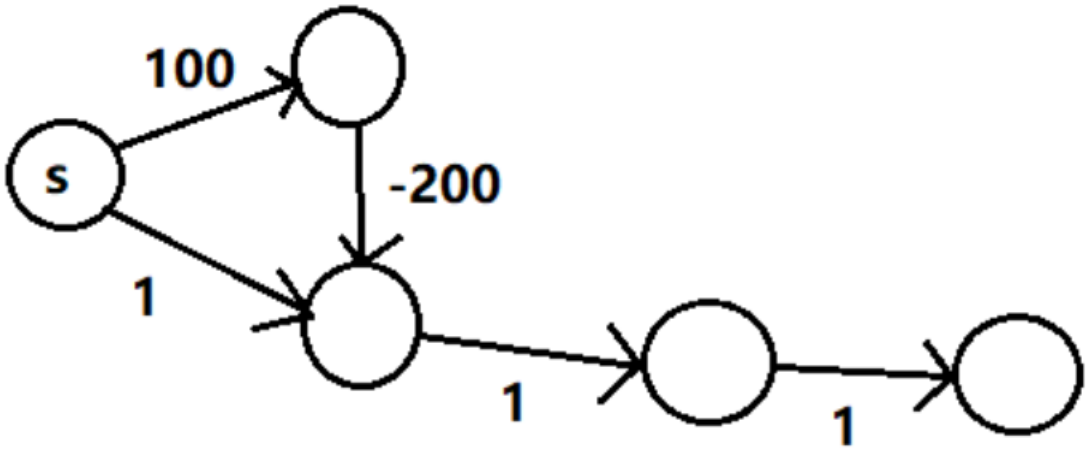
S={D(0),C(3),E(4),F(6),G(12),B(13),A(22)}

Notes

- Ties in Steps 3 and 4 can be broken arbitrarily.
- The algorithm can be used with directed, undirected, or mixed graphs, as long as link length $l(i,j) \geq 0$ for all $(i,j) \in A$.
- The algorithm can also be used for finding the shortest path between a source node and a specified terminal node.
- The algorithm is generally attributed to Dijkstra (1959).

Question

- What will happen if Dijkstra algorithm is applied to the following problem?



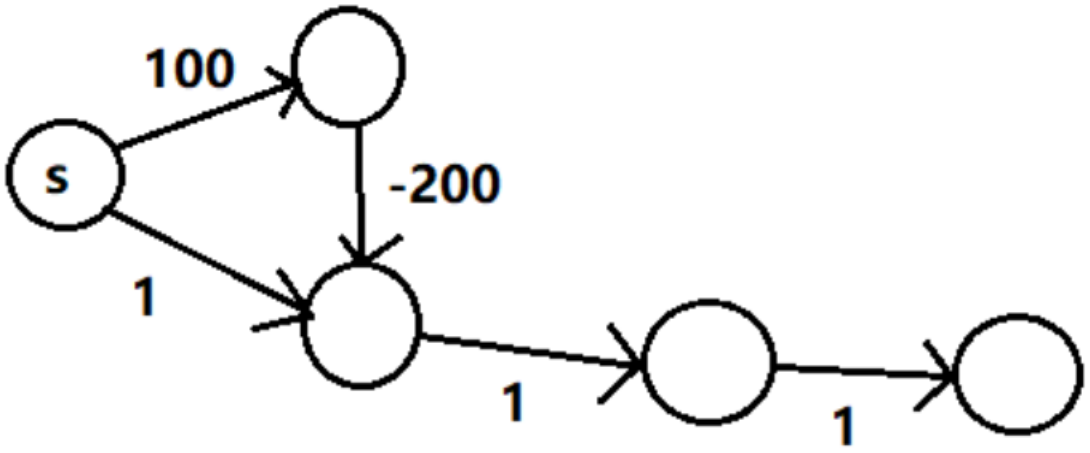
3.2 Bellman-Ford algorithm

Algorithm 1: Bellman Ford Algorithm

$\forall v \in V, d[v] \leftarrow \infty$ // set initial distance estimates
//optional: set $\pi(v) \leftarrow$nil for all v, $\pi(v)$ represents the predecessor of v
$d[s] \leftarrow 0$ // set distance to start node trivially as 0
for i *from* $1 \rightarrow n-1$ **do**
    for $(u, v) \in E$ **do**
        $d[v] \leftarrow \min\{d[v], d[u] + w(u, v)\}$ // update estimate of v
        // optional - if $d[v]$ changes, then $\pi(v) \leftarrow u$
// Negative Cycle Step
for $(u, v) \in E$ **do**
    if $d[v] > d[u] + w(u, v)$ **then**
        return "Negative Cycle"; // negative cycle detected
return $d[v] \; \forall \; v \in V$

