Week 09 Graphs SSSP for weighted and unweighted graphs

MIDS 261

Notebook:

https://www.dropbox.com/s/zjm0ka9jr7yvlih/demo9_workbook_MASTER_2023_07_04.ipynb?dl=0

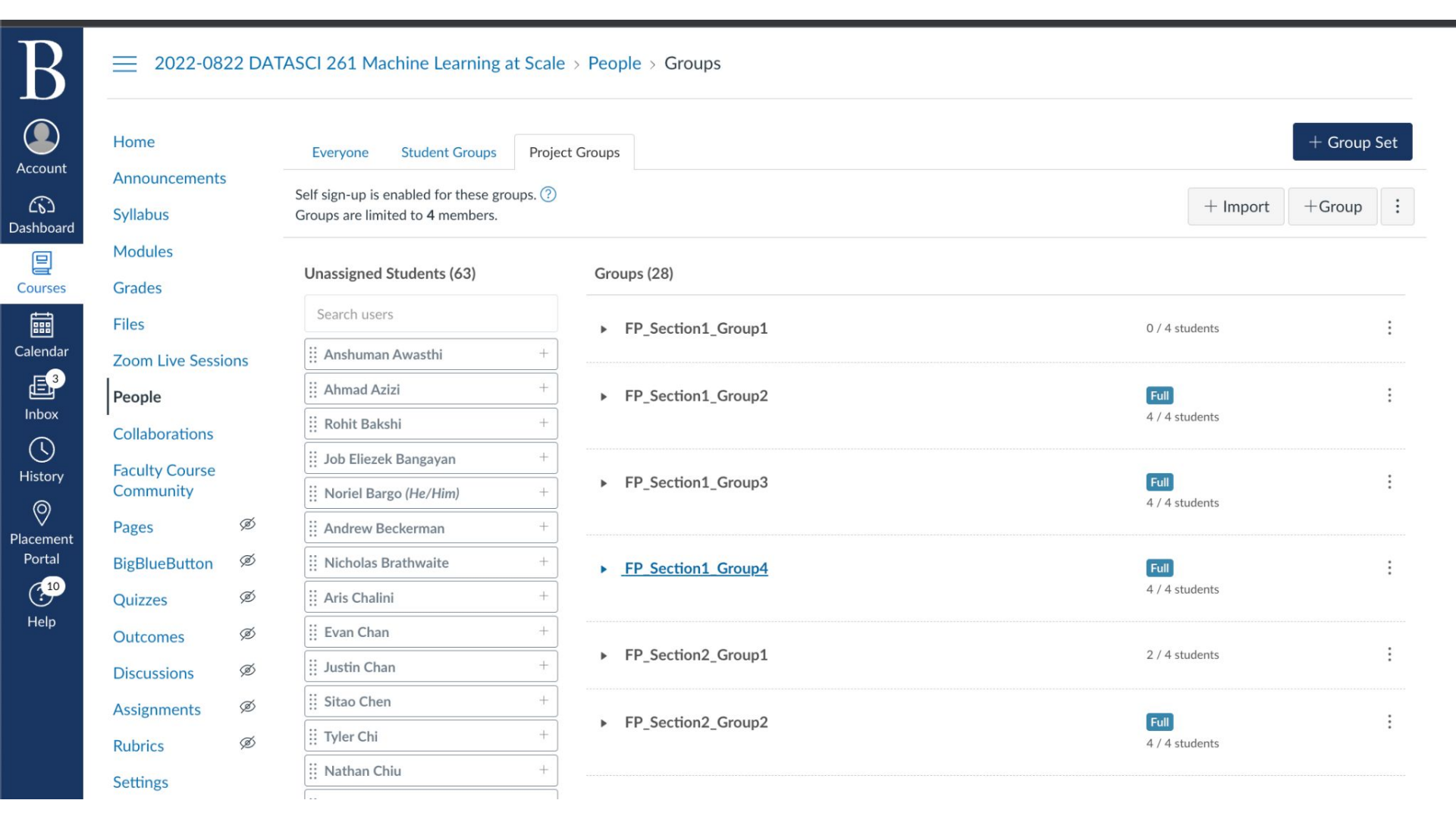
Housekeeping

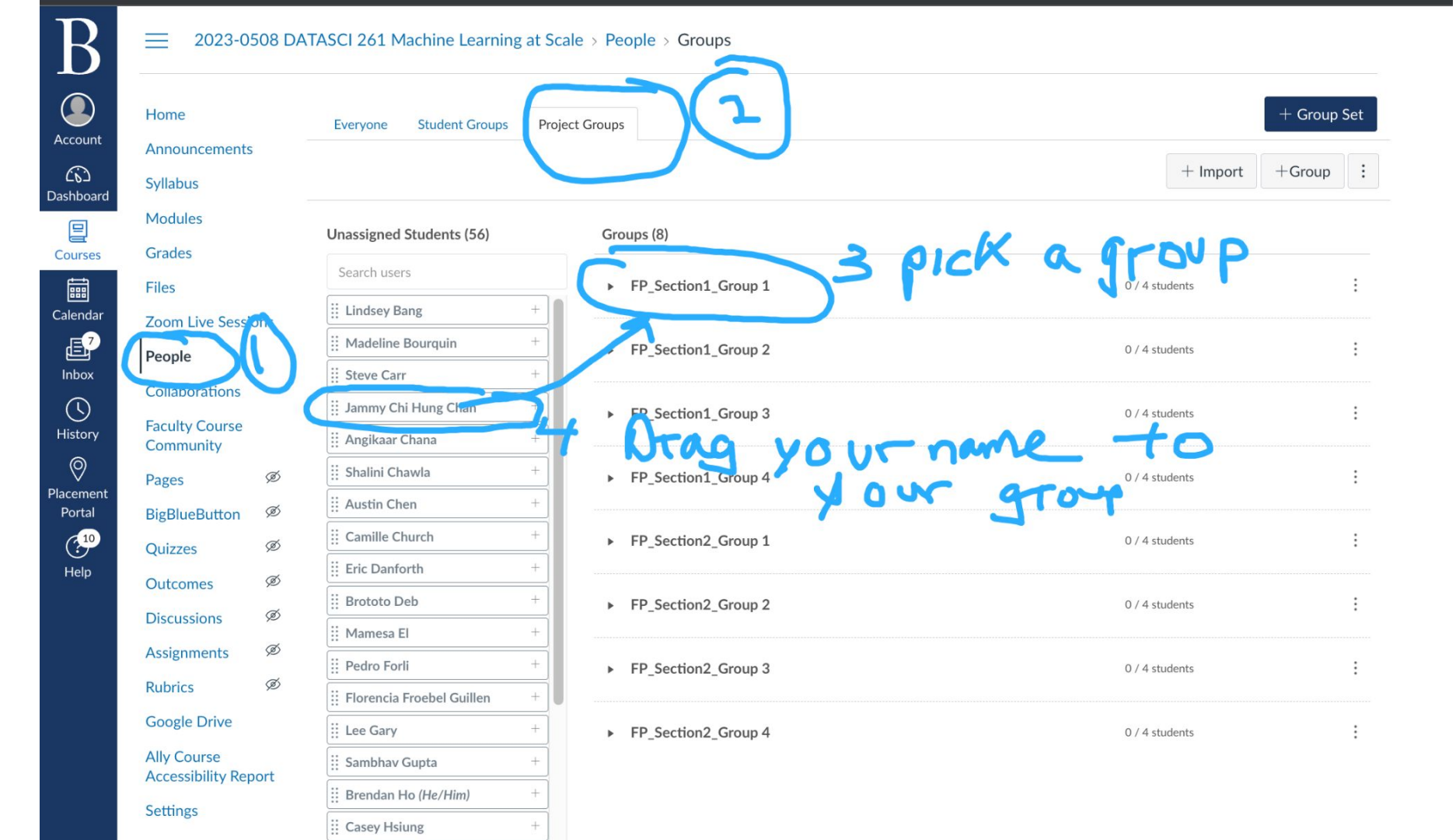
- Once teams are formed, everyone needs to fill in their team details on DC under the **"People"** section. You can join a group by adding your nam to one of the final project groups in DC. Click the **"People"** TAB on the left-hand navigation menu, and select the **"FP_SectionK_GroupN"** tab to view available groups. Each group is limited to a max of four students inside DC.
- Please see here to learn more about joining a group in DC (do not create your own student groups, you must join a pre-setup group created by your Instructors for this project).
- Please use the team following team name prefix, *FP_SectionK_**GroupN**_* as is but feel free to add a team suffix such as *RootFinders* to yield a tean name of *FP_Section1_Group2-RootFinders; when K denotes the section you belong to and N denotes a unique group number with your section.*

