Introduction to Computers and Programming

Prof. I. K. Lundqvist

Some slides adapted from:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA and
16.410 Brian C. Williams

Lecture 6
Mar 30 2004

Today

- Problem Formulation
  – Problem solving as state space search
- Definition of Graphs
  – Types of Graphs
- Shortest Path problems
  – Dijkstra's Algorithm

Today

- **Problem Formulation**
  - **Problem solving as state space search**
- Definition of Graphs
  - Types of Graphs
- Shortest Path problems
  - Dijkstra's Algorithm

Complex missions must carefully:

- Plan complex sequences of actions
- Schedule tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

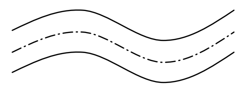
⇨ Most AI problems, like these, may be formulated as state space search.

Simple $\not\rightarrow$ Trivial

Can the astronaut get its produce safely across the Martian canal?

Astronaut
Goose
Grain
Fox

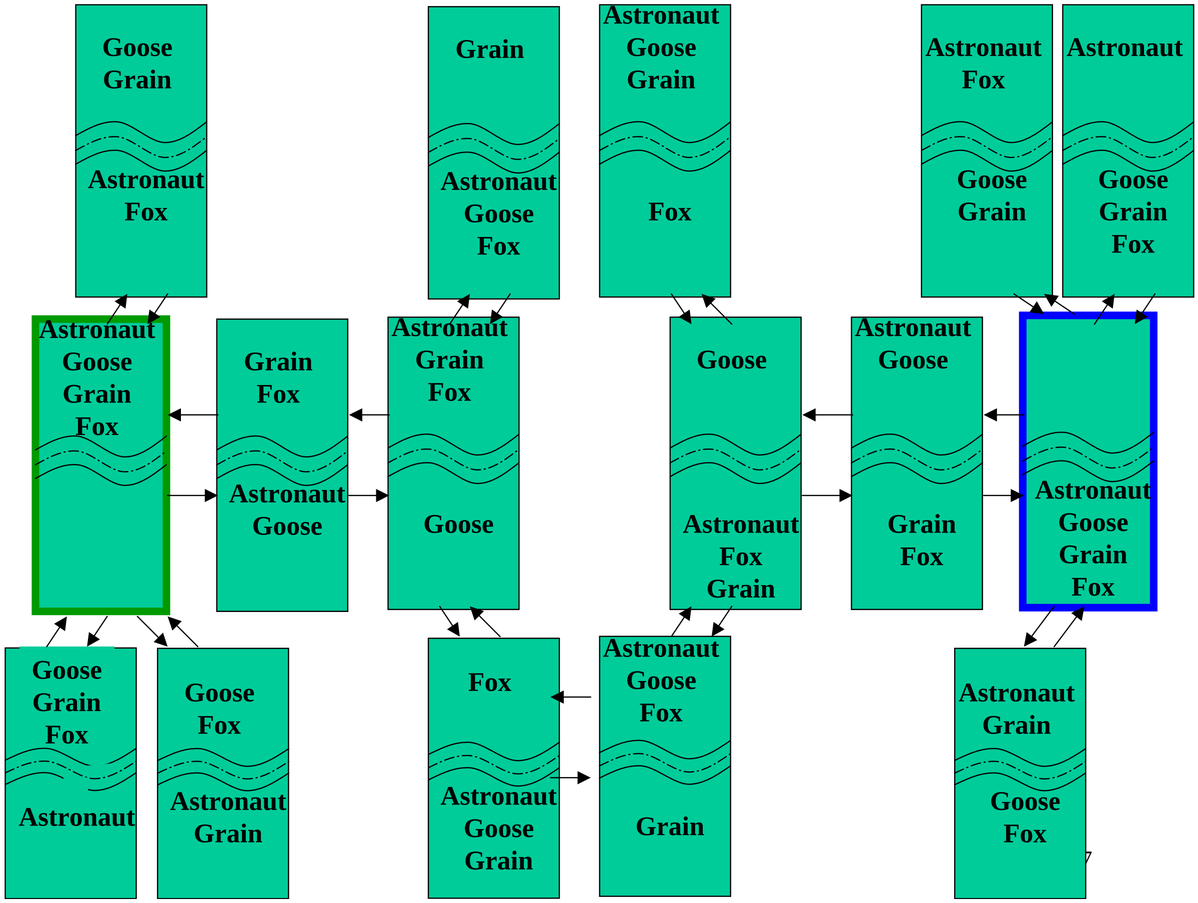
Rover

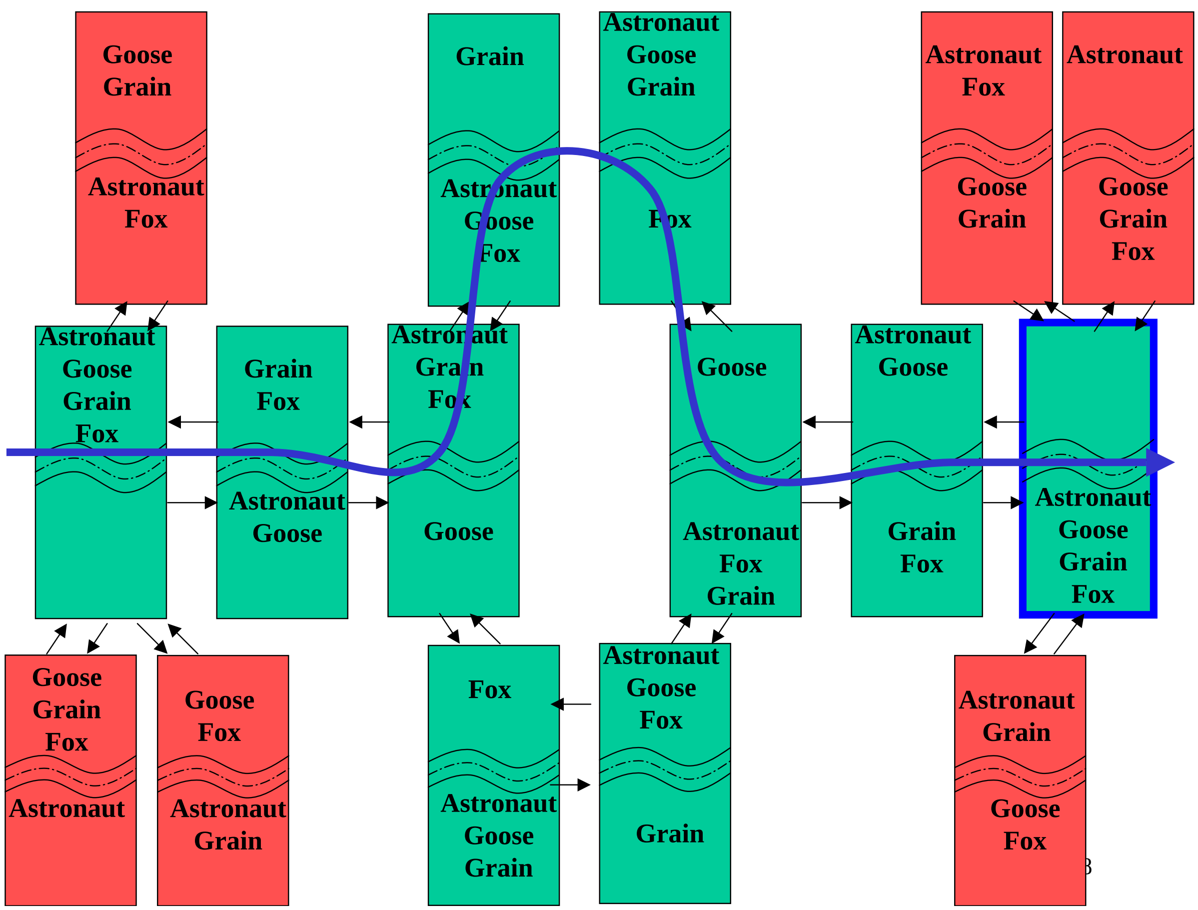


- Astronaut + 1 item allowed in the rover.
- Goose alone eats Grain
- Fox alone eats Goose

Problem Solving as State Space Search

- Formulate Goal
  - State
    - Astronaut, Fox, Goose & Grain across river
- Formulate Problem
  - States
    - Location of Astronaut, Fox, Goose & Grain at top or bottom river bank
  - Operators
    - Move rover with astronaut & 1 or 0 items to other bank
- Generate Solution
  - Sequence of States
    - Move(goose,astronaut), Move(astronaut), . . .


