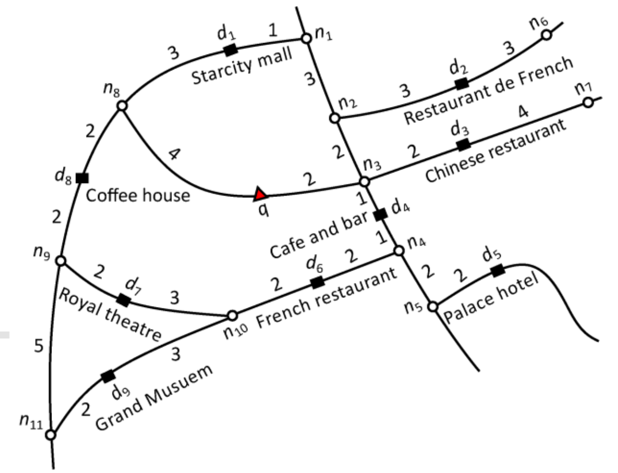
COMP 1002/COMP 1001

Lecture 9

Problem Solving III

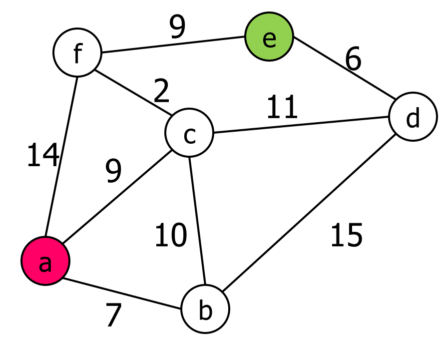
Graph



- Graphs are so common.
- How can we represent a graph?
  - There are two parts:
    - Nodes: can be represented as a list or a set.
    - Edges: can be represented as a list or a set.
      - For each edge, we would need a pair or a tuple expressing the two nodes.
  - Some graphs need more information on the edges:
    - Weight of an edge.
    - Direction of an edge.

Graph

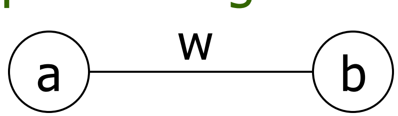
- In computer science, a graph G is often written like:
  - **G = (V,E)**
    - **V** is a set of vertices or nodes.
    - **E** is a set of edges.



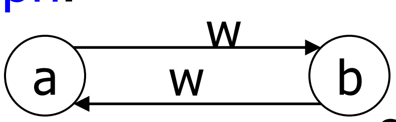
- Example
  - Nodes
    - {a,b,c,d,e,f} (6 nodes)
  - Edges
    - {ab, ac, af, bc, bd, cd, cf, de, ef} (9 edges)
  - With weights on edges:
    - {ab=7, ac=9, af=14, bc=10, bd=15, cd=11, cf=2, de=6, ef=9}
  - This graph is undirected (no direction on edges).

Graph

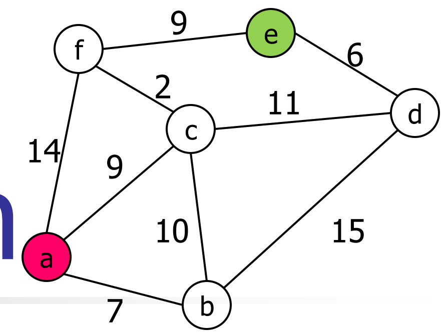
- We need to build the data model for the graph before we can use the computer to process it.
  - It is easy to represent the nodes, but perhaps harder with the edges.
  - We may represent nodes and edges separately as two different groups.
  - It is more natural to represent edges as linked to nodes. There are two common representations.
    - Adjacency list
    - Adjacency matrix
  - Note that an edge in an undirected graph is equivalent to a pair of edges in a directed graph.