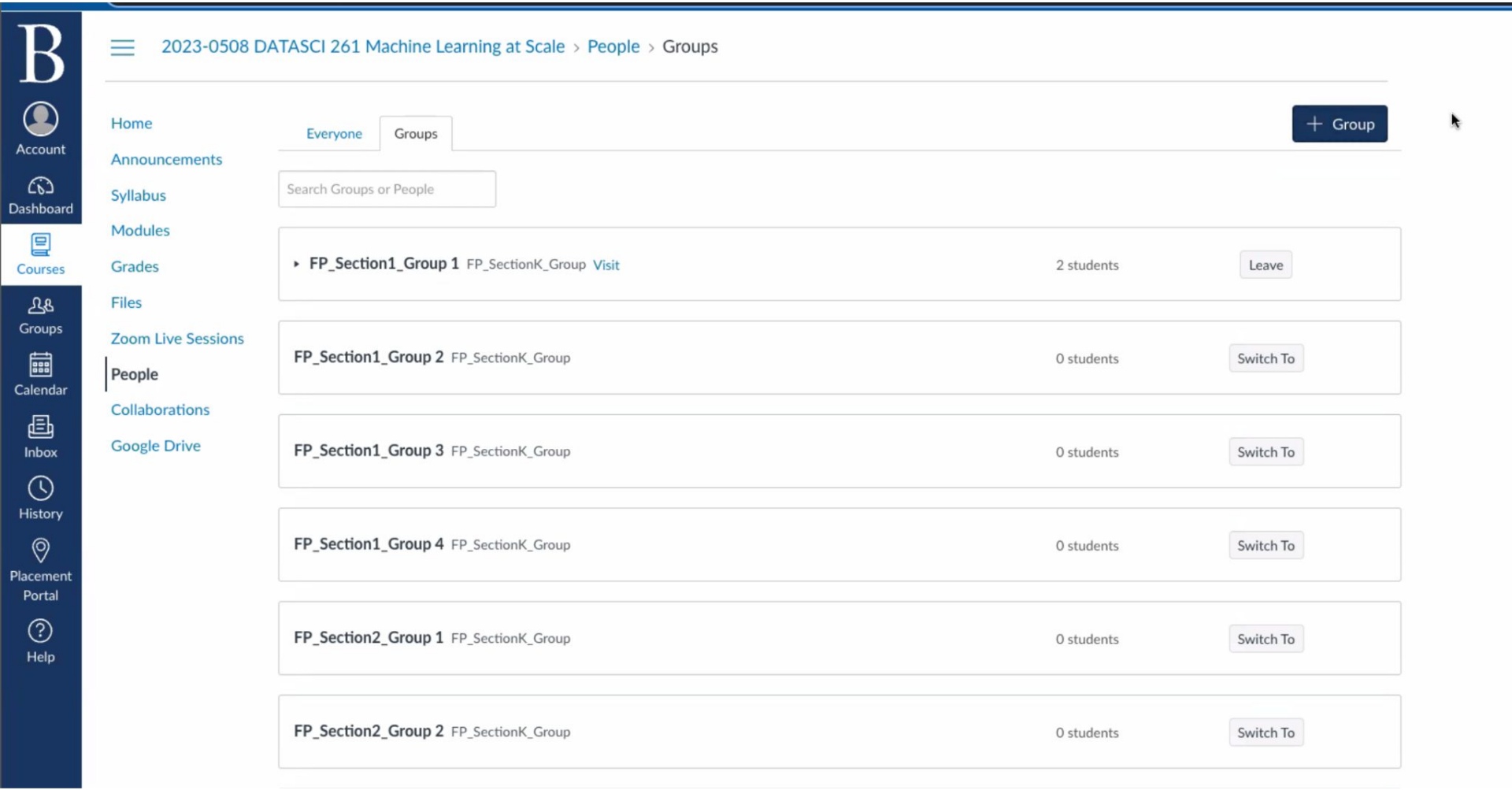
For more details on the final project, please see Module - Final Project - Flight Delays.

Schedule at a glance

Phase Description (For details please see detailed phase descriptions and deliverables)	**Submission details (or just see *Module - Final Project - Flight Delays*)**	**Week number**	**DataBricks cluster details**
Phase 0: Finalize Teams (teams of four people)	Due Sunday of Week 9	Week 9	Available from Monday/Tues week 10 (One master and one worker)
Phase I - project plan, describe datasets, joins, tasks, and metrics	Phase 1 Update: Notebook. Due Sunday of Week 10	Week 10	
Phase II - EDA, Baseline Pipeline on all data: Scalability, Efficiency, Distributed/parallel Training, and Scoring Pipeline	Phase 2 Update: Video (2 minutes max) + Notebook. Due Sunday of Week 11	Week 11	AutoScaling enabled from Monday of this week (up to 5)
Phase III Feature engineering + hyperparameter tuning, + **in-class review**	Phase 3 update: Notebook. Due by Sunday of Week 12	Week 12	
Phase IV - Advanced model architectures and loss functions, select an optimal algorithm, fine-tune & Final report write up	Phase 4 update: 2-minute Video + Notebook due by Sunday of Week 13	Week 13	AutoScaling enabled from Monday of this week (up to 10 depending on budget) NOTE: Clusters will be scaled back on Sunday of this week.
Phase V - **In-class Presentation**	Slides uploaded prior to in-class presentation.	Week 14	


2022-0822 DATASCI 261 Machine Learning at Scale › People › Groups
Account
Dashboard
Courses
Calendar
Inbox
History
Placement Portal
Help
Home
Announcements
Syllabus
Modules
Grades
Files
Zoom Live Sessions
People
Collaborations
Faculty Course Community
Pages
BigBlueButton
Quizzes
Outcomes
Discussions
Assignments
Rubrics
Settings
Everyone
Student Groups
Project Groups
+ Group Set
Self sign-up is enabled for these groups.
Groups are limited to 4 members.
+ Import
+ Group
Unassigned Students (63)
Search users
Anshuman Awasthi
Ahmad Azizi
Rohit Bakshi
Job Eliezek Bangayan
Noriel Bargo (He/Him)
Andrew Beckerman
Nicholas Brathwaite
Aris Chalini
Evan Chan
Justin Chan
Sitao Chen
Tyler Chi
Nathan Chiu
Groups (28)
FP_Section1_Group1
0 / 4 students
FP_Section1_Group2
Full
4 / 4 students
FP_Section1_Group3
Full
4 / 4 students
FP_Section1_Group4
Full
4 / 4 students
FP_Section2_Group1
2 / 4 students
FP_Section2_Group2
Full
4 / 4 students


2023-0508 DATASCI 261 Machine Learning at Scale › People › Groups
Account
Dashboard
Courses
Calendar
Inbox
History
Placement Portal
Help
Home
Announcements
Syllabus
Modules
Grades
Files
Zoom Live Sessions
People
Collaborations
Faculty Course Community
Pages
BigBlueButton
Quizzes
Outcomes
Discussions
Assignments
Rubrics
Google Drive
Ally Course Accessibility Report
Settings
Everyone
Student Groups
Project Groups
+ Group Set
+ Import
+ Group
Unassigned Students (56)
Search users
Lindsey Bang
Madeline Bourquin
Steve Carr
Jammy Chi Hung Chan
Angikaar Chana
Shalini Chawla
Austin Chen
Camille Church
Eric Danforth
Brototo Deb
Mamesa El
Pedro Forli
Florencia Froebel Guillen
Lee Gary
Sambhav Gupta
Brendan Ho (He/Him)
Casey Hsiung
Groups (8)
FP_Section1_Group 1
FP_Section1_Group 2
FP_Section1_Group 3
FP_Section1_Group 4
FP_Section2_Group 1
FP_Section2_Group 2
FP_Section2_Group 3
FP_Section2_Group 4
0 / 4 students
1
2
3 pick a group
4 Drag your name to your group


2023-0508 DATASCI 261 Machine Learning at Scale › People › Groups
Account
Dashboard
Courses
Groups
Calendar
Inbox
History
Placement Portal
Help
Home
Announcements
Syllabus
Modules
Grades
Files
Zoom Live Sessions
People
Collaborations
Google Drive
Everyone
Groups
+ Group
Search Groups or People
FP_Section1_Group 1 FP_SectionK_Group Visit
2 students
Leave
FP_Section1_Group 2 FP_SectionK_Group
0 students
Switch To
FP_Section1_Group 3 FP_SectionK_Group
0 students
Switch To
FP_Section1_Group 4 FP_SectionK_Group
0 students
Switch To
FP_Section2_Group 1 FP_SectionK_Group
0 students
Switch To
FP_Section2_Group 2 FP_SectionK_Group
0 students
Switch To

Quick review

9: Graph Algorithms at Scale

- Week 09 Async Slides Combined.pdf
- Weekly Introduction 9
- Networks Introduction and Motivation
- Applications Graphs
- Graph Definitions
- Shortest Path Introduction
- Single-Source Shortest-Path Unweighted Graphs
- BFS for Unweighted Graphs: Directed and Animations
- BFS for Unweighted Graphs: Undirected and Animations
- SSSP for Weighted Graphs BFS
- Quiz 9-1
  - Quiz
  - Question: Quiz 9-1
  - Quiz
- SSSP for Weighted Graphs: Dijkstra's Algorithm
- Distributed SSSP: Unweighted Graph
- Distributed SSSP: Unweighted Graph Algorithm
- Distributed SSSP: Weighted Graph
- Question: Quiz 9-2
- HW5 - PageRank
  Jul 23 | 100 pts

Some common problems

Traverse all the vertices in a graph (Keys and Rooms in LeetCode)

Whether two vertices are connected or not (Division Evaluation)

Shortest path between two vertices (Shortest Path in Binary Matrix, Word Ladder I & II, Walls and Gates)

Number of connected components (Number of Islands)

Dependency path (Course Schedule II, Alien Dictionary)

Vertex grouping (Is Graph Bipartite?)

Detect cycle in a graph (Course Schedule I, Valid Tree Graph)

All possible paths (Word Search, Binary Tree Paths)

Leetcode

841. Keys and Rooms

Medium 550 44 Favorite Share

There are `N` rooms and you start in room `0`. Each room has a distinct number in `0, 1, 2, ..., N-1`, and each room may have some keys to access the next room.

Formally, each room `i` has a list of keys `rooms[i]`, and each key `rooms[i][j]` is an integer in `[0, 1, ..., N-1]` where `N = rooms.length`. A key `rooms[i][j] = v` opens the room with number `v`.

Initially, all the rooms start locked (except for room `0`).

You can walk back and forth between rooms freely.

Return `true` if and only if you can enter every room.

Example 1:

Room 0 1 2 3

```
Input: [[1],[2],[3],[]]
Output: true
Explanation:
We start in room 0, and pick up key 1.
We then go to room 1, and pick up key 2.
We then go to room 2, and pick up key 3.
We then go to room 3.  Since we were able to go to every room, we return true.
```

Example 2:

```
Input: [[1,3],[3,0,1],[2],[0]]
Output: false
Explanation: We can't enter the room with number 2.
```

127. Word Ladder

Medium 1749 882 Favorite Share

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

```
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
```

269. Alien Dictionary

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of words from the dictionary, where words are sorted lexicographically by the rules of this new language. Derive the order of letters in this language.