Goose Grain
Astronaut Fox
Grain
Astronaut Goose Fox
Astronaut Goose Grain
Fox
Astronaut Fox
Goose Grain
Astronaut
Goose Grain Fox
Astronaut Goose Grain Fox
Grain Fox
Astronaut Goose
Astronaut Grain Fox
Goose
Goose
Astronaut Fox Grain
Astronaut Goose
Grain Fox
Astronaut Goose Grain Fox
Goose Grain Fox
Astronaut
Goose Fox
Astronaut Grain
Fox
Astronaut Goose Grain
Astronaut Goose Fox
Grain
Astronaut Grain
Goose Fox

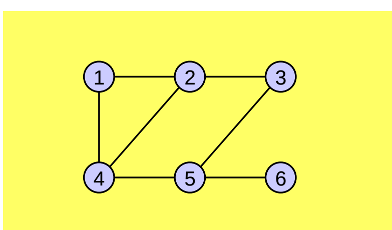

Goose Grain
Astronaut Fox
Grain
Astronaut Goose Fox
Astronaut Goose Grain
Fox
Astronaut Fox
Goose Grain
Astronaut
Goose Grain Fox
Astronaut Goose Grain Fox
Grain Fox
Astronaut Goose
Astronaut Grain Fox
Goose
Goose
Astronaut Fox Grain
Astronaut Goose
Grain Fox
Astronaut Goose Grain Fox
Goose Grain Fox
Astronaut
Goose Fox
Astronaut Grain
Fox
Astronaut Goose Grain
Astronaut Goose Fox
Grain
Astronaut Grain
Goose Fox

Today

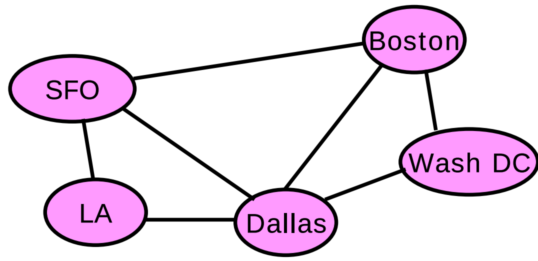
- Problem Formulation
  - Problem solving as state space search
- **Definition of Graphs**
  - **Types of Graphs**
- Shortest Path problems
  - Dijkstra's Algorithm

Graph

- A graph is a generalization of the simple concept of a set of dots (called **vertices** or nodes) connected by links (called **edges** or arcs)
  - Example: graph with 6 vertices and 7 edges



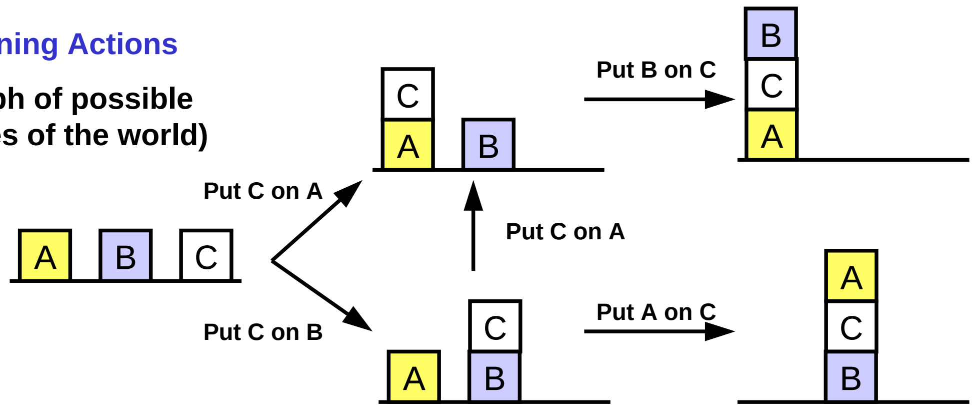
Examples of Graphs



Airline Routes

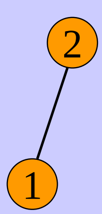
Planning Actions

(graph of possible states of the world)