same as

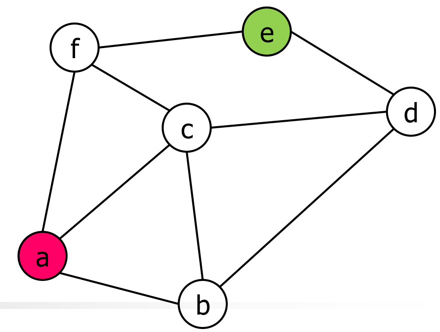


Graph Representation



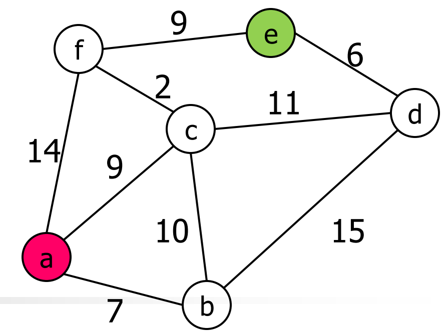
- Simply **G = (V,E)**.
  - Set of nodes:
    - **V** = {a,b,c,d,e,f}
  - Set of edges:
    - **E** = {ab, ac, af, bc, bd, cd, cf, de, ef}
    - **E** = {(a,b), (a,c), (a,f), (b,c), (b,d), (c,d), (c,f), (d,e), (e,f)}
  - Edges storing the weights:
    - **E** = {ab=7, ac=9, af=14, bc=10, bd=15, cd=11, cf=2, de=6, ef=9}
    - **E** = {(a,b,7), (a,c,9), (a,f,14), (b,c,10), (b,d,15), (c,d,11), (c,f,2), (d,e,6), (e,f,9)}
  - Should we try to store nodes and edges together instead of separately?

Adjacency List



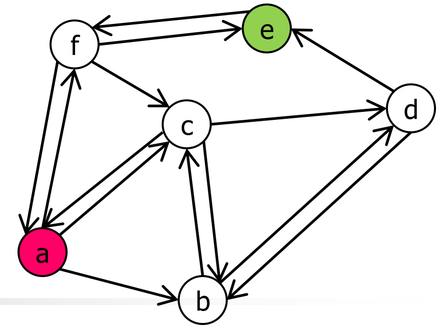
- For each node, put together the edges for each node.
- An adjacency list is a list for each node, showing the neighbors of that node, i.e. edges.
- Example
  - Nodes, a set
    - {a,b,c,d,e,f}
  - Edges in 6 adjacency lists: D(node), each being a set
    - D(a) = {b,c,f}
    - D(b) = {a,c,d}
    - D(c) = {a,b,d,f}
    - D(d) = {b,c,e}
    - D(e) = {d,f}
    - D(f) = {a,c,e}

Adjacency List



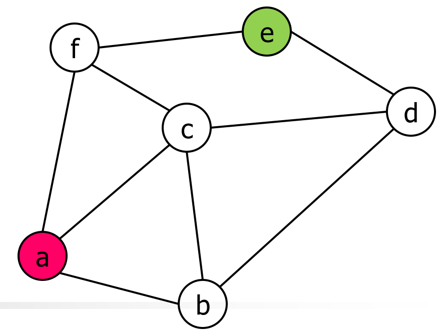
- For edges with weights, an adjacency list is a list for each node, showing the edges and the weights.
- Example
  - Nodes, a set
    - {a,b,c,d,e,f}
  - Edges in adjacency lists: D(node), each being a set of tuples
    - D(a) = {(b,7),(c,9),(f,14)}
    - D(b) = {(a,7),(c,10),(d,15)}
    - D(c) = {(a,9),(b,10),(d,11),(f,2)}
    - D(d) = {(b,15),(c,11),(e,6)}
    - D(e) = {(d,6),(f,9)}
    - D(f) = {(a,14),(c,2),(e,9)}

