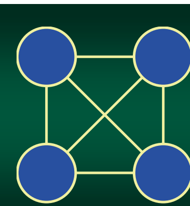
Graphs

Section 4.1 – 4.3

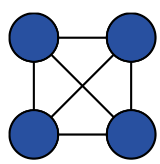
Introduction to Graphs



Nope, not the same as charts

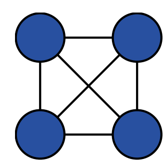
Graphs

- Lists and trees are just a special case of another structure - the *graph*
- Graphs are the basis for <u>all</u> of computer science



Graphs

- Computer science is not about chips, processors, etc...
- ... this is just implementation technology



Where are Graphs Used?

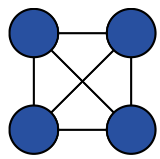
- The easy answer is: *everywhere*
- In computer science
  - state machines
  - mazes and networks
- Other fields
  - chemistry
  - physics
  - government

Motivation

- Several real-life problems can be converted to problems on graphs
- They are one of the pervasive data structures used in computer science
- They are useful tool for modeling real-world problems
- Allows us to abstract details and focus on the problem

Terminology

- The terminology for graphs is a bit different from trees and linked lists
- Rather, it is more generalized
  - nodes are called *vertices*
  - branches are called *edges*

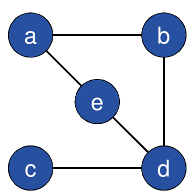


Formal Definition

- A graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ is defined by a pair of two sets
  - a <u>finite</u> set $\boldsymbol{V}$ of items called *vertices*
  - a <u>finite</u> set $\boldsymbol{E}$ of vertex ordered-pairs called *edges*

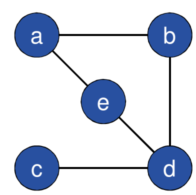
Example Graph

- Set of vertices $\boldsymbol{V}$
  - a
  - b
  - c
  - d
  - e



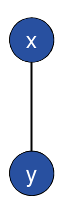
Example Graph

- Set of edges $\boldsymbol{E}$
  - (a, b)
  - (a, e)
  - (b, d)
  - (c, d)
  - (d, e)



Adjacent and Incident

- When two vertices x and y share an edge (x, y)
  - they are said to be *adjacent*
  - in other words, they are connected
- The edge (x, y)
  - is called *incident* on vertices x and y
  - in other words, it is the connection



Directed Graph

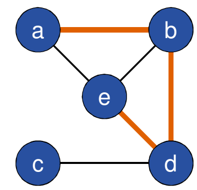
- A *directed graph (digraph)* is a graph where each edge has a source and target vertex
- This is the basis most of the data structures used today:
  - trees
  - linked lists

Undirected Graphs

- Undirected graphs have edges that link both vertices together
- So, the edge has the same meaning for both directions (set rather than tuple)
- Examples:
  - mathematical equality: if a = b then b = a
  - marriage: if Jane is married to Joe, Joe is married to Jane

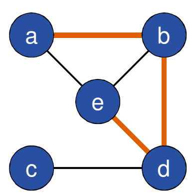
Paths

- A *path* is a sequence of vertices $v_1, v_2, . . . v_k$ such that consecutive vertices v_n and v_{n+1} are adjacent
- This can represent
  - a physical path
  - logical connection
  - etc...



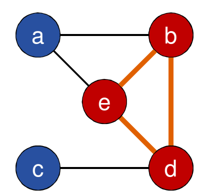
Paths

- In a *simple path*, all edges of a path are distinct
- The *length* of a path is measured by either the total number of edges or vertices



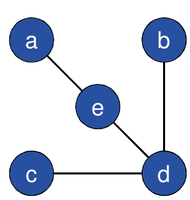
Cycles

- Unlike linked lists, graphs can have loops
- A *cycle* is a sequence of vertices $v_1, v_2, . . . v_k$ such that consecutive vertices v_n and v_{n+1} are adjacent and $v_1 = v_k$



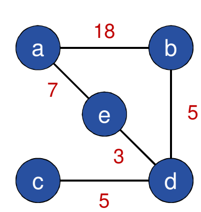
Cycles

- An *acyclic graph* contains no cycles
- An acyclic directed graph is often called a *DAG* (Directed Acyclic Graph)