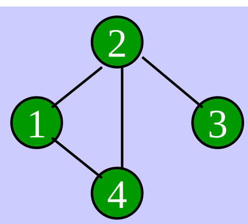


Graphs

- A **graph** G = (V, E) is a finite nonempty set of vertices and a set of edges

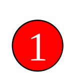


V = {1, 2, 3}
E = {(1, 2)}



V = {1, 2, 3, 4}
E = {(1,2)(2,3)(1,4)(2,4)}

- An **empty graph** is the graph whose edge set is empty

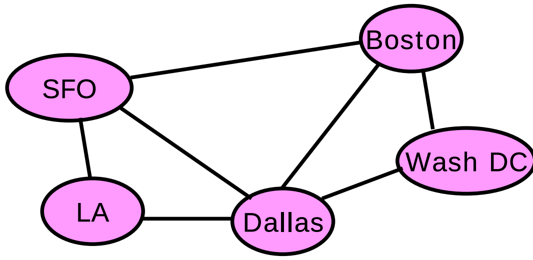


V = {1}
E = {∅}

- The **null graph** is the graph whose edge set and vertex set are empty

V = {∅}
E = {∅}

Examples of Graphs



Airline Routes

Graph AirlineRoutes is represented as the pair (V,E)

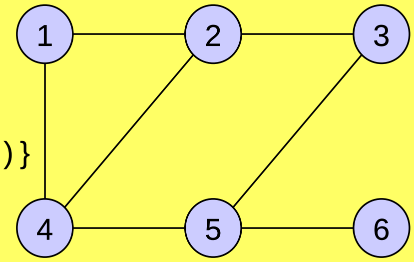
V= {Bos, SFO, LA, Dallas, Wash DC}

E= {(SFO,Bos),(SFO,LA),(LA,Dallas),(Dallas,Wash DC)...}

Graphs

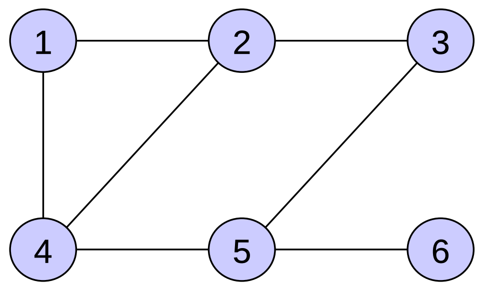
- A **loop** in a graph is an edge e in E whose endpoints are the same vertex.
- A **simple** graph is a graph with no loops, and there is at most one edge between any pair of vertices.

A simple graph with
V = {1, 2, 3, 4, 5, 6}
E = {(1,2), (1,4), (2,3), (2,4), (3,5), (5,6), (4,5)}

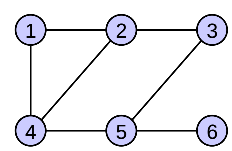


Graphs

- A **multigraph** has two or more edges that connect the same pair of vertices
- A **cycle** is a path that begins and ends with the same vertex
  - A cycle of length 1 is a loop
  - (1, 2, 3, 5, 4, 2, 1) is a cycle of length 6



Vertices

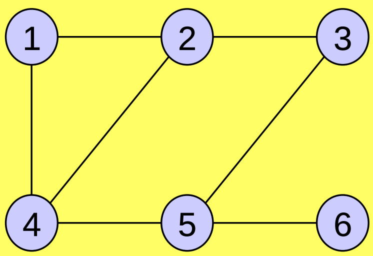


- Two vertices, *u* and *v* in an undirected graph G are called **adjacent** (or neighbors) in G, if {(*u*,*v*)} is an edge of G.

- The **degree** of a vertex in an undirected graph is the number of edges **incident** with it, except that a loop at a vertex contributes twice to the degree of that vertex.