Adjacency List



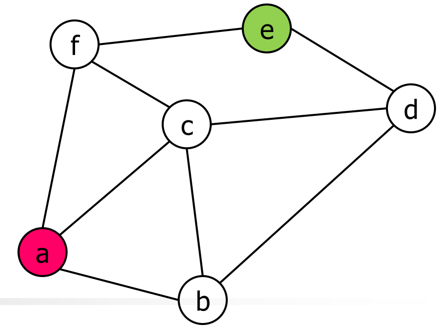
- For a directed graph (graph with direction on edges), an adjacency list is a list for each node, showing the next reachable node and perhaps also the weights.
- Example
  - Nodes, a set
    - {a,b,c,d,e,f}
  - Edges in adjacency lists: D(node), each being a set
    - D(a) = {b,c,f}
    - D(b) = {c,d}
    - D(c) = {a,b,d}
    - D(d) = {b,e}
    - D(e) = {f}
    - D(f) = {a,c,e}

Adjacency List



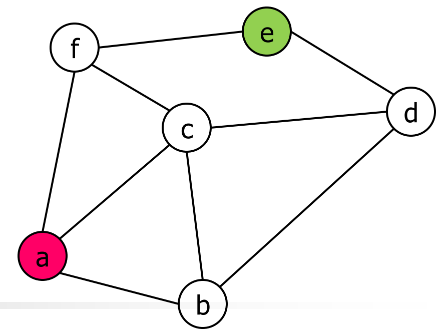
- Modeling in Python
  - Nodes can be represented as a list.
  - Edges can be represented as a list of lists.
- Example
  - Nodes as a list:
    - N = ["a", "b", "c", "d", "e", "f"]
  - Edges as a list of lists:
    - D = [["b", "c", "f"],
      ["a", "c", "d"],
      ["a", "b", "d", "f"],
      ["b", "c", "e"],
      ["d", "f"],
      ["a", "c", "e"]]

Adjacency List

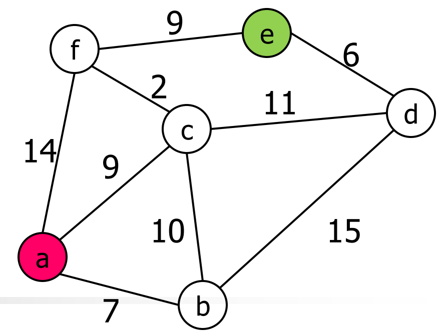


- Modeling in Python
  - Nodes can be represented as a list.
  - Edges can be represented as a list of sets.
- Example
  - Nodes as a list:
    - N = ["a", "b", "c", "d", "e", "f"]
  - Edges as a list of sets:
    - D = [{"b", "c", "f"},
      {"a", "c", "d"},
      {"a", "b", "d", "f"},
      {"b", "c", "e"},
      {"d", "f"},
      {"a", "c", "e"}]

Adjacency List



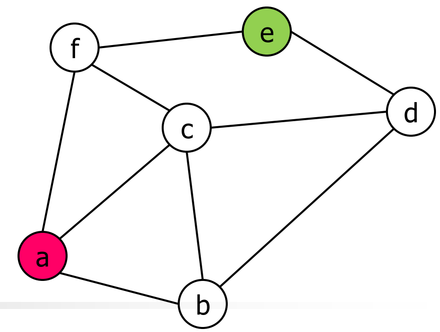
- Modeling in Python
  - Nodes can be represented as dictionary keys.
  - Edges can be represented as dictionary values in list/set.
- Example
  - Node and edges as a dictionary:
    - D = { ″a″: {″b″, ″c″, ″f″},
      ″b″: {″a″, ″c″, ″d″},
      ″c″: {″a″, ″b″, ″d″, ″f″},
      ″d″: {″b″, ″c″, ″e″},
      ″e″: {″d″, ″f″},
      ″f″: {″a″, ″c″, ″e″} }



Adjacency List

- Modeling in Python
  - Nodes can be represented as a list.
  - Edges with weights can be represented as a list of lists of tuples.
- Example
  - Nodes as a list:
    - N = ["a", "b", "c", "d", "e", "f"]
  - Edges as a list of lists of tuples:
    - D = [[("b",7),("c",9),("f",14)],
      [("a",7),("c",10),("d",15)],
      [("a",9),("b",10),("d",11),("f",2)],
      [("b",15),("c",11),("e",6)],
      [("d",6),("f",9)],
      [("a",14),("c",2),("e",9)]]