For example, Given the following words in dictionary,

```
[
  "wrt",
  "wrf",
  "er",
  "ett",
  "rftt"
]
```

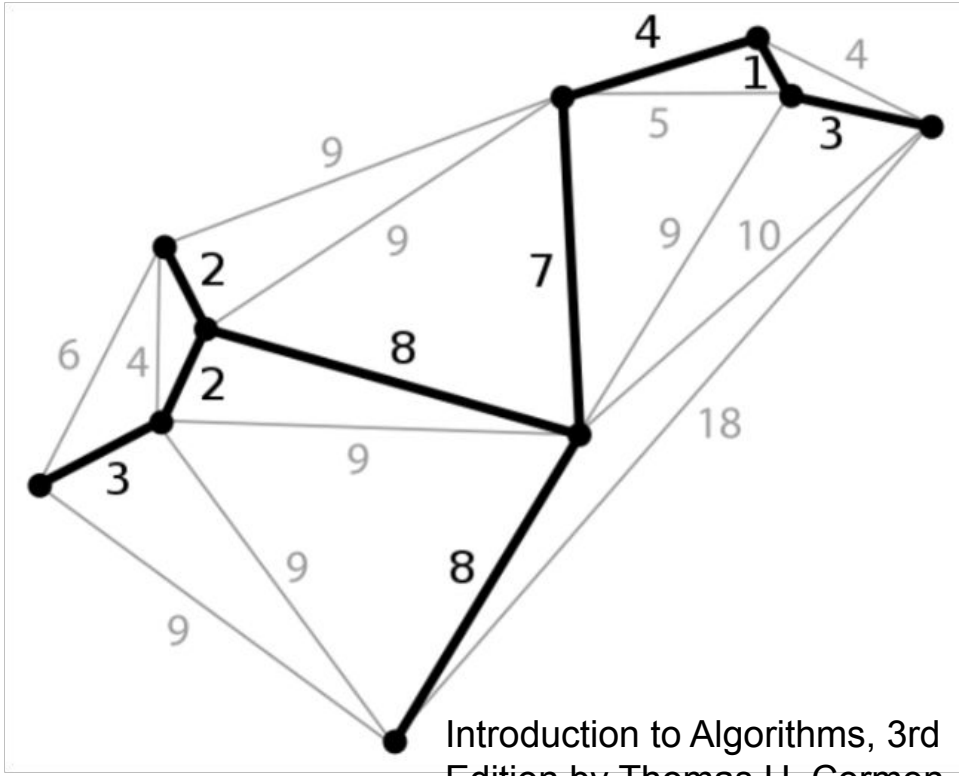
The correct order is: "wertf".

Note: You may assume all letters are in lowercase. If the order is invalid, return an empty string. There may be multiple valid order of letters, return any one of them is fine.

Minimum Spanning Tree

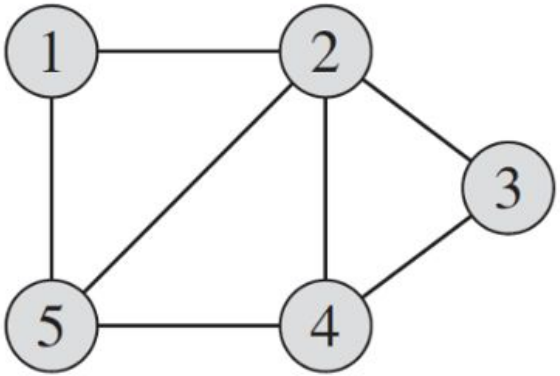
MST is a subset of the edges that connects all the vertices of a connected, edge-weighted undirected graph, without any cycles and with the minimum possible total edge weight.

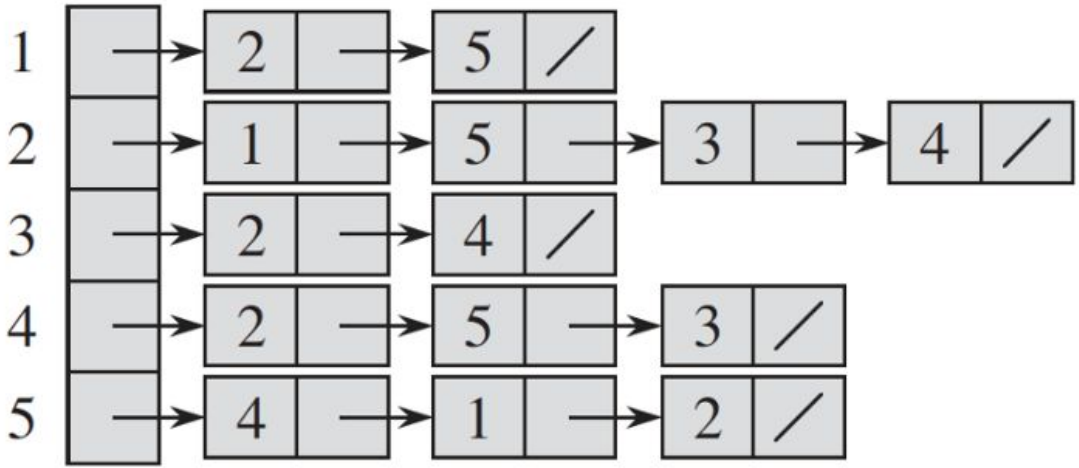
Telco/Cable/Pipeline problem



Introduction to Algorithms, 3rd Edition by Thomas H. Cormen, Charles E. Leiserso, Ronald L. Rivest, Clifford Stein

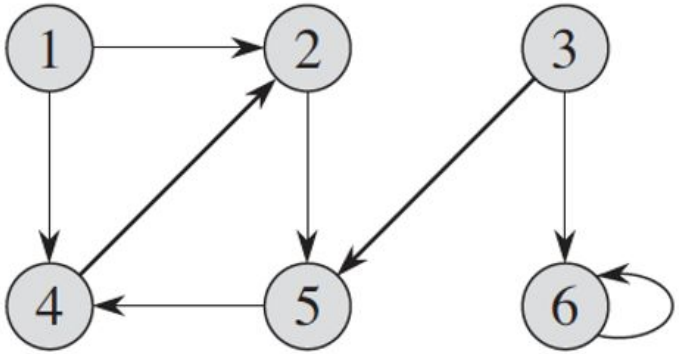
Adjacency List (Undirected Graph)

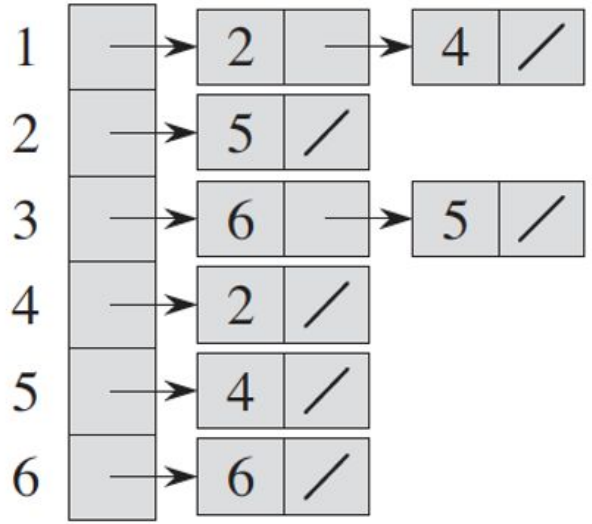




Introduction to Algorithms, 3rd Edition by Thomas H. Cormen, Charles E. Leiserso, Ronald L. Rivest, Clifford Stein

Adjacency List (directed Graph)





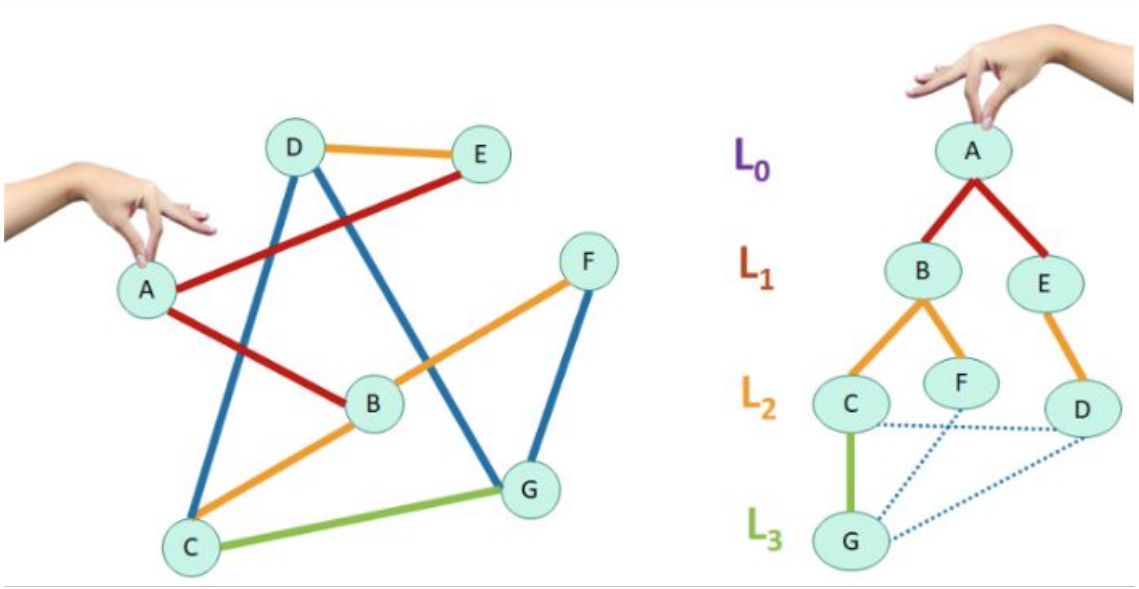
Introduction to Algorithms, 3rd Edition by Thomas H. Cormen, Charles E. Leiserso, Ronald L. Rivest, Clifford Stein