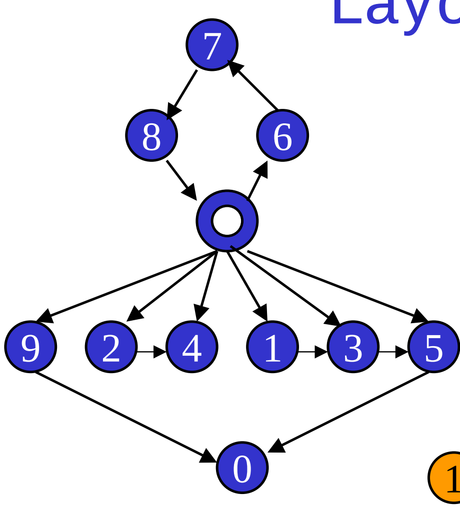
Adjacency Matrix

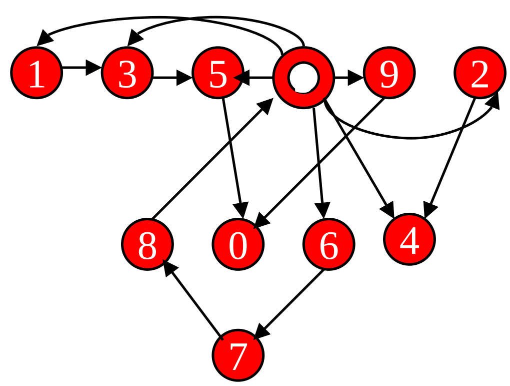
- A finite graph is often represented by its **adjacent matrix**.
  - An entry in row *I* and column *j* gives the number of edges from the i^{th} to the j^{th} vertex.

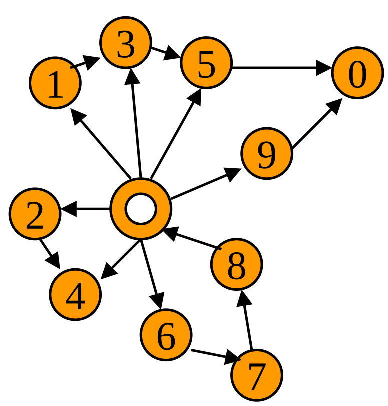


$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Layout of Graphs

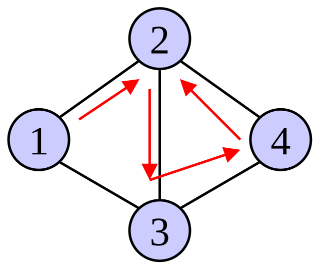




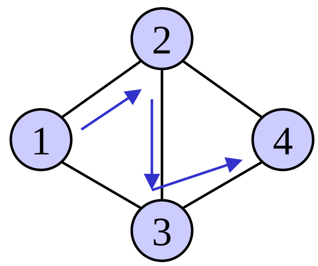


Walks and Paths

- A **walk** is a sequence of vertices (v_1, v_2, …, v_k) in which each *adjacent* vertex pair is an edge
- A **path** is a walk with no repeated vertices



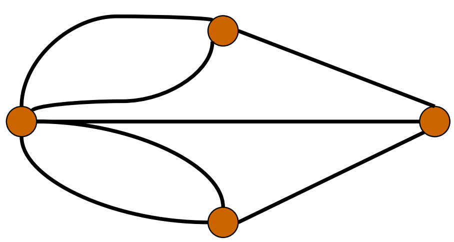
Walk (1,2,3,4,2)

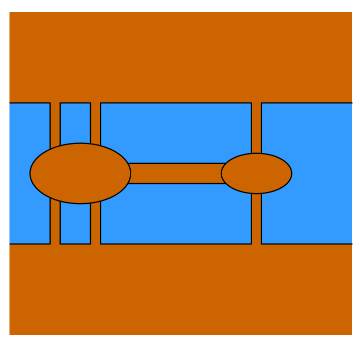


Path (1,2,3,4)

"The 1st problem in Graph Theory" Seven Bridges of Königsberg

- The city of Königsberg was set on the River Pregel, and included two large islands which were connected to each other and the mainland by seven bridges.
  - Was it possible to walk a route that crossed each bridge exactly once, and return to the starting point?