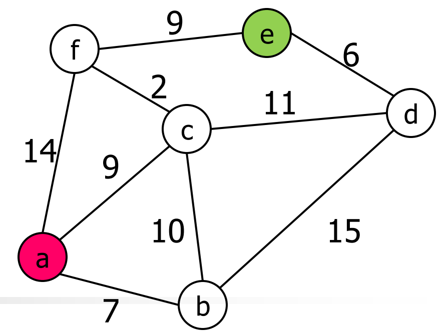
Adjacency Matrix



- For each pair of nodes, maintain a matrix to show the neighborhood between the pair.
  - An adjacency matrix has each node in a row (source) and in a column (destination).
  - 1 / True means an edge and 0 / False means no edge.
- Example
  - Matrix D[6,6] for a graph with 6 nodes
  - D =

	a	b	c	d	e	f
a	0	1	1	0	0	1
b	1	0	1	1	0	0
c	1	1	0	1	0	1
d	0	1	1	0	1	0
e	0	0	0	1	0	1
f	1	0	1	0	1	0

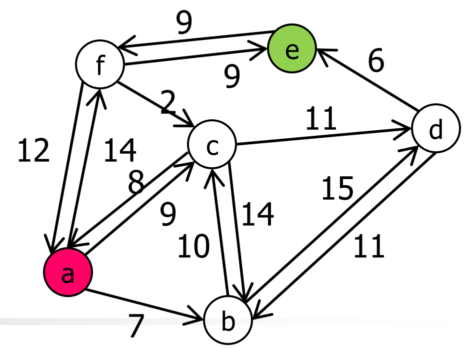
Adjacency Matrix



- An adjacency matrix for a graph with weight have matrix elements showing neighborhood and weight.
- This is often called a distance matrix when the weight is the distance.
- Example
  - Matrix D[6,6] for a graph with 6 nodes
  - D =

	a	b	c	d	e	f
a	0	7	9	∞	∞	14
b	7	0	10	15	∞	∞
c	9	10	0	11	∞	2
d	∞	15	11	0	6	∞
e	∞	∞	∞	6	0	9
f	14	∞	2	∞	9	0

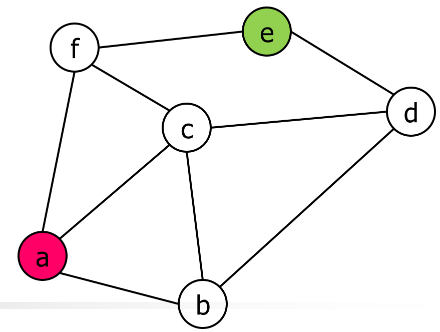
Adjacency Matrix



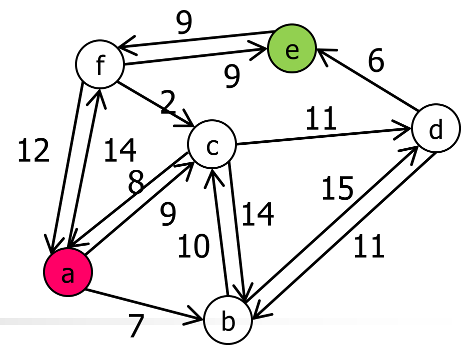
- An distance matrix for a directed graph with weight have matrix elements showing the weight/distance from node in row i to node in column j.
- Example
  - Matrix D[6,6] for a graph with 6 nodes
  - D =

	a	b	c	d	e	f
a	0	7	9	∞	∞	14
b	∞	0	10	15	∞	∞
c	8	14	0	11	∞	∞
d	∞	11	∞	0	6	∞
e	∞	∞	∞	∞	0	9
f	12	∞	2	∞	9	0