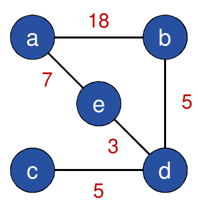
Weighted Graphs

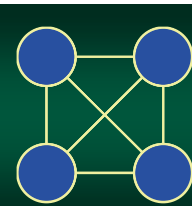
- *Weighted graph* has values on each edge
- These values can be anything – and are defined by the ADT using the graph



Weighted Graph

- Typical uses
  - networks – for finding the fastest speed
  - driving – fastest route
  - etc...
- Example:
  - minimum path from a to b is: a → e → d → b





Types of Graphs

Broad categories for a broad topic

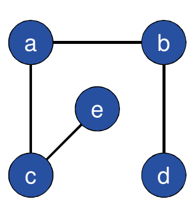
Connected and Unconnected

- A *connected* graph
  - has a path from every vertex to all other vertex
  - so, everything is connected somehow
- An *unconnected* graph
  - at least one vertex exists in which no path exists to another vertex
  - so, there are 2+ sub-graphs that are unlinked

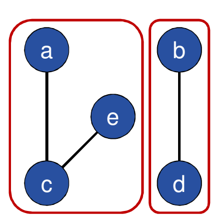
Connected and Unconnected

- The *connected component* is the maximum connected subgraph of a given graph
- If the graph is connected, then the whole graph is one single connected component

Connected and Unconnected



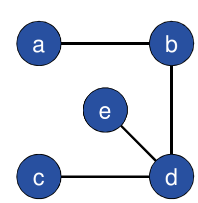
Connected



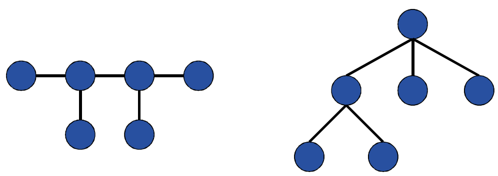
Unconnected

Trees

- *Tree* is defined as a connected acyclic graph
- *Rooted Tree* selects an arbitrary vertex and considers it as the root
- *Forest* is a collection of trees

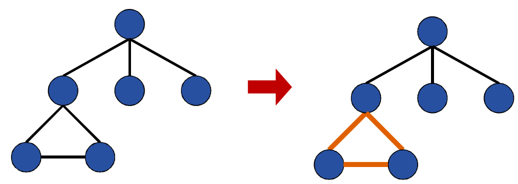


Trees



Tree   Same Tree

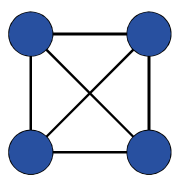
Trees



NOT a Tree

Complete Graphs

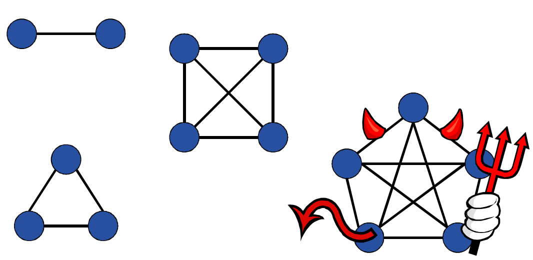
- A *complete graph* is one in which all pairs of vertices are adjacent

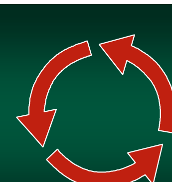


Complete Graphs

- The # of edges in a complete graph...
  - if n is the total number of vertices, each vertex is incident to $n - 1$ edges
  - we can compute $n \times (n - 1)$ edges, but this would count each edge twice!
  - so, the number of edges = $n \times (n - 1) / 2$
- For a noncomplete graph...
  - number of edges < $n \times (n - 1) / 2$

Example Complete Graphs





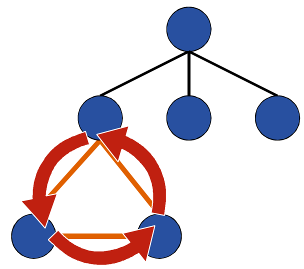
Graph Traversal

Not as easy as trees

Graph Traversal

- Typically when you search a tree, you can use a simple depth-first search
- However, graphs can contain cycles
- Your program can get stuck in an graph loop and never escape
- This has to be taken into account