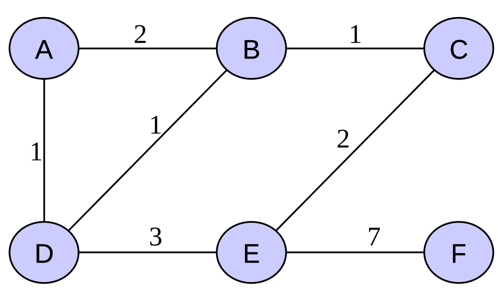


“The 1st problem in Graph Theory” Seven Bridges of Königsberg

- An **Eulerian path** in a graph is a path that uses each edge precisely once.
  - If such path exists, the graph is called traversable
- Euler showed that an Eulerian cycle exists if and only if all vertices in the graph are of even **degree**.

Weighted Graph

- A **weighted** graph associates a value (weight) to every edge in the graph.
  - A **weight of a path** in a weighted graph is the sum of the weights of the traversed edges.



- **Directed** graph (digraph) is a graph with one-way edges

Today

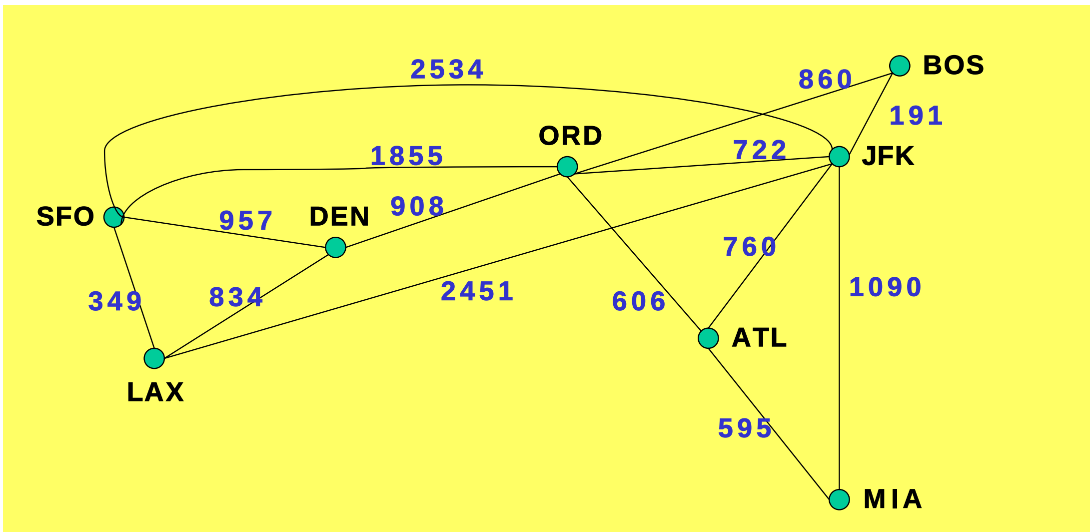
- Problem Formulation
  - Problem solving as state space search
- Definition of Graphs
  - Types of Graphs
- **Shortest Path problems**
  - **Dijkstra's Algorithm**

Shortest Path Problems

- The **shortest path** from v_1 to v_2
  - Is the path of the smallest weight between the two vertices
  - Shortest may be least number of edges, least total weight, etc.
  - The weight of that path is called the **distance** between them

Shortest Path Problems

- Example: the weight can be mileage, fares, etc.



Shortest Path Problems

- **Dijkstra's algorithm**
  - Finds shortest path for a directed and connected graph G(V,E) which has non-negative weights.
  - Applications:
    - Internet routing
    - Road generation within a geographic region
    - ...

Dijkstra's Algorithm

- Dijkstra(G,w,s)

```
Init_Source(G,s)
S := empty set
Q := set of all vertices

while Q is not an empty set loop
    u := Extract_Min(Q)
    S := S union {u}
    for each vertex v which is a neighbor of u loop
        Relax(u,v,w)
```


Dijkstra's Algorithm

- Init_Source(G,s)