Adjacency Matrix



- Modeling in Python
  - The 2-D matrix is normally represented as a list of lists as a logical 2-D array.
- Example
  - Nodes as a list:
    - N = ["a", "b", "c", "d", "e", "f"]
  - Matrix as a list of lists:
    - D = [[0, 1, 1, 0, 0, 1],
      [1, 0, 1, 1, 0, 0],
      [1, 1, 0, 1, 0, 1],
      [0, 1, 1, 0, 1, 0],
      [0, 0, 0, 1, 0, 1],
      [1, 0, 1, 0, 1, 0]]

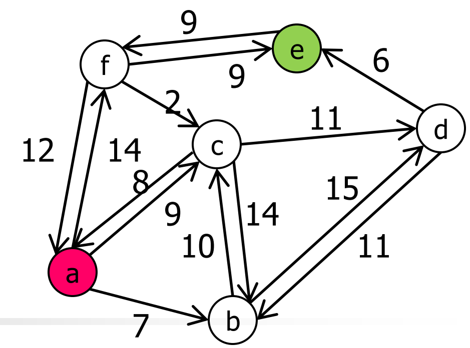


Adjacency Matrix

- Modeling in Python
  - The 2-D matrix can also be represented as a dictionary of dictionary.
- Example
  - Nodes as a list:
    - N = [″a″, ″b″, ″c″, ″d″, ″e″, ″f″]

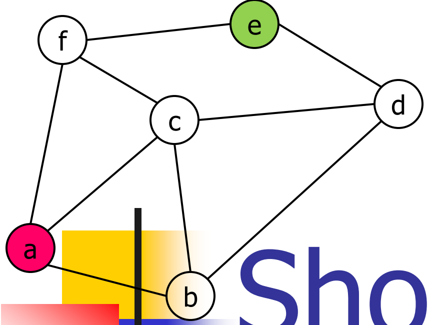
Not really needed with d

  - Matrix as a dictionary of dictionary.
    - D = { ″a″: {″a″:0,″b″:7,″c″:9,″d″:inf,″e″:inf,″f″:14},
      ″b″: {″a″:inf,″b″:0,″c″:10,″d″:15,″e″:inf,″f″:inf},
      ″c″: {″a″:8,″b″:14,″c″:0,″d″:11,″e″:inf,″f″:inf},
      ″d″: {″a″:inf,″b″:11,″c″:inf,″d″:0,″e″:6,″f″:inf},
      ″e″: {″a″:inf,″b″:inf,″c″:inf,″d″:inf,″e″:0,″f″:9},
      ″f″: {″a″:12,″b″:inf,″c″:2,″d″:inf,″e″:9,″f″:0} }
    - Here, `inf = float(″infinity″)` in Python, meaning infinity (∞).