Graph Traversal

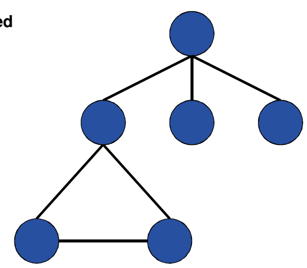


Graph Traversal

- So, to use either depth-first or breadth-first...
  - the vertices need to visited only <u>ONCE</u>
  - otherwise, we have a loop
- *Solution:* each vertex has a "visited" property
  - before the search, each vertex is set to false
  - when the search visits them, it is set true
  - search never follows an edge to a visited vertex

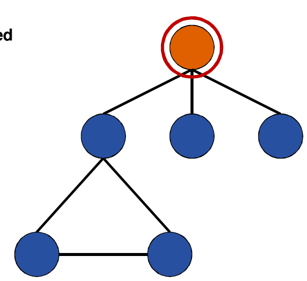
Graph Traversal

Unvisited
Visited



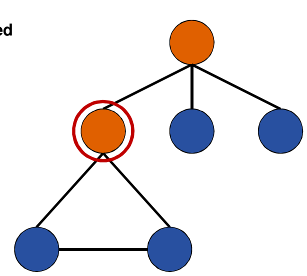
Graph Traversal

Unvisited
Visited



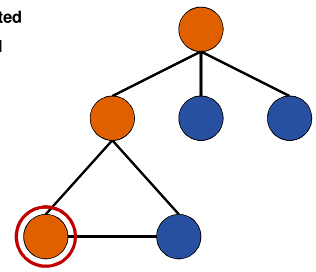
Graph Traversal

Unvisited
Visited



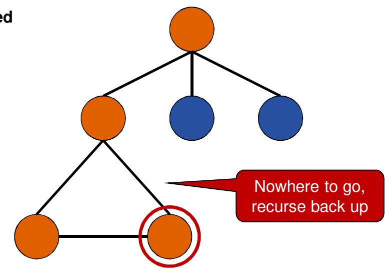
Graph Traversal

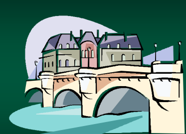
- Unvisited
- Visited



Graph Traversal

- Unvisited
- Visited





Birth of Graph Theory

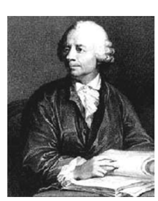
... and, later, computer science!

Birth of Graph Theory

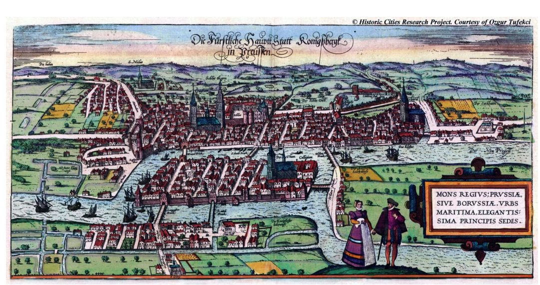
- Graph theory was created due to a perplexing question troubling mathematician *Leonhard Euler*
- Lived in Konigsberg
  - now "Kaliningrad" in Russia)
  - city occupied 2 islands plus areas on both banks
  - 7 bridges over the Pregel River

Birth of Graph Theory

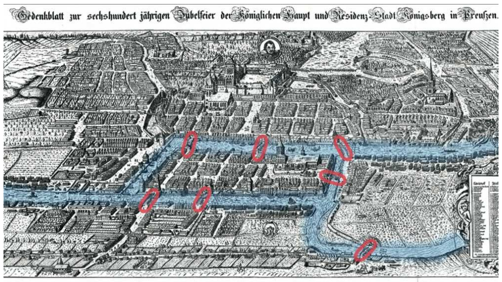
- People wondered if they could leave home…
  - cross every bridge exactly once
  - and return to their starting point
  - This is known as the *Konigsberg Bridge Problem*