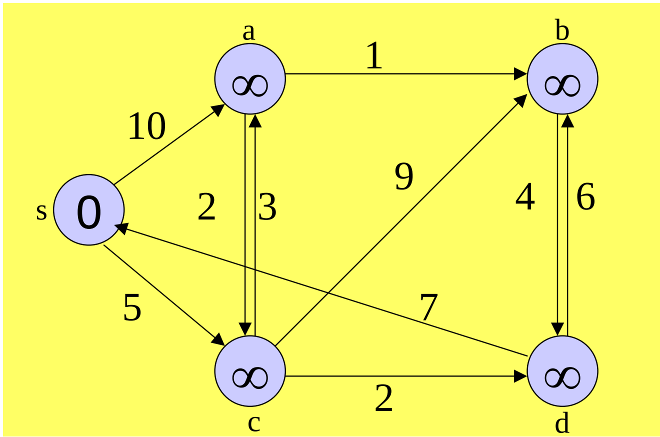
```
for each vertex v in V[G] loop
    d[v] := infinite
    previous[v] := 0
d[s] := 0
```

- v = Extract_Min(Q) searches for the vertex v in the vertex set Q that has the least d[v] value. That vertex is removed from the set Q and then returned.

- Relax(u,v,w)

```
if d[v] > d[u] + w(u,v) then
    d[v] := d[u] + w(u,v)
    previous[v] := u
```

Dijkstra’s Algorithm

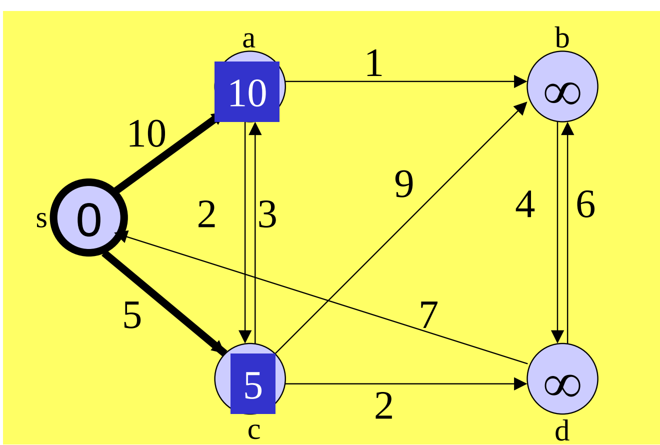


V = {a, b, c, d, s}
E = {(s,c), (c,d), (d,b), (b,d), (c,b), (a,c), (c,a), (a,b), (s,a)}

S = {∅}
Q = {s, a, b, c, d}

$$d = \begin{pmatrix} 0 \\ \infty \\ \infty \\ \infty \\ \infty \end{pmatrix} \qquad prev = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Dijkstra’s Algorithm



S = {s}
Q = {a, b, c, d}

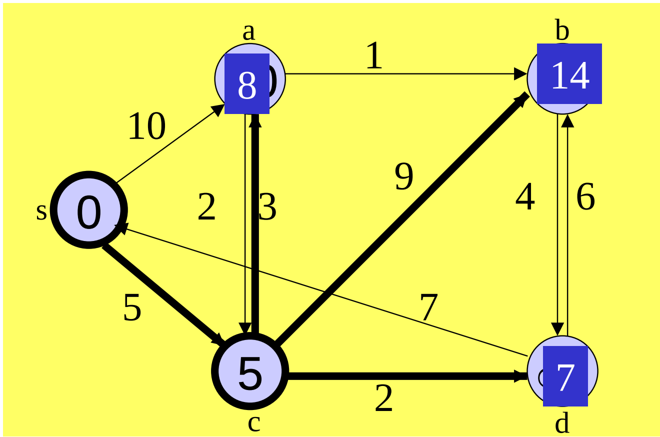
$$d = \begin{pmatrix} 0 \\ \infty \\ \infty \\ \infty \\ \infty \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 10 \\ \infty \\ 5 \\ \infty \end{pmatrix}$$

Extract_Min (Q) → s
Neighbors of s = a, c

Relax (s,c,5)
Relax (s,a,10)

$$prev = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ s \\ 0 \\ s \\ 0 \end{pmatrix}$$