BFS Traversal of a graph

BFS shorted path for unweighted graphs

BFS traverses by level

Number of levels is also the shortest distance between two vertices

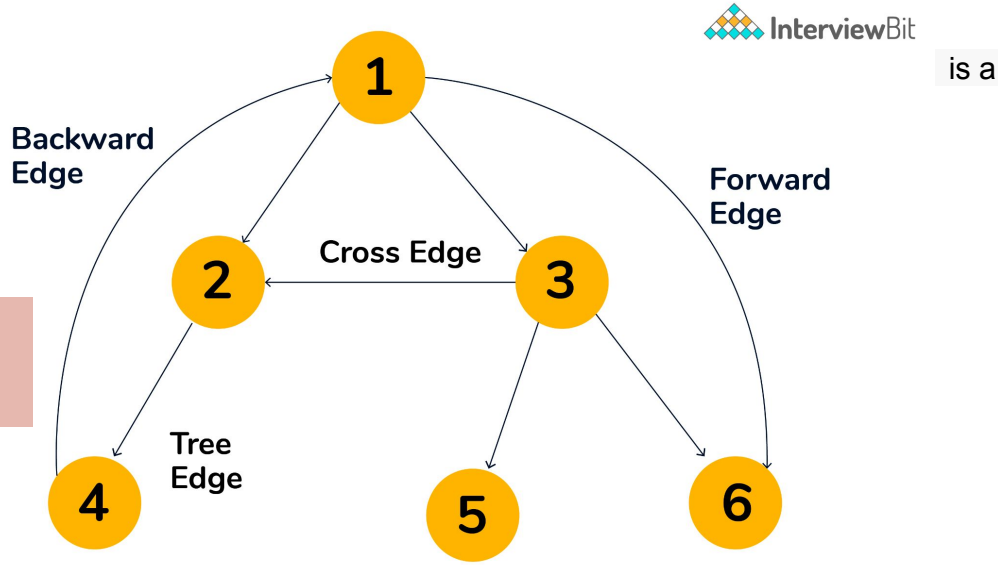


Original Graph

Visualized BFS Traversal

Cross edges, back edges (loops in graphs)

- Tree Edge: The edge which is present in the tree obtained after applying DFS on the graph. All the Green edges are tree edges.
- Forward Edge: `Edge (u, v)` such that v is descendant but not part of the DFS tree. Edge from node 1 to node 6 is a forward edge.
- Back edge:
  - `Edge (u, v)` such that `v` is ances is a back edge.
  - Presence of back edge indicates a
- Cross Edge: It is a edge which connects tw relationship between them. Edge from nod



Visited arrays prevents loops in BFS traversal and SSSP for unweighted graphs

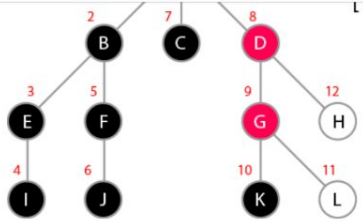
https://www.interviewbit.com/courses/programming/graph-data-structure-algorithms/breadth-first-search/

DEMO9_WORKBOOK_MASTER.IPYNB

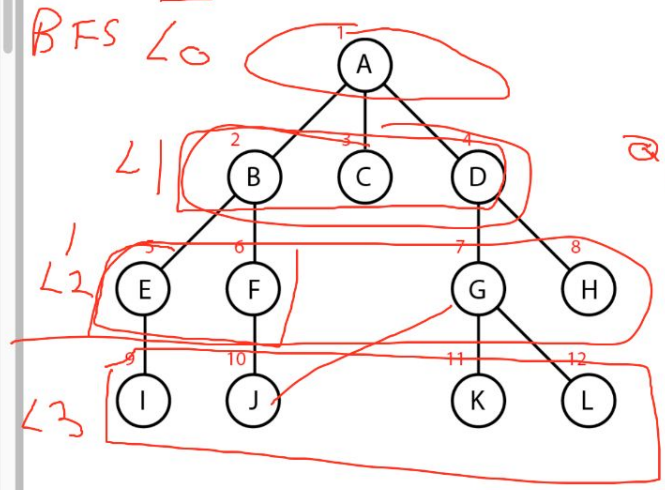
1. MASTER Unit 9 - Graph Algorithms at Scale
    1.0.1. Notebook Set-Up
2. Exercise 1. Graphs Overview
3. Warm-up questions:
4. Exercise 2. Data Structures Review.
5. Exercise 3. Graph Traversal.
    5.1. Depth-First Search
    5.2. Breadth-First Search
    5.3. Other great algorithm books:
        5.3.1. BFS traversal
        5.3.2. DFS
    5.4. BFS versus DFS (no animation)
    5.5. Example random graph
6. TASK: Write BFS Search Function In Apache Spark And Pandas Dataframe:
    6.1. TODO: iterative search over undirected graph
7. Exercise 4. SSSP for unweighted graphs.
    7.1. SSSP with a more illustrative example
    7.2. SSSP for a random graph
8. Exercise 5: Dijkstra's algorithm (SSSP for weighted graphs)
        8.0.0.1. What about longest pahs?
    8.1. 8.1.

Launcher | demo9_workbook_MASTER

Markdown | PySpark



5.4. BFS versus DFS (no animation)



[A]
[B, C, D]
[C, D, E, F]
[D, E, F]
[E, F, G, H]
[F, G, H, I]
[G, H, I, J]
[H, I, J, K, L]
[I, J, K, L]
[J, K, L]
[K, L]
[L]
[]

STATE of node
- Visited = [A, B, C, ...]
- Queue [... -]
- Unvisited [...]

Breadth First - FIFO

Start with node A and put it on the Queue.
Pop the "first node", mark it as "visited", and put all unvisited reachable nodes on the queue:

state of Q

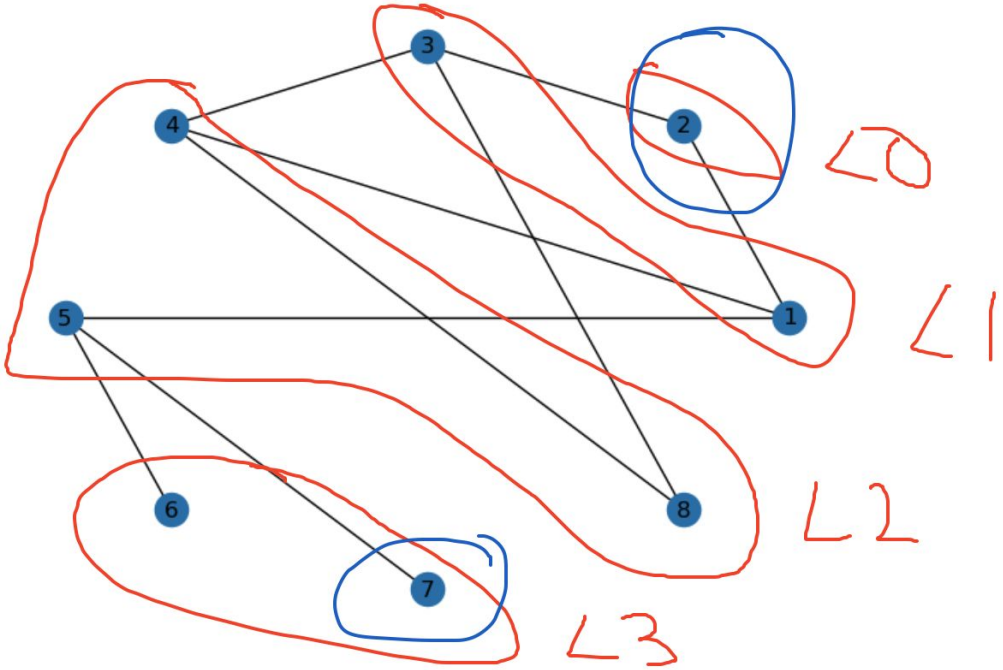
Q node	neighbors	State	Dist from SRC
A	B C D	Q	0
B	E F	V	1
C		V	
D		V	
		V	

Single shortest path calculation
Distance and levels: when we put each neighbor in the Queue we add (neighbor, distance to source node) structure to the Queue instead

Simple | 0 | 1 | PySpark Idle | Saving completed | Mode: Command | Ln 1, Col 1 | demo9_workbook_MASTER.ipynb

```
41 print(f'bfs_shortest_path(ADJ_GRAPH, 2, 7) is {bfs_shortest_path(ADJ_GRAPH_2, 2, 7)}')
```