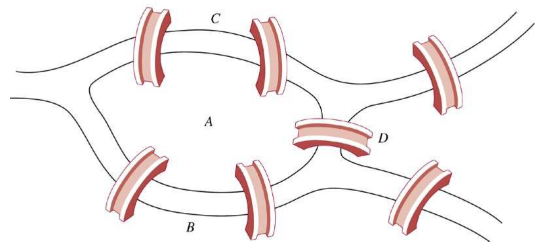
Konigsberg Bridge Problem



Konigsberg Bridge Problem



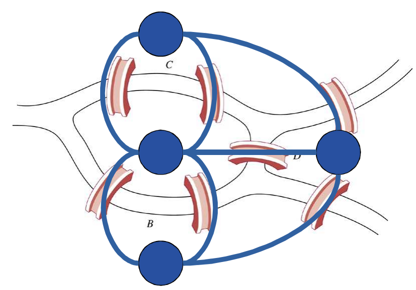
Konigsberg Bridge Problem



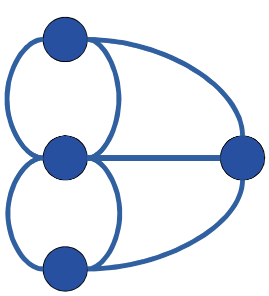
Konigsberg Bridge Problem

- The problem reduces down to a number of points and links to each point
- From this, Euler created the first graph and began the study of their properties

Konigsberg Bridge Problem



Konigsberg Bridge Problem



The Solution to Konigsberg

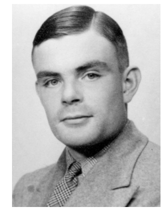
- In 1736, Euler proved that no such traversal exists
- From his work on Konigsberg, a path is named after him
- An *Eulerian circuit*, in a graph...
  - is cycle containing all the edges in the graph
  - and only traversing each edge once

The Solution to Konigsberg

- Euler proved:
  - a graph may have an Eulerian circuit <u>if and only if</u> there are no vertices with an odd number of edges touching them
- Konigsberg Bridge Problem
  - 4 vertices, all with an odd number of edges
  - Sorry people of Konigsberg, there is no solution!

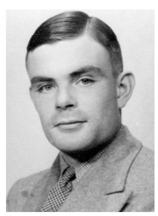
Alan Turing

- Mathematician, logician & cryptographer
- *Father of Computer Science*
  - Highest award in Computer Science is the Turing Award
  - Developed Turing Machines



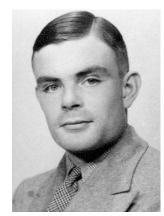
Major Work: Turing Machines

- Invented in 1937
- Logical model – not an actual computer or machine
- Based on 2 graphs (and sets on each of the edge)



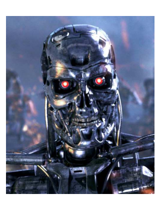
Major Work: Turing Machines

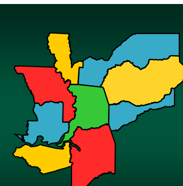
- One graph is simple array, but the other could be anything
- From this, he proved programming



Major Work: Turing Test

- Used in artificial intelligence
- Consists of a human operator *texting* a human <u>or</u> computer
- If the operator can't ascertain if it is a computer or human, the computer is "intelligent"
- No computer has passed it





Real World Examples

The origin and the usage

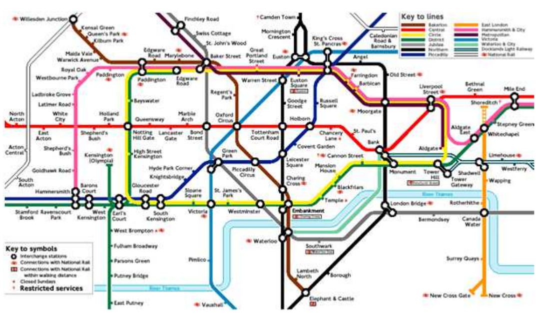
Real World Examples

- How many layers does a computer chip need so that wires in the same layer don't cross?
- How can the season of a sports league be scheduled into the minimum number of weeks?
- In what order should a traveling salesman visit cities to minimize travel time?
- Can we color the regions of every map using four colors so that neighboring regions receive different colors?

Real World Examples

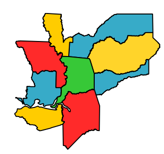
- How can we lay cable at minimum cost to make every network reachable from every other?
- What is the fastest route from the national capital to each state capital?
- How can *n* jobs be filled by *n* people with maximum total utility?