Dijkstra's Algorithm



S = {s, c}
Q = {a, b, d}

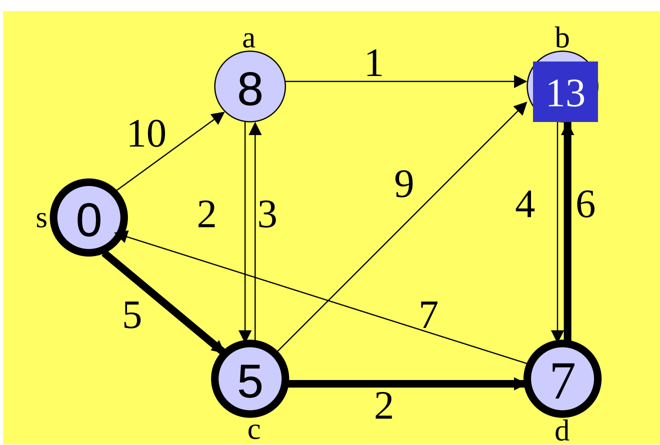
d = (0, 10, ∞, 5, ∞) → (0, 8, 14, 5, 7)

Extract_Min (Q) → c
Neighbors of c = a, b, d

prev = (0, s, 0, s, 0) → (0, c, c, s, c)

Relax (c,a,3)
Relax (c,b,9)
Relax (c,d,2)

Dijkstra's Algorithm



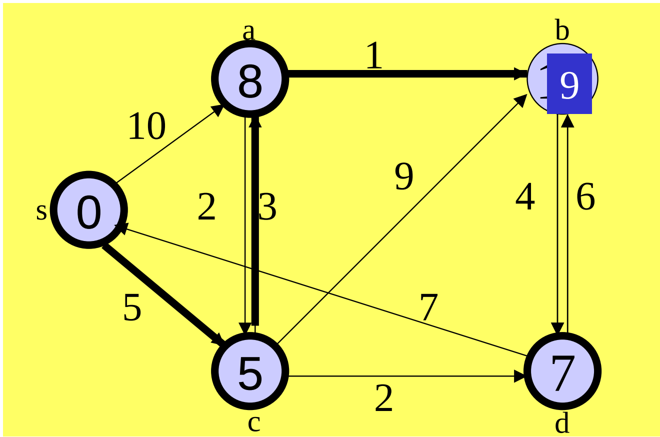
S = {s, c, d}
Q = {a, b}

d = (0, 8, 14, 5, 7) → (0, 8, 13, 5, 7)

Extract_Min (Q) → d
Neighbors of d = b
Relax (d,b,6)

prev = (0, c, c, s, c) → (0, c, d, s, c)

Dijkstra's Algorithm



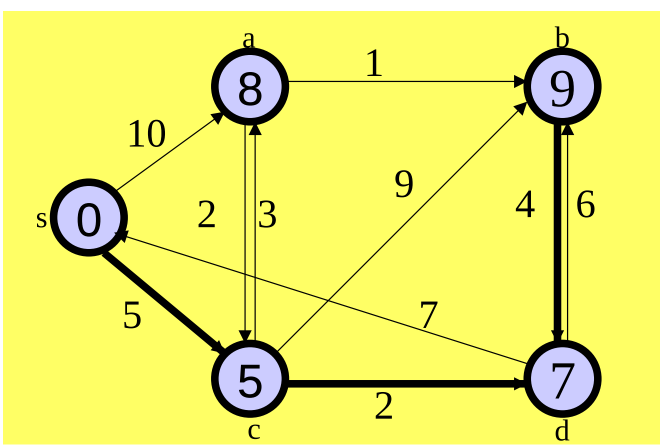
S = {s, c, d, a}
Q = {b}

$$d = \begin{pmatrix} 0 \\ 8 \\ 13 \\ 5 \\ 7 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 8 \\ 9 \\ 5 \\ 7 \end{pmatrix}$$

Extract_Min (Q) → a
Neighbors of a = b, c
Relax (a,b,1)
Relax (a,c,3)

$$prev = \begin{pmatrix} 0 \\ c \\ d \\ s \\ c \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ c \\ a \\ s \\ c \end{pmatrix}$$

Dijkstra's Algorithm



S = {s, c, d, a, b}
Q = {}

$$d = \begin{pmatrix} 0 \\ 8 \\ 9 \\ 5 \\ 7 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 8 \\ 9 \\ 5 \\ 7 \end{pmatrix}$$

Extract_Min (Q) → b
Neighbors of b = d
Relax (b, d, 4)

$$prev = \begin{pmatrix} 0 \\ c \\ a \\ s \\ c \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ c \\ a \\ s \\ c \end{pmatrix}$$