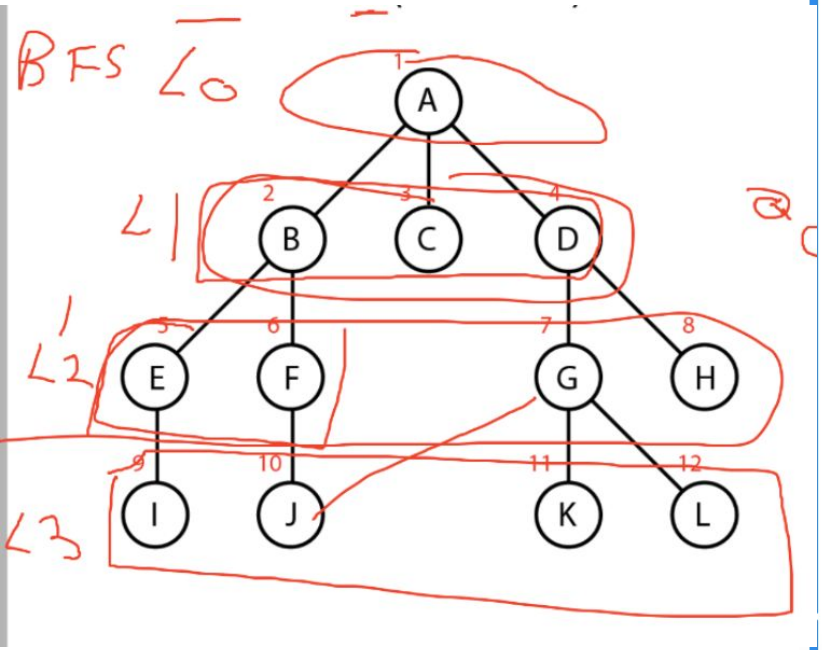
```
[1, 2, 3, 4, 5, 6, 7, 8]
[(1, 2), (1, 4), (1, 5), (2, 3), (3, 4), (3, 8), (4, 8), (5, 6), (5, 7)]
[2, 4, 5]
3
{1: [2, 4, 5], 2: [1, 3], 3: [2, 4, 8], 4: [3, 1, 8], 5: [1, 6, 7], 6: [5], 7: [5], 8: [4, 3]}
BFS Shortest paths
_____________________________________________
visited: [];  queue: deque([[2]])
visited: [2];  queue: deque([[2, 1], [2, 3]])
visited: [2, 1];  queue: deque([[2, 3], [2, 1, 2], [2, 1, 4], [2, 1, 5]])
visited: [2, 1, 3];  queue: deque([[2, 1, 2], [2, 1, 4], [2, 1, 5], [2, 3, 2], [2, 3, 4], [2, 3, 8]])
visited: [2, 1, 3];  queue: deque([[2, 1, 4], [2, 1, 5], [2, 3, 2], [2, 3, 4], [2, 3, 8]])
visited: [2, 1, 3, 4];  queue: deque([[2, 1, 5], [2, 3, 2], [2, 3, 4], [2, 3, 8], [2, 1, 4, 3], [2, 1, 4, 1], [2, 1, 4, 8]])
bfs_shortest_path(ADJ_GRAPH, 2, 7) is ([2, 1, 5, 7], 3)
```



SSSP of an unweighted graph

BFS traversal with distances (via BFS tree)

- Each node is in a single state:
  - Queue
  - Visited (to avoid loops)
  - Unvisited (implicite)



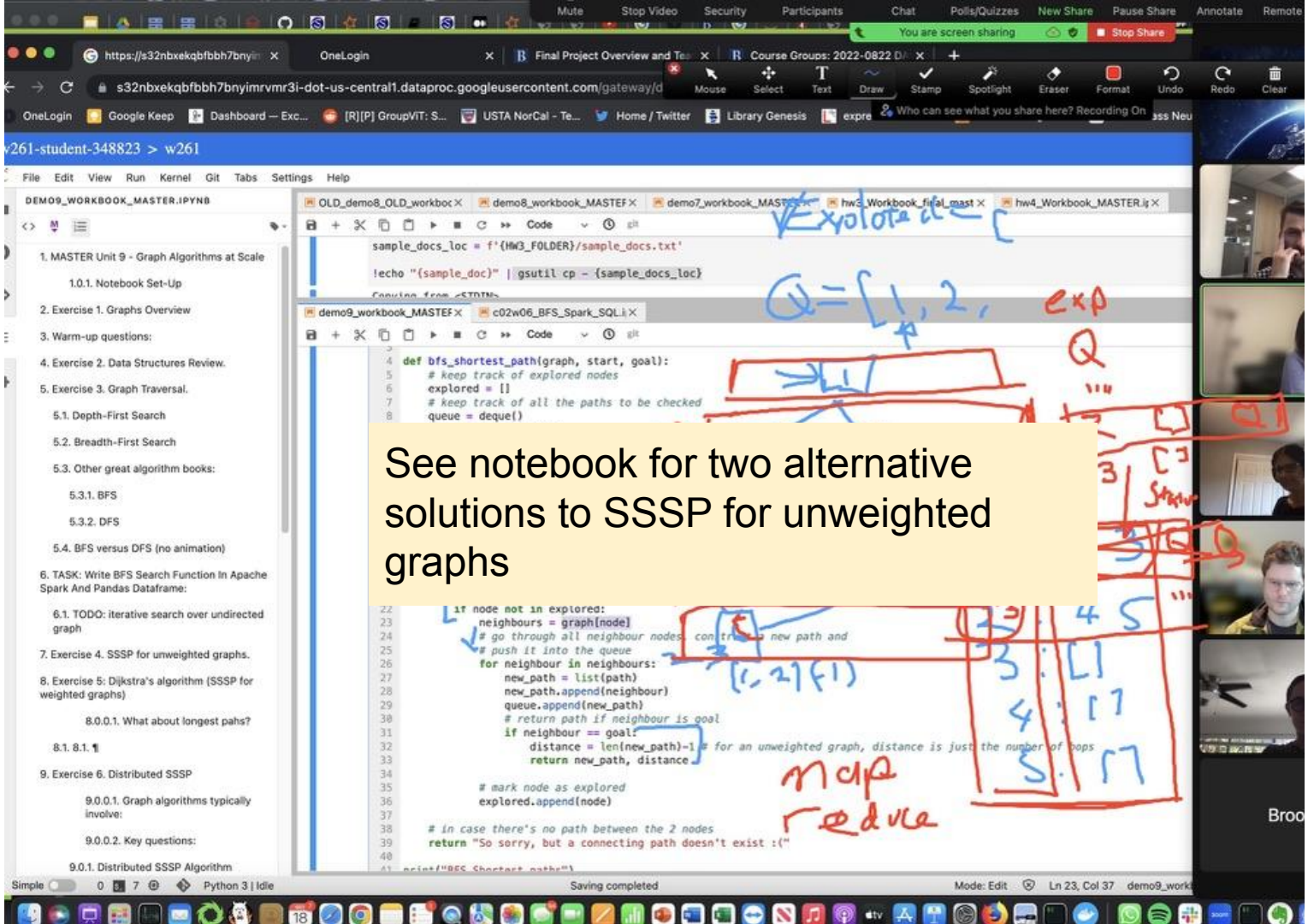
demo9_workbook_MASTER

Code

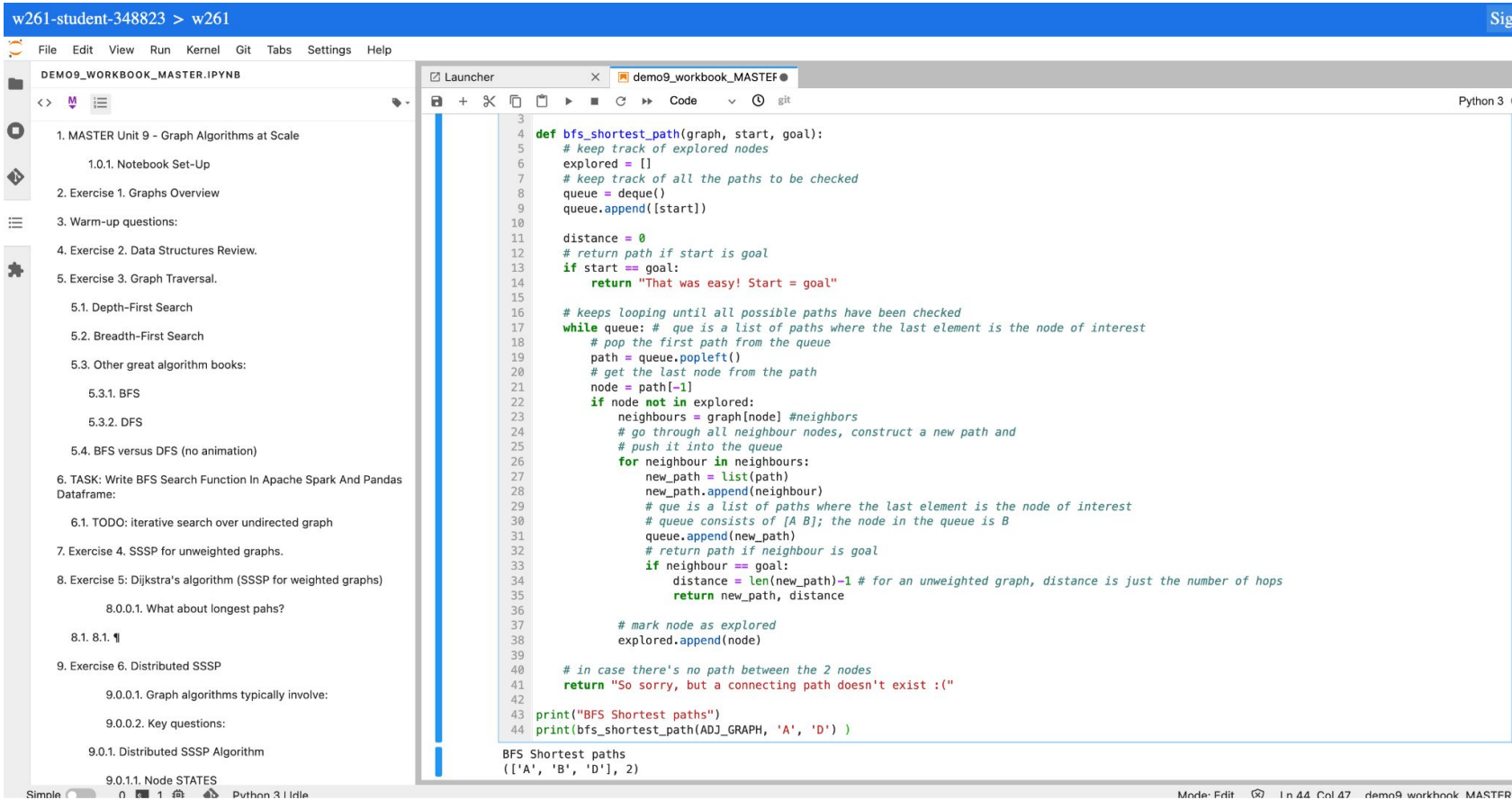
[8]:

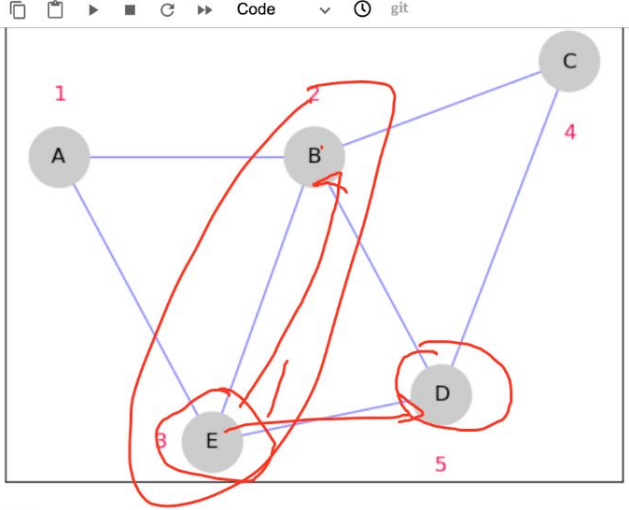
```python
from collections import deque
# Adapted from: [here](https://www.geeksforgeeks.org/shortest-path-unweighted-graph/)
def bfs_shortest_path(graph, start):
    # Create a queue and add the starting vertex to it
    queue = deque([start])

    # Create an array to keep track of the distances from the starting vertex to all other vertices
    distances = [float('inf')] * len(graph)
    distances[start] = 0

    # Create a set to keep track of visited vertices
    visited = set()

    # Perform BFS
    while queue:
        # Dequeue the next vertex
        vertex = queue.popleft()
        visited.add(vertex)

        # Update the distances of neighbors
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                distances[neighbor] = distances[vertex] + 1
                queue.append(neighbor)

    return distances


# Example graph: unweighted, directed graph with 5 vertices
# Vertices are represented by integers 0 through 4
# Edges: (0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4)
graph = [[1, 2], [2, 3], [3], [4], []]

start_vertex = 0
distances = bfs_shortest_path(graph, start_vertex)
print(distances) # Output: [0, 1, 1, 2, 3]

# Graph taken from above
#Graph as adjecency list
#{'A': ['B', 'E'], 'B': ['A', 'C', 'D', 'E'], 'C': ['B', 'D'], 'D': ['B', 'C', 'E'], 'E': ['A', 'B', 'D']}
# Graph nodes remapped to integers
# A  0
# B  1
# C  2
# D  3
# E  4

graph = [[1, 4],  [0, 2,3,4], [2, 3], [1, 2, 4], [0, 1, 3]]

start_vertex = 0
distances = bfs_shortest_path(graph, start_vertex)
print(distances) #
print(f"Distance from node (0) to node (4) is {distances[4]}")
```

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 2]
Distance from node (0) to node (4) is 2
```


See notebook for two alternative solutions to SSSP for unweighted graphs

Use BFS to calculated shortest path for unweighted graphs





Q · [A] [A,B] [A,E] [A B C] [A B D] A E D

V A B E

U

[20]:
```
ADJ_GRAPH
#Graph as adjecency list
#{'A': ['B', 'E'], 'B': ['A', 'C', 'D', 'E'], 'C': ['B', 'D'], 'D': ['B', 'C', 'E'], 'E': ['A', 'B', 'D']}
ADJ_GRAPH = {'A': ['B', 'E'],
 'B': ['A', 'C', 'D', 'E'],
 'C': ['B', 'D'],
 'D': ['B', 'C', 'E'],
 'E': ['A', 'B', 'D']}
ADJ_GRAPH
```

[20]:
```
{'A': ['B', 'E'],
 'B': ['A', 'C', 'D', 'E'],
 'C': ['B', 'D'],
 'D': ['B', 'C', 'E'],
 'E': ['A', 'B', 'D']}
```

[ ]:
```

```

[27]:
```
# finds shortest path between 2 nodes of a graph using BFS
# Does not consider edge weights

def bfs_shortest_path(graph, start, goal):
    """    """

    # keep tracvisit of explored nodes
    explored = []
```

Settings Help

OLD_demo8_OLD_workboc | demo8_workbook_MASTEF | demo7_workbook_MASTER | hw3_Workbook_final_mast | hw4_Workbook_MASTER.ip

Code git

```
/ [1 files][    0.0 B/    0.0 B]
Operation completed over 1 objects.
```

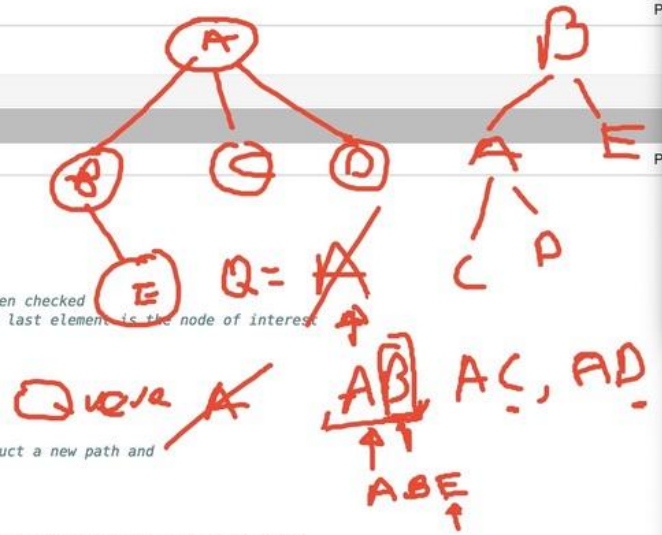
[22]:

demo9_workbook_MASTEF | c02w06_BFS_Spark_SQL.ip

Code git

```
10
11      distance = 0
12      # return path if start is goal
13      if start == goal:
14          return "That was easy! Start = goal"
15
16      # keeps looping until all possible paths have been checked
17      while queue: #  que is a list of paths where the last element is the node of interest
18          # pop the first path from the queue
19          path = queue.popleft()
20          # get the last node from the path
21          node = path[-1]
22          if node not in explored:
23              neighbours = graph[node] #neighbors
24              # go through all neighbour nodes, construct a new path and
25              # push it into the queue
26              for neighbour in neighbours:
27                  new_path = list(path)
28                  new_path.append(neighbour)
29                  # que is a list of paths where the last element is the node of interest
30                  # queue consists of [A B]; the node in the queue is B
31                  queue.append(new_path)
32                  # return path if neighbour is goal
33                  if neighbour == goal:
34                      distance = len(new_path)-1 # for an unweighted graph, distance is just the number of hops
35                      return new_path, distance
36
37              # mark node as explored
38              explored.append(node)
39
40      # in case there's no path between the 2 nodes
41      return "So sorry, but a connecting path doesn't exist :("
42
43  print("BFS Shortest paths")
44  print(bfs_shortest_path(ADJ_GRAPH, 'A', 'D') )
45
```

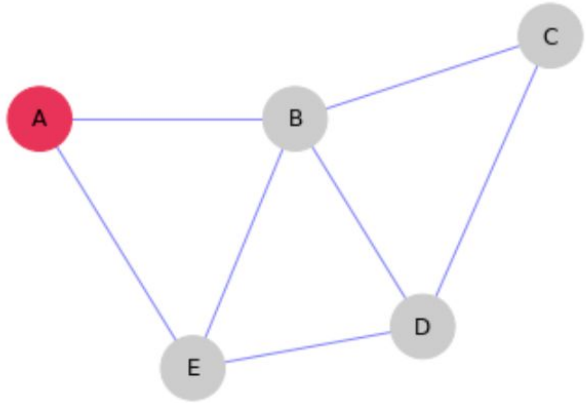
```
BFS Shortest paths
```



Idle | Saving completed | Mode: Edit | Ln 35, Col 46 | demo9_workbook_MAST

Possible solution 1 to BFS/SSSP/Dikstra with a single Spark Dataframe

Implement BFS at scale to calculated shortest path for unweighted graphs




```
queue_len = 0

graph_df = load_graph(location)

status_df =

While queue_len > 0:

    graph…..(status_df).flatmap(.....)

    reduce
```

graph_df

Node	Out_nodes	distance	state
A	B,E	0	Q
B	A,C,D,E	inf	U
C	B,D	inf	U
D	B,C,E	inf	U
E	A,B,D	inf	U

Iterate through rows in a dataframe: map(), mapPartitions()

PySpark provides map(), mapPartitions() to loop/iterate through rows in RDD/DataFrame to perform the complex transformations, and these two returns the same number of records as in the original DataFrame but the number of columns could be different (after add/update).

PySpark also provides foreach() & foreachPartitions() actions to loop/iterate through each Row in a DataFrame but these two returns nothing.

This article explains how to use these methods to get DataFrame column values and process them:

https://sparkbyexamples.com/pyspark/pyspark-loop-iterate-through-rows-in-dataframe/

https://sparkbyexamples.com/pyspark/pyspark-loop-iterate-through-rows-in-dataframe/

DataFrames: mini review: create a dataframe

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

data = [('James','Smith','M',30),('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]
columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()
+---------+--------+------+------+
|firstname|lastname|gender|salary|
+---------+--------+------+------+
|    James|   Smith|     M|    30|
|     Anna|    Rose|     F|    41|
|   Robert|Williams|     M|    62|
+---------+--------+------+------+
```

```
from pyspark.sql.functions import concat_ws,col,lit
df.select(concat_ws(",",df.firstname,df.lastname).alias("name"), \
          Df.gender,  lit(df.salary*2).alias("new_salary")).show()
+---------------+------+----------+
|           name|gender|new_salary|
+---------------+------+----------+
|    James,Smith|     M|        60|
|      Anna,Rose|     F|        82|
|Robert,Williams|     M|       124|
+---------------+------+----------+
```

Mostly for simple computations, instead of iterating through using map() and foreach(), you should use either DataFrame select() or DataFrame withColumn() in conjunction with PySpark SQL functions.

https://sparkbyexamples.com/pyspark/pyspark-loop-iterate-through-rows-in-dataframe/

map() to loop/iterate through the PySpark DataFrame/RDD

PySpark map() Transformation is used to loop/iterate through the PySpark DataFrame/RDD by applying the transformation function (lambda) on every element (Rows and Columns) of RDD/DataFrame. PySpark doesn't have a map() in DataFrame instead it's in RDD hence we need to convert DataFrame to RDD first and then use the map(). It returns an RDD and you should Convert RDD to PySpark DataFrame if needed.