The London Underground Subway



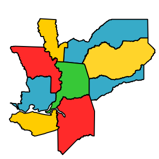
Four Color Theorem

- The four color theorem is used extensively in map making – as well as other fields
- The four color theorem states
  - given any plane cut into contiguous regions – called a map
  - regions can be colored using at most four colors
  - so that no two adjacent regions have the same color



Four Color Theorem

- Two regions are adjacent...
  - only if they share a border segment
  - not just a point



Four Color Theorem Proved

- The four color theorem was proven in 1976 by Kenneth Appel and Wolfgang Haken
- It was proven using a computer
  - started by showing with a set of 1,936 maps
  - each of which cannot be part of a smallest-sized example to the four color theorem
  - by proving these maps, they proved smaller maps

Four Color Theorem Proved

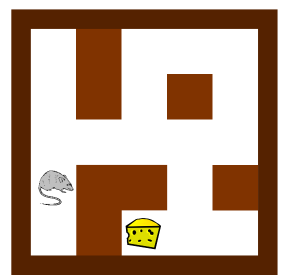
- At first, their proof was rejected
  - mathematicians thought a computer-assisted proof as infeasible
  - to complex for a human to check by hand
  - but the proof has gained wider acceptance, although doubts remain

Maze Traversal

- One example of where a graph is useful is a maze traversal
- Basically, any maze can be represented with a graph
- ... and this is not so much different to how networks actually work
- ... a source must find a destination through various vertices

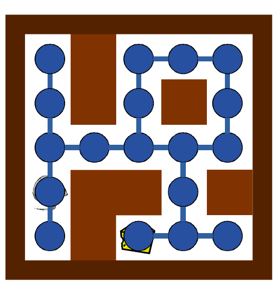
Maze Traversal

- This is a simple maze – though not to the mouse!
- We can help him find the cheese if we convert this to a graph



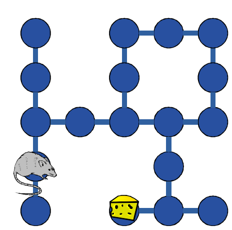
Maze Traversal

- The empty spaces are vertices
- The bordering ones are connected with an edge
- Is this a directed graph?



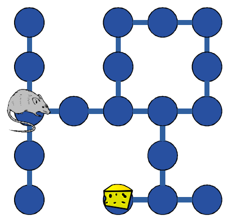
Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order



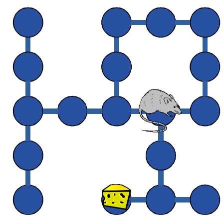
Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order



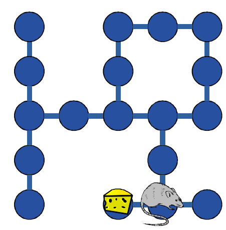
Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order



Maze Traversal

- So, to help the mouse, we can get to depth-first search on the maze
- If we find it, we can print off the vertices post-order

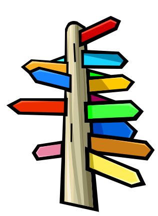


Dijkstra's Algorithm

Finding the best route!

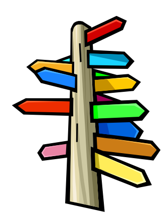
Getting the Shortest Path

- Weighted graphs have edges with numerical values
- These weights are abstract and can represent *anything*
  - distances – driving, flight routes
  - costs
  - server latency times
  - etc...



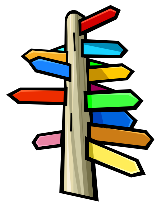
Getting the Shortest Path

- It is useful to find the "best" path through the graph
- This is known as the *distance*
- Given 2 vertices u and v ...
  - we want to find a path with the "best" total weight between u and v
  - the total is the sum of all the edges in the path



Some Real World Examples

- Internet packet routing
  - fastest route – faster downloads
  - load distribution
- Driving directions
  - shortest route in miles
  - fastest route – given traffic and speed limit data

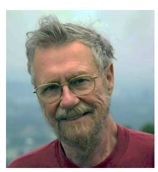


Shortest Path Attributes

- A sub-path of a shortest path
  - … is itself a shortest path
  - so, *any subset of a optimal solution is optimal*
- There is a tree of shortest paths
  - there are multiple solutions from the start vertex to all the other vertices
  - might be multiple equally-optimal solutions