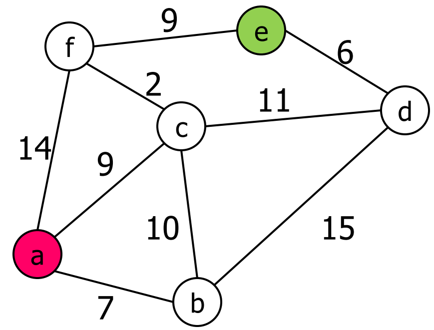


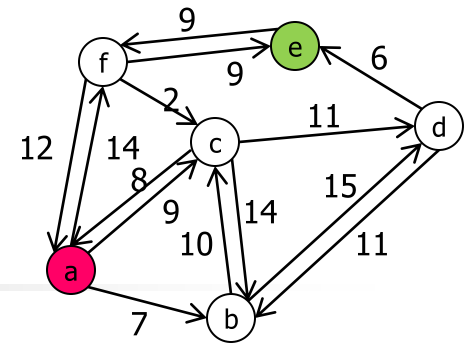
Adjacency Matrix

- Modeling in Python
  - The 2-D matrix can also be represented as a dictionary of dictionary.
- Example
  - Nodes as a list: Not really needed with dictionary D
    - N = ["a", "b", "c", "d", "e", "f"]
  - Matrix as a dictionary of dictionary.
    - D = { "a": {"a":0,"b":7,"c":9,"d":inf,"e":inf,"f":14},
      "b": {"a":inf,"b":0,"c":10,"d":15,"e":inf,"f":inf},
      "c": {"a":8,"b":14,"c":0,"d":11,"e":inf,"f":inf},
      "d": {"a":inf,"b":11,"c":inf,"d":0,"e":6,"f":inf},
      "e": {"a":inf,"b":inf,"c":inf,"d":inf,"e":0,"f":9},
      "f": {"a":12,"b":inf,"c":2,"d":inf,"e":9,"f":0} }
    - Here, `inf = float("infinity")` in Python, meaning infinity (∞).



Shortest Path

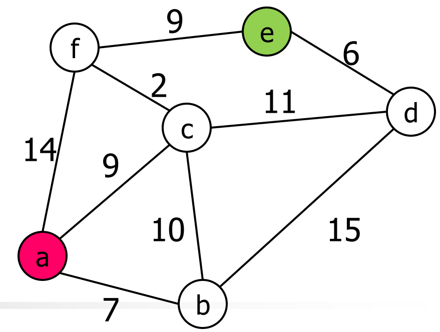




- A highly common application on a graph is to find the shortest path from one node to another.
  - The starting node is often called source node in graph theory.
  - The target node is often called destination node.
  - The graphs may contain weights, or without.
    - We call them weighted graphs and unweighted graphs respectively.
  - The graphs may contain edges with or without directions.
    - We call these two types directed graphs and undirected graphs respectively.

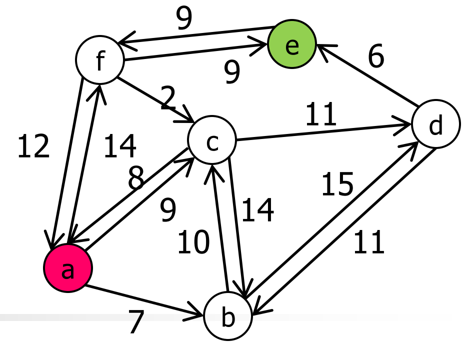
- There are other applications on a graph, e.g. breadth first search, depth first search, topological sort.

Shortest Path



- Can you find the shortest path from a to e?
  - Algorithm 1: Layman approach to find all possible paths first.
    - $a \to f \to e$ — 23
    - $a \to c \to f \to e$ — 20
    - $a \to c \to d \to e$ — 26
    - $a \to b \to c \to d \to e$ — 34
    - $a \to b \to d \to e$ — 28
    - $a \to c \to b \to d \to e$ — 40
    - Any more?
    - $a \to b \to c \to f \to e$ — 28
    - $a \to b \to d \to c \to f \to e$ — 44
    - Yet more?
    - $a \to f \to c \to d \to e$ — 33
    - $a \to f \to c \to b \to d \to e$ — 47
  - Mission impossible for larger graph!

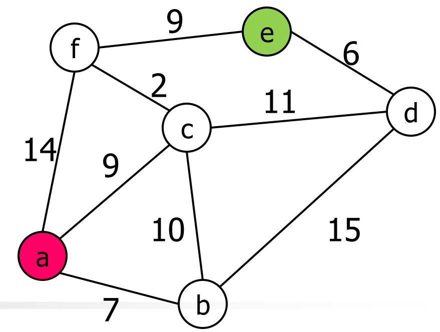
Shortest Path



- Can you find the shortest path from a to e?
  - This is a directed graph.
  - There are fewer possible paths than previous one.
    - a → f → e
    - a → c → d → e
    - a → b → c → d → e
    - a → b → d → e
    - a → c → b → d → e
    - a → f → c → d → e
    - a → f → c → b → d → e

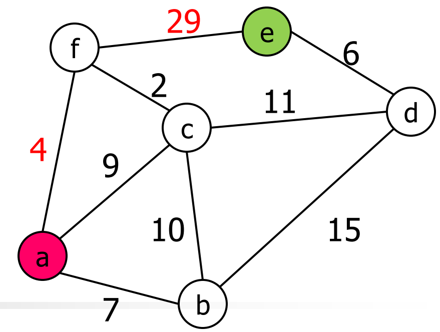
    - 23
    - 26
    - 34
    - 28
    - 44
    - 33
    - 51
  - Still mission impossible for larger graph!
  - What is the maximum number of possible paths from a to e in this small graph with 6 nodes?

Improvement



- There are so many possible paths.
- Algorithm 2: search for paths starting with fewer steps.
  - $a \to f \to e$ — 23
  - $a \to b \to d \to e$ — 28
  - $a \to c \to d \to e$ — 26
  - $a \to c \to f \to e$ — 20
  - Stop here?
  - $a \to b \to c \to d \to e$ — 34
  - $a \to b \to c \to f \to e$ — 28
  - $a \to c \to b \to d \to e$ — 40
  - $a \to f \to c \to d \to e$ — 33
  - Stop here?
  - $a \to b \to d \to c \to f \to e$ — 44
  - $a \to f \to c \to b \to d \to e$ — 47

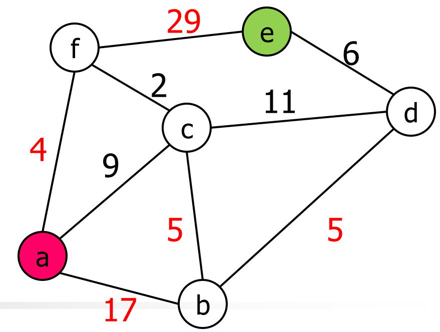
Improvement



- Consider this graph.
- Search for paths starting with fewer steps.
  - $a \rightarrow f \rightarrow e$
  - $a \rightarrow b \rightarrow d \rightarrow e$
  - $a \rightarrow c \rightarrow d \rightarrow e$
  - $a \rightarrow c \rightarrow f \rightarrow e$
  - Stop here?
  - $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$
  - $a \rightarrow b \rightarrow c \rightarrow f \rightarrow e$
  - $a \rightarrow c \rightarrow b \rightarrow d \rightarrow e$
  - $a \rightarrow f \rightarrow c \rightarrow d \rightarrow e$
  - Stop here?
  - $a \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow e$
  - $a \rightarrow f \rightarrow c \rightarrow b \rightarrow d \rightarrow e$

- 33
- 28
- 26
- 40
- 34
- 48
- 40
- 23
- 64
- 37

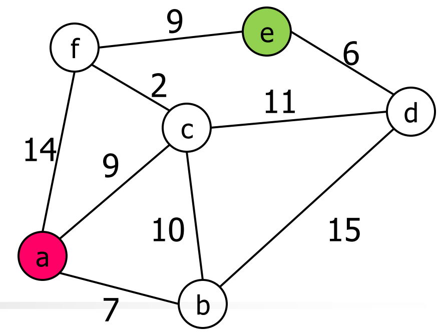
Improvement



- Consider this graph.
- Search for paths starting with fewer steps.

$a \to f \to e$	33
$a \to b \to d \to e$	28
$a \to c \to d \to e$	26
$a \to c \to f \to e$	40
Stop here?	
$a \to b \to c \to d \to e$	39
$a \to b \to c \to f \to e$	53
$a \to c \to b \to d \to e$	25
$a \to f \to c \to d \to e$	23
Stop here?	
$a \to b \to d \to c \to f \to e$	64
$a \to f \to c \to b \to d \to e$	22

Improvement



- We may miss the correct answer if not trying out all possible paths.
  - We cannot afford to try all possible paths in larger graph.
  - We need a more clever and systematic approach.
- Possible approach:
  - Look at edges and use them to improve on existing paths.
  - An edge is said to lead to improvement, if passing through it would lead to a better path.
  - Example:
    - Going from a to c directly, the distance is 9.
    - Going from a to f directly, the distance is 14.
    - Going from a to f via c, the distance improves to 11 < 14.
  - We will try to implement this idea.

End of Lecture 9