Exercise

- Use Bellman-Ford algorithm to solve the following problem



3.3 Floyd algorithm

- Objective: To find the shortest paths between all pairs of nodes
- One straightforward idea: repeated application of Dijkstra algorithm
- Graph $G(N, A)$ with nodes $1,2, \ldots, n$
- Matrix D_k: element $d_k(i,j)$ is the shortest distance from i to j without traversing any node in $\{k+1, \ldots, n\}$
- Start with distance matrix D_0, and predecessor matrix P_0:

$$d_0(i,j) = \begin{cases} \ell(i,j) & \text{if arc } (i,j) \text{ exists}^3 \\ 0 & \text{if } i = j \\ \infty & \text{if arc } (i,j) \text{ does not exist} \end{cases}$$

$$p_0(i,j) = \begin{cases} i & \text{for } i \neq j \\ -(\text{blank}) & \text{for } i = j \end{cases}$$

3.3 Floyd algorithm

- Step 1: Set $k = 1$.
- Step 2: Obtain all the elements of the updated distance matrix D_k from the relation:

$$d_k(i,j) = \text{Min}\,[d_{k-1}(i,j), d_{k-1}(i,k) + d_{k-1}(k,j)]$$

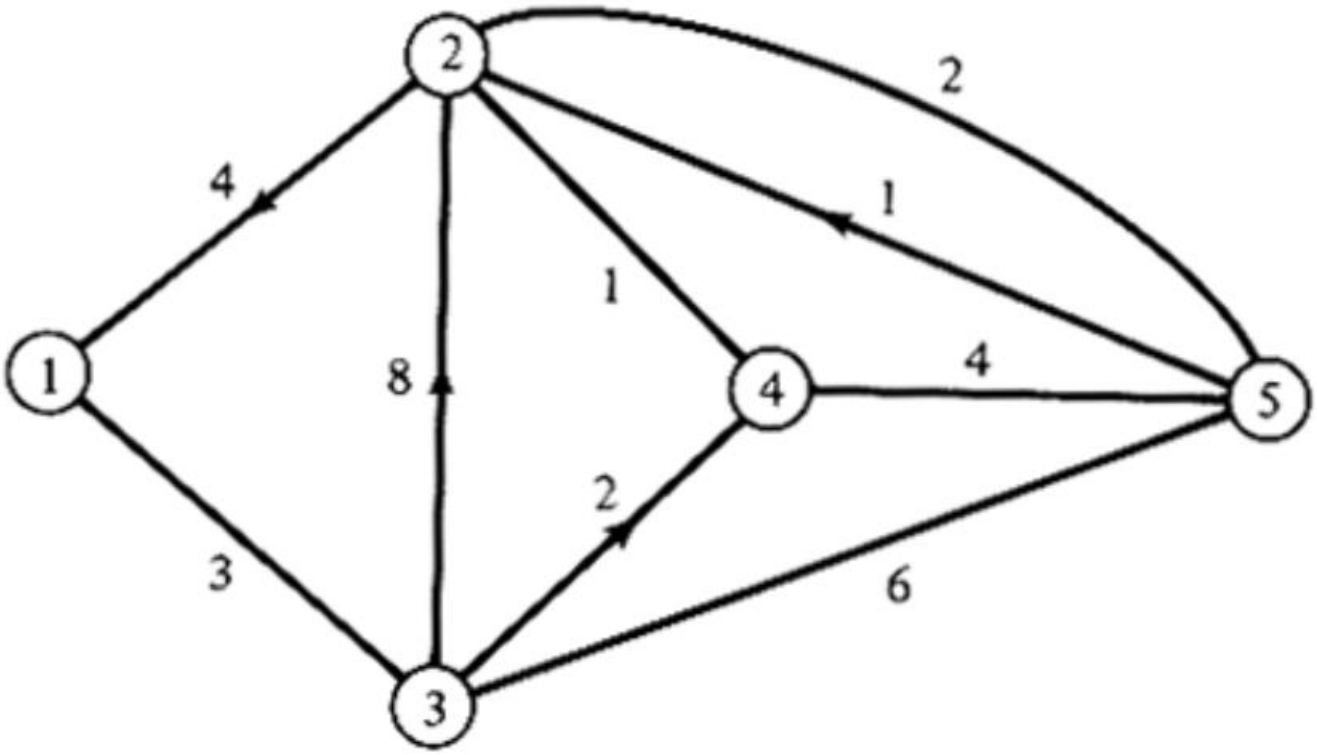
- Step 3: Obtain all the elements of the updated predecessor matrix P_k by using

$$p_k(i,j) = \begin{cases} p_{k-1}(k,j) & \text{if } d_k(i,j) \neq d_{k-1}(i,j) \\ p_{k-1}(i,j) & \text{otherwise} \end{cases}$$