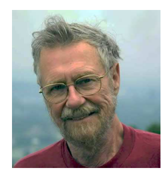
Edsger Dijkstra

- Edsger Dijkstra is a World famous computer scientist
- He authored numerous algorithms we use today
- For his contributions, he received the Turing Award



Edsger Dijkstra

- The algorithm used today to find the optimal path was written by him
- In fact, his algorithm runs the Internet – a self healing, load distribution network
- So, no Dijkstra → no Internet



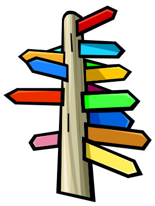
Dijkstra's Algorithm

- Dijkstra’s algorithm computes the distances of all the vertices from a given start vertex s
- Works on both directed and undirected graphs
- However
  - all edges must have nonnegative weight
  - the graph is connected

Dijkstra's Algorithm

- Dijkstra's algorithm is *greedy*
  - always takes the best immediate or local solution while finding an answer
  - they find optimal solutions for some optimization problems very efficiently
  - but may find less-than-optimal solutions
- Greedy algorithms require extra analyzation to determine if they work – Dijkstra's does

Dijkstra's Algorithm Logic



- Each vertex has its own "distance" table
- The table contains the best weight to each other vertex - as well as the best path

Dijkstra's Algorithm Logic

1. Initialize a distance table
   - table represents the best distance (weight) to get to the specific vertices
   - set all to infinity (worst possible)
   - it will be updated as we see more vertices
2. Loop until all vertices are *settled* (visited)
   - depth-first search the graph
   - advance on smallest weighted edge

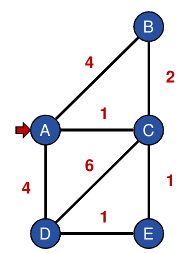
Dijkstra's Algorithm Logic

- For each vertex v
  - compute the total distance to get each vertex t that v is adjacent to (v edges point to t)
  - if the value is better than the current table value, update
- Given table D, vertex v, edge weight w with target t : $D(t) = D(v) + w$

Example 1: Vertex A Table

Vertex	Best Path	Best Weight	Current
A		0	
B		∞	
C		∞	
D		∞	
E		∞	

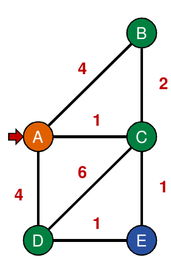
Stack	



Example 1: Look Adjacent

Vertex	Best Path	Best Weight	Current
A		0	
B		∞	0 + 4
C		∞	0 + 1
D		∞	0 + 4
E		∞	

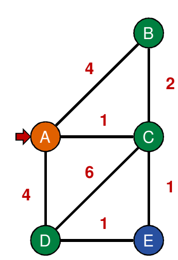
Stack	A



Example 1: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		0	
B	A	4	0 + 4
C	A	1	0 + 1
D	A	4	0 + 4
E		∞	

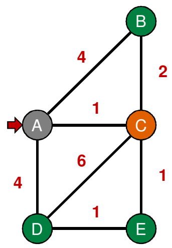
Stack	A



Example 1: Take Best Edge

Vertex	Best Path	Best Weight	Current
A		0	
B	A	4	1 + 2
C	A	1	
D	A	4	1 + 6
E		∞	1 + 1

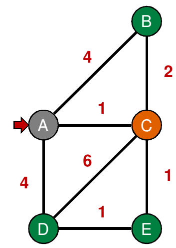
Stack	A C



Example 1: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		0	
B	A C	3	1 + 2
C	A	1	
D	A	4	1 + 6
E	A C	2	1 + 1

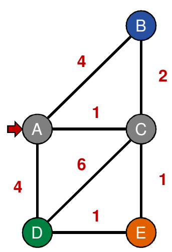
Stack	A C



Example 1: Take Best Edge

Vertex	Best Path	Best Weight	Current
A		0	
B	A C	3	
C	A	1	
D	A	4	1 + 2
E	A C	2	

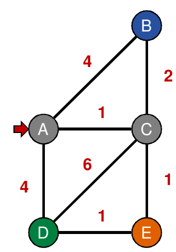
Stack	A C E



Example 1: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		0	
B	A C	3	
C	A	1	
D	A C E	3	1 + 2
E	A C	2	