If you have a heavy initialization use PySpark mapPartitions() transformation instead of map(), as with mapPartitions() heavy initialization executes only once for each partition instead of every record.

```
# Referring columns by index.
rdd=df.rdd.map(lambda x: (x[0]+","+x[1],x[2],x[3]*2))
df2=rdd.toDF(["name","gender","new_salary"])
df2.show()
# Referring Column Names
rdd2=df.rdd.map(lambda x: (x.firstname+","+x.lastname,x.gender,x.salary*2) )

# By Calling function
def func1(x):
    name=x.firstname+","+x.lastName
    return (name, x.gender.lower(),    x.salary*2)

rdd2=df.rdd.map(lambda x: func1(x))
```

```
data = [('James','Smith','M',30),
  ('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()
+---------+--------+------+------+
|firstname|lastname|gender|salary|
+---------+--------+------+------+
|    James|   Smith|     M|    30|
|     Anna|    Rose|     F|    41|
|   Robert|Williams|     M|    62|
+---------+--------+------+------+
```

X is a row in the dataframe

```
# Refering columns by index.
rdd2=df.rdd.map(lambda x:
    (x[0]+","+x[1],x[2],x[3]*2)
    )
df2=rdd2.toDF(["name","gender","new_salary"]   )
df2.show()
+--------------+------+----------+
|          name|gender|new_salary|
+--------------+------+----------+
|    James,Smith|     M|        60|
|      Anna,Rose|     F|        82|
|Robert,Williams|     M|       124|
+--------------+------+----------+
```

https://sparkbyexamples.com/pyspark/pyspark-map-transformation/?expand_article=1

```
data = [('James','Smith','M',30),
  ('Anna','Rose','F',41),
  ('Robert','Williams','M',62),
]

columns = ["firstname","lastname","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()
+---------+--------+------+------+
|firstname|lastname|gender|salary|
+---------+--------+------+------+
|    James|   Smith|     M|    30|
|     Anna|    Rose|     F|    41|
|   Robert|Williams|     M|    62|
+---------+--------+------+------+
```

```
# Refering columns by index.
rdd2=df.rdd.map(lambda x:
    (x[0]+","+x[1],x[2],x[3]*2)
    )
df2=rdd2.toDF(["name","gender","new_salary"]   )
df2.show()
+--------------+------+----------+
|          name|gender|new_salary|
+--------------+------+----------+
|    James,Smith|     M|        60|
|      Anna,Rose|     F|        82|
|Robert,Williams|     M|       124|
+--------------+------+----------+
```

Note that above we used the index to get the column values, alternatively, you can also refer to the DataFrame column names while iterating.

Copy

```
# Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]*2)
    )
```

Another alternative

```
# Referring Column Names
rdd2=df.rdd.map(lambda x:
    (x.firstname+","+x.lastname,x.gender,x.salary*2)
    )
```

You can also create a custom function to perform an operation. Below `func1()` function executes for every DataFrame row from the lambda function.

https://sparkbyexamples.com/pyspark/pyspark-map-transformation/?expand_article=1

df.foreach

```
# Foreach example
def f(x): print(x)
df.foreach(f)

# Another example
df.foreach(lambda x:
  print("Data==>"+x["firstname"]+","+x["lastname"]+","+x["gender"]+","+str(x["salary"]*2))
)
```

https://sparkbyexamples.com/pyspark/pyspark-loop-iterate-through-rows-in-dataframe/

Possible solution 1 to BFS/SSSP/Dikstra with a single Spark Dataframe

Implement BFS at scale to calculated shortest path for unweighted graphs

queue_len = 0

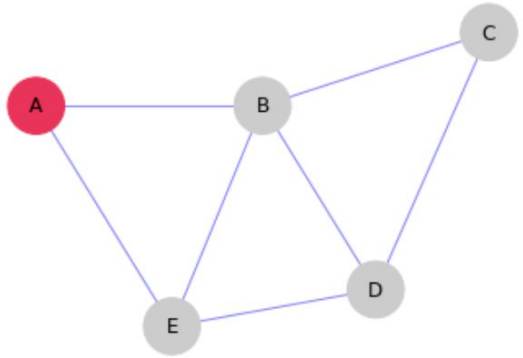
graph_df = load_graph(location)

status_df =

While queue_len > 0:

    graph…..(status_df).flatmap(.....)

    reduce



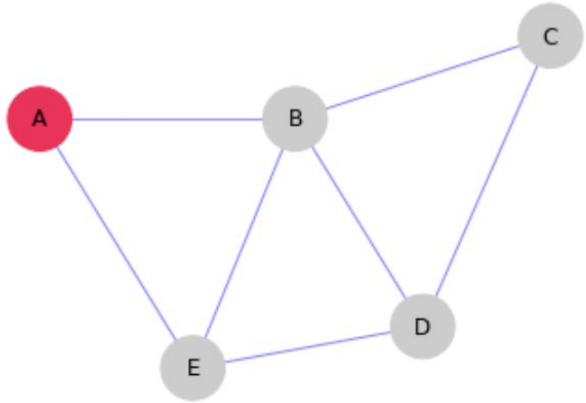
graph_df

Node	Out_nodes	distance	state
A	B,E	0	Q
B	A,C,D,E	inf	U
C	B,D	inf	U
D	B,C,E	inf	U
E	A,B,D	inf	U

Two

Possible solution 2 to BFS/SSSP/Dikstra with a single Spark DataframeS

Implement BFS at scale to calculated shortest path for unweighted graphs




```
queue_len = 0

graph_df = load_graph(location)

status_df =

While queue_len > 0:

    graph…..(status_df).flatmap(.....)

    reduce
```

JOIN

graph_df

Node	Out_nodes
A	B,E
B	A,C,D,E
C	B,D
D	B,C,E
E	A,B,D

status_df

Node	distance	state
A	0	Q
B	inf	U
C	inf	U
D	inf	U
E	inf	U

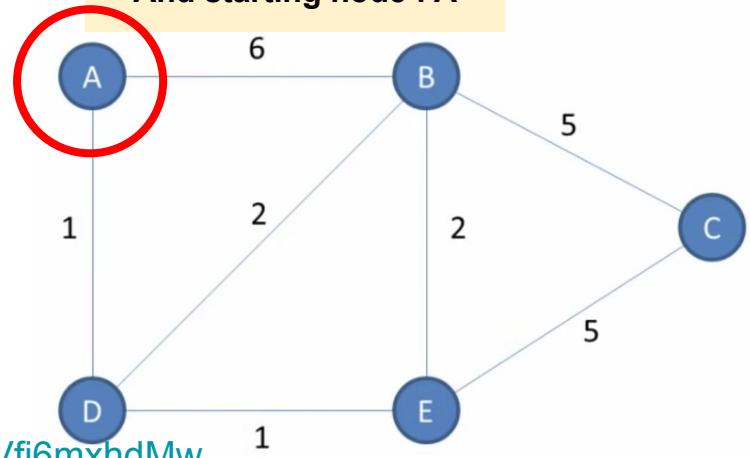
Dijkstra's shortest path algorithm for a weighted graph

Dijkstra’s shortest path algorithm for a weighted graph

- Dijkstra’s shortest path algorithm generates a set of information that includes the shortest paths from a starting vertex and every other vertex in the graph.
- Why Dijkstra’s shortest path algorithm is an example of a greedy algorithm.

Find the shortest path from vertex A to every other vertex

Input Graph
And starting node : A

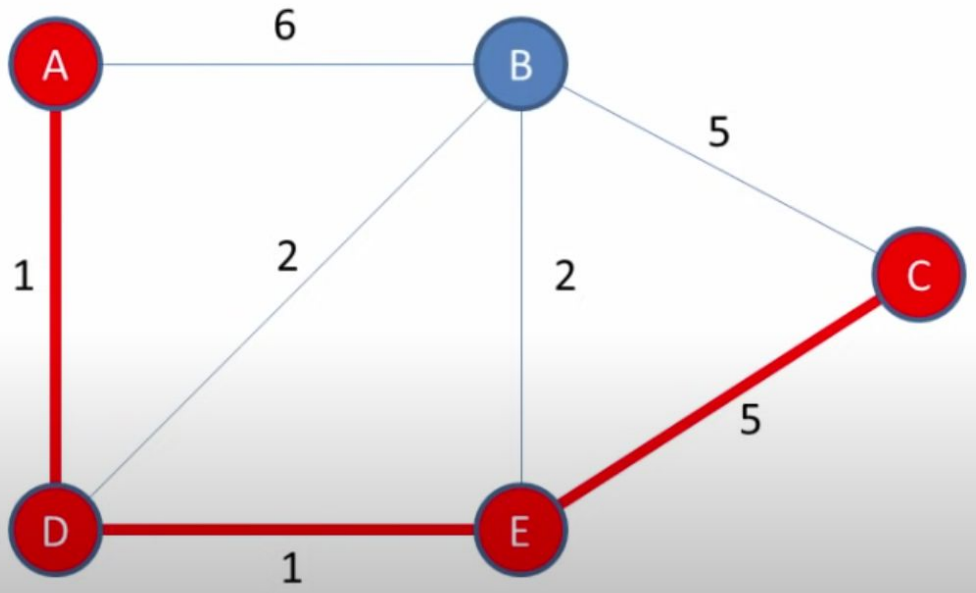


Output
Solution

Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

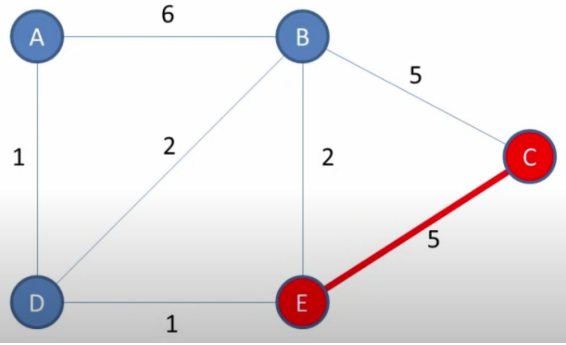
https://www.youtube.com/watch?v=pVfj6mxhdMw