- Step 4: If $k = n$, stop; if $k < n$, set $k = k + 1$ and return to step 2.

Example

- Find shortest paths between all pairs of nodes



Initialization:

$D^{(0)} =$

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

$P^{(0)} =$

	1	2	3	4	5
1	—	1	1	1	1
2	2	—	2	2	2
3	3	3	—	3	3
4	4	4	4	—	4
5	5	5	5	5	—

Through Node 1:

$D^{(1)} =$

	1*	2	3	4	5
*1	0	∞	3	∞	∞
2	4	0	7^+	1	2
3	3	8	0	2	6
4	∞	1	∞	0	4
5	∞	1	6	4	0

$P^{(1)} =$

	1*	2	3	4	5
*1	—	1	1	1	1
2	2	—	1^+	2	2
3	3	3	—	3	3
4	4	4	4	—	4
5	5	5	5	5	—

Through Node 2:

$D^{(2)} =$

	1	2*	3	4	5
1	0	∞	3	∞	∞
*2	4	0	7	1	2
3	3	8	0	2	6
4	5^+	1	8^+	0	3^+
5	5^+	1	6	2^+	0

$P^{(2)} =$

	1	2*	3	4	5
1	—	1	1	1	1
*2	2	—	1	2	2
3	3	3	—	3	3
4	2^+	4	1^+	—	2^+
5	2^+	5	5	2^+	—

Through Node 3:

$D^{(3)} =$

	1	2	3*	4	5
1	1	11^+	3	5^+	9^+
2	4	0	7	1	2
*3	3	8	0	2	6
4	5	1	8	0	3
5	5	1	6	2	0

$P^{(3)} =$

	1	2	3*	4	5
1	—	3^+	1	3^+	3^+
2	2	—	1	2	2
*3	3	3	—	3	3
4	2	4	1	—	2
5	2	5	5	2	—

Through Node 4:

$D^{(4)} =$

	1	2	3	4*	5
1	0	6^+	3	5	8^+
2	4	0	7	1	2
3	3	3^+	0	2	5^+
*4	5	1	8	0	3
5	5	1	6	2	0

$P^{(4)} =$

	1	2	3	4*	5
1	—	4^+	1	3	2^+
2	2	—	1	2	2
3	3	4^+	—	3	2^+
*4	2	4	1	—	2
5	2	5	5	3	—

Through Node 5:

$D^{(5)} =$

	1	2	3	4	5*
1	0	6	3	5	8
2	4	0	7	1	2
3	3	3	0	2	5
4	5	1	8	0	3
*5	5	1	6	2	0

$P^{(5)} =$

	1	2	3	4	5*
1	—	4	1	3	2
2	2	—	1	2	2
3	3	4	—	3	2
4	2	4	1	—	2
*5	2	5	5	2	—

Some complications for urban travel

- Turn Penalties and Constraints: a penalty will be incurred for left and/or right turns, or left and/or right turns are prohibited at some intersections.

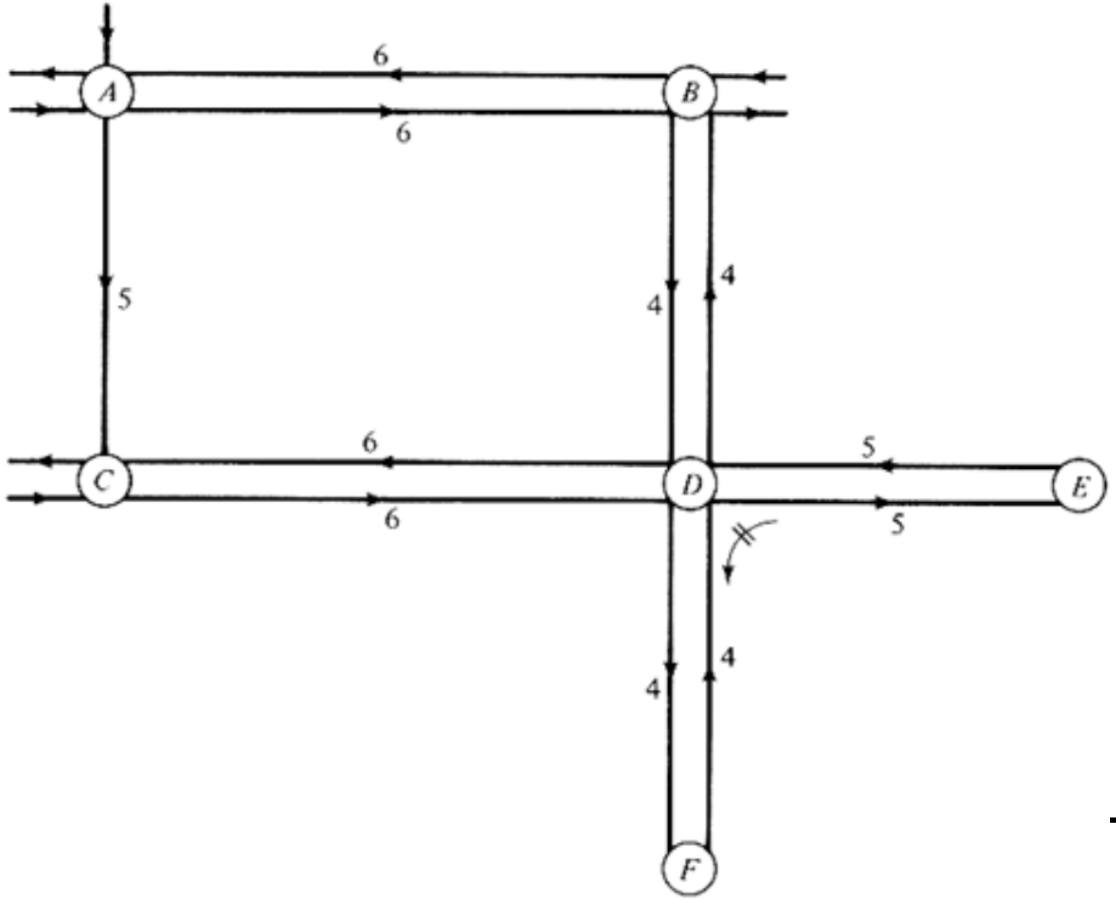


FIGURE 6.8 Section of a downtown urban grid

3.4 Complexity of algorithms

- Elementary operations: addition, subtraction, multiplication, division, comparison, branching,…
- Complexity of an algorithm: number of elementary operations required under "worst-case conditions".
- $O(n^3)$: complexity is proportional to n^3.
- Polynomial algorithms: complexity is proportional to or bounded by a polynomial function of the size of the input. E.g. $O(n^3)$, $O(n\log_2 n)$
- Exponential (or non-polynomial) algorithms: violate all polynomial bounds for sufficiently large sizes of the input. E.g. $O(2^n)$, $O(n!)$
- Exact vs heuristic algorithm

3.4 Complexity of algorithms

- Step 1: Set $k = 1$.
- Step 2: Obtain all the elements of the updated distance matrix D_k from the relation:

$$d_k(i, j) = \text{Min}\,[d_{k-1}(i, j), d_{k-1}(i, k) + d_{k-1}(k, j)]$$

$2(n\text{-}1)^2$

- Step 3: Obtain all the elements of the updated predecessor matrix P_k by using

$$p_k(i, j) = \begin{cases} p_{k-1}(k, j) & \text{if } d_k(i, j) \neq d_{k-1}(i, j) \\ p_{k-1}(i, j) & \text{otherwise} \end{cases}$$

$(n\text{-}1)^2$

- Step 4: If $k = n$, stop; if $k < n$, set $k = k + 1$ and return to step 2.

3

Homework

- P1. Give the conditions under which Dijkstra algorithm can work. Prove that under these conditions Dijkstra algorithm is correct, that is, we can indeed obtain the shortest paths by using Dijkstra algorithm.
- P2. Give the conditions under which Floyd algorithm can work. Prove that under these conditions Floyd algorithm is correct, that is, we can indeed obtain the shortest paths by using Floyd algorithm.
- P3. Give a counterexample for which Floyd algorithm does not work.
- P4. Analyze the time complexity of Dijkstra algorithm and Bellman-Ford algorithm.

- ***Due on April 28 (Wed), before 15:00 pm***
- ***Late submission will NOT be accepted!***