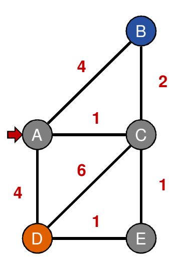
Stack	A C E



Example 1: Take Best Edge

Vertex	Best Path	Best Weight	Current
A		0	
B	A C	3	
C	A	1	
D	A C E	3	
E	A C	2	

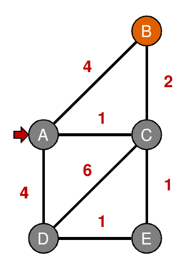
Stack	A C E D



Example 1: Recurse up, try B

Vertex	Best Path	Best Weight	Current
A		0	
B	A C	3	
C	A	1	
D	A C E	3	
E	A C	2	

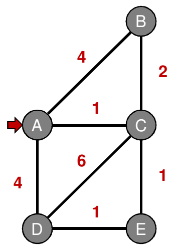
Stack	A C B



Example 1: Done

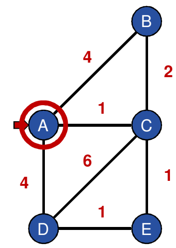
Vertex	Best Path	Best Weight	Current
A		0	
B	A C	3	
C	A	1	
D	A C E	3	
E	A C	2	

Stack	



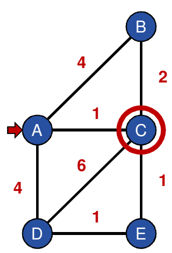
Example 1: Solution A → D

Vertex	Best Path	Best Weight
A		0
B	A C	3
C	A	1
D	A C E	3
E	A C	2



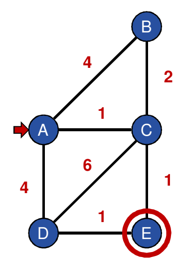
Example 1: Solution A → D

Vertex	Best Path	Best Weight
A		0
B	A C	3
C	A	1
D	A C E	3
E	A C	2



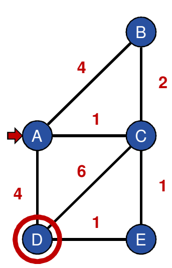
Example 1: Solution A → D

Vertex	Best Path	Best Weight
A		0
B	A C	3
C	A	1
D	A C E	3
E	A C	2



Example 1: Solution A → D

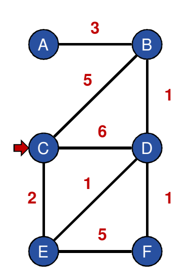
Vertex	Best Path	Best Weight
A		0
B	A C	3
C	A	1
D	A C E	3
E	A C	2



Example 2: Vertex C Table

Vertex	Best Path	Best Weight	Current
A		∞	
B		∞	
C		0	
D		∞	
E		∞	
F		∞	

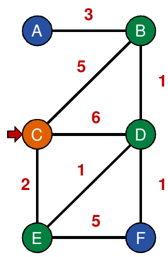
Stack	



Example 2: Look Adjacent

Vertex	Best Path	Best Weight	Current
A		∞	
B		∞	0 + 5
C		0	
D		∞	0 + 6
E		∞	0 + 2
F		∞	

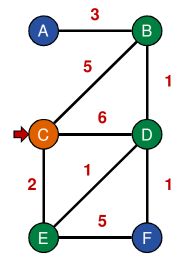
Stack	C



Example 2: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		∞	
B	C	5	0 + 5
C		0	
D	C	6	0 + 6
E	C	2	0 + 2
F		∞	

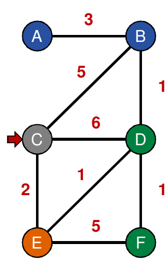
Stack	C



Example 2: Take Best Edge

Vertex	Best Path	Best Weight	Current
A		∞	
B	C	5	
C		0	
D	C	6	2 + 1
E	C	2	
F		∞	2 + 5