A to C



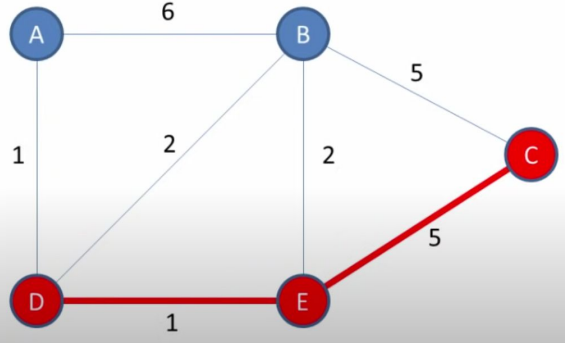
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at C via E Path = E → C



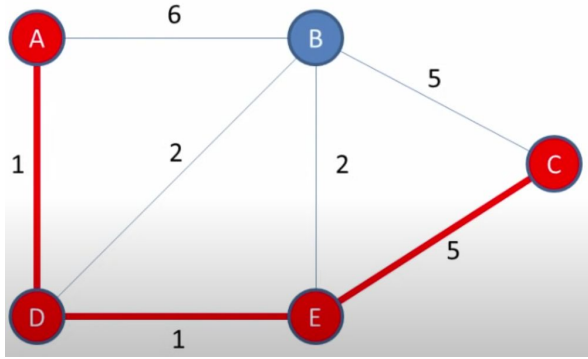
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at E via D Path = D → E → C



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

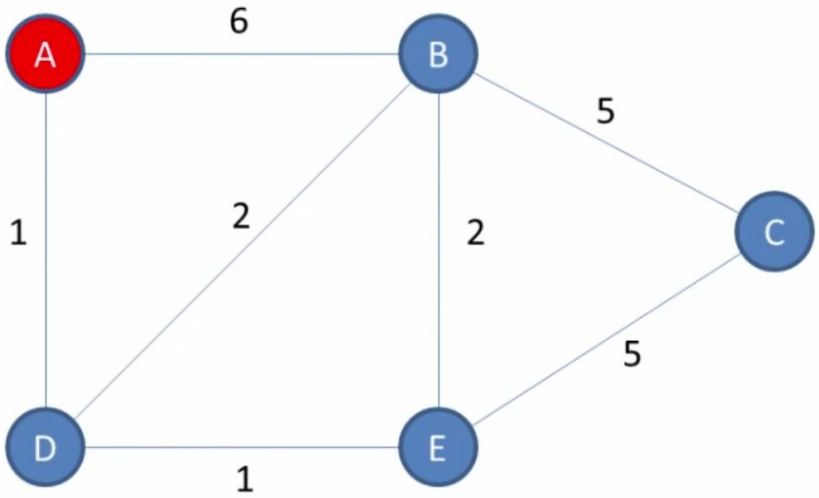
We arrived at D via A Path = A → D → E → C



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)

A node's State can be {Queue, Visited, Unvisited}



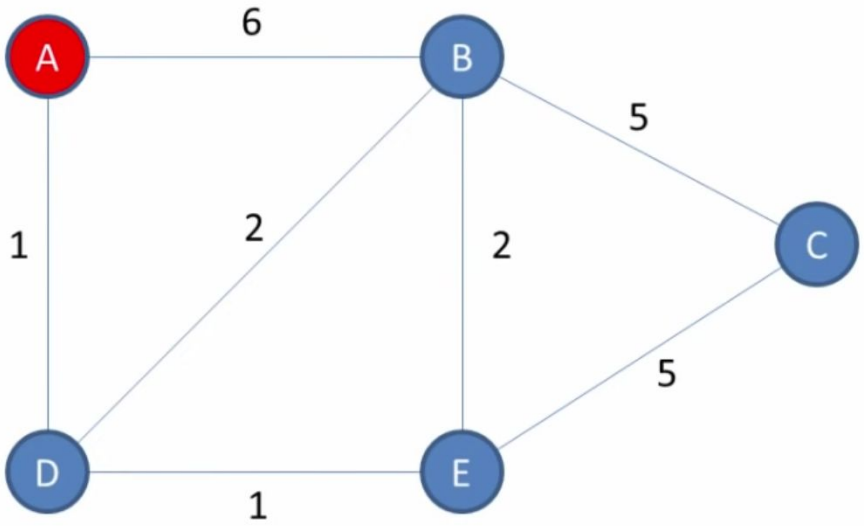
Vertex	Shortest distance from A	Previous vertex	State
A			Q
B			U
C			U
D			U
E			U

Visited = [ ]

Unvisited = [A, B, C, D, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

First time around, this is the start vertex itself, A

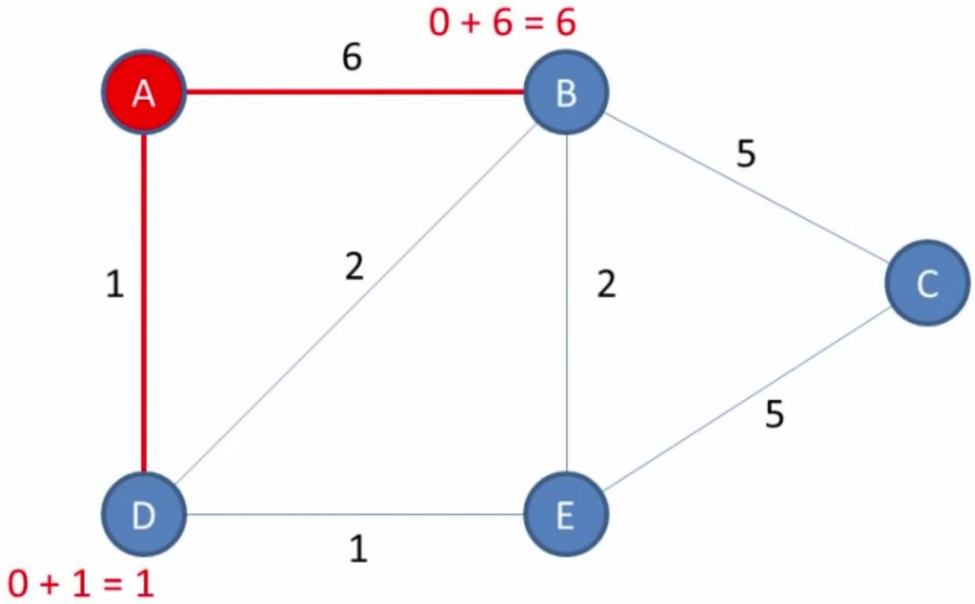


Vertex	Shortest distance from A	Previous vertex	State
A	0		Q
B	∞		U
C	∞		U
D	∞		U
E	∞		U

Visited = [ ]

Unvisited = [A, B, C, D, E]

For the current vertex, calculate the distance of each neighbour from the start vertex

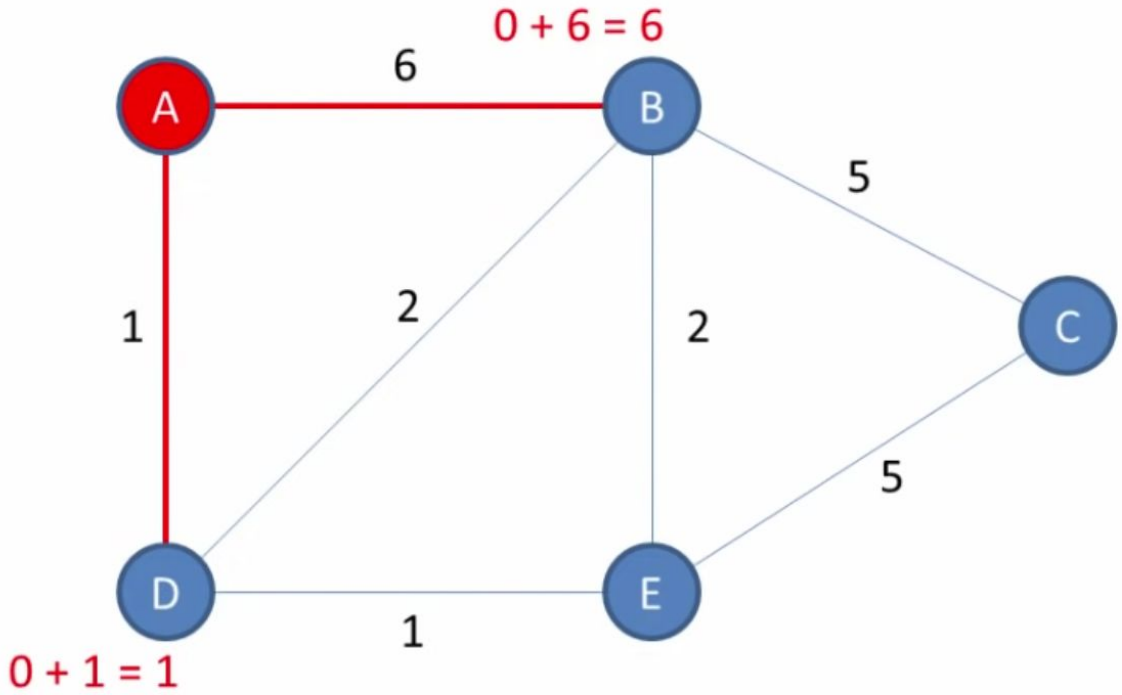


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance