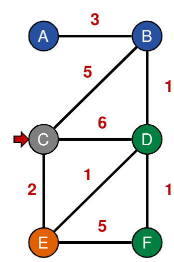
Stack	C E



Example 2: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		∞	
B	C	5	
C		0	
D	C E	3	2 + 1
E	C	2	
F	C E	7	2 + 5

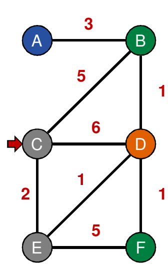
Stack	C E



Example 2: Take Best Edge

Vertex	Best Path	Best Weight	Current
A		∞	
B	C	5	3 + 1
C		0	
D	C E	3	
E	C	2	
F	C E	7	3 + 1

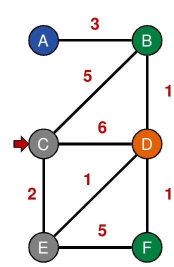
Stack	C E D



Example 2: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		∞	
B	C E D	4	3 + 1
C		0	
D	C E	3	
E	C	2	
F	C E D	4	3 + 1

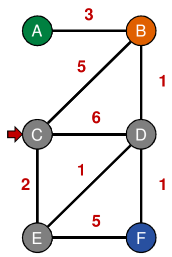
Stack	C E D



Example 2: Take Best Edge

Vertex	Best Path	Best Weight	Current
A		∞	4 + 3
B	C E D	4	
C		0	
D	C E	3	
E	C	2	
F	C E D	4	

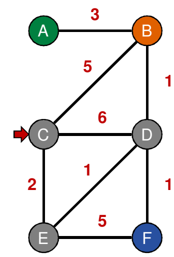
Stack	C E D B



Example 2: Set Best Weight/Path

Vertex	Best Path	Best Weight	Current
A		7	4 + 3
B	C E D	4	
C		0	
D	C E	3	
E	C	2	
F	C E D	4	

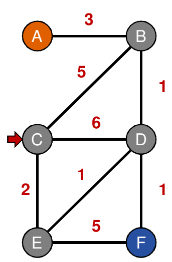
Stack	C E D B



Example 2: Take Best Path

Vertex	Best Path	Best Weight	Current
A	C E D B	7	
B	C E D	4	
C		0	
D	C E	3	
E	C	2	
F	C E D	4	

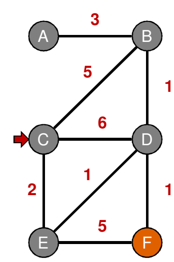
Stack	C E D B



Example 2: Recursed Up

Vertex	Best Path	Best Weight	Current
A	C E D B	7	
B	C E D	4	
C		0	
D	C E	3	
E	C	2	
F	C E D	4	

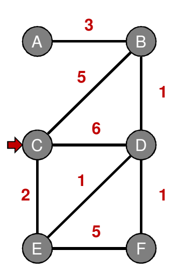
Stack	C E D F



Example 2: Done

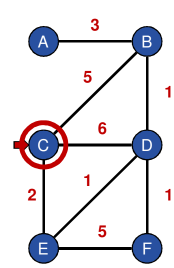
Vertex	Best Path	Best Weight	Current
A	C E D B	7	
B	C E D	4	
C		0	
D	C E	3	
E	C	2	
F	C E D	4	

Stack	



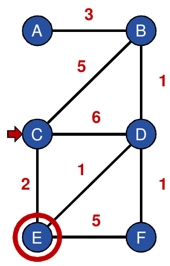
Example 2 Solution: C → F

Vertex	Best Path	Best Weight
A	C E D B	7
B	C E D	4
C		0
D	C E	3
E	C	2
F	C E D	4



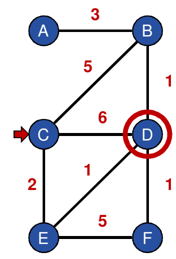
Example 2 Solution: C → F

Vertex	Best Path	Best Weight
A	C E D B	7
B	C E D	4
C		0
D	C E	3
E	C	2
F	C E D	4



Example 2 Solution: C → F

Vertex	Best Path	Best Weight
A	C E D B	7
B	C E D	4
C		0
D	C E	3
E	C	2
F	C E D	4



Example 2 Solution: C → F

Vertex	Best Path	Best Weight
A	C E D B	7
B	C E D	4
C		0
D	C E	3
E	C	2
F	C E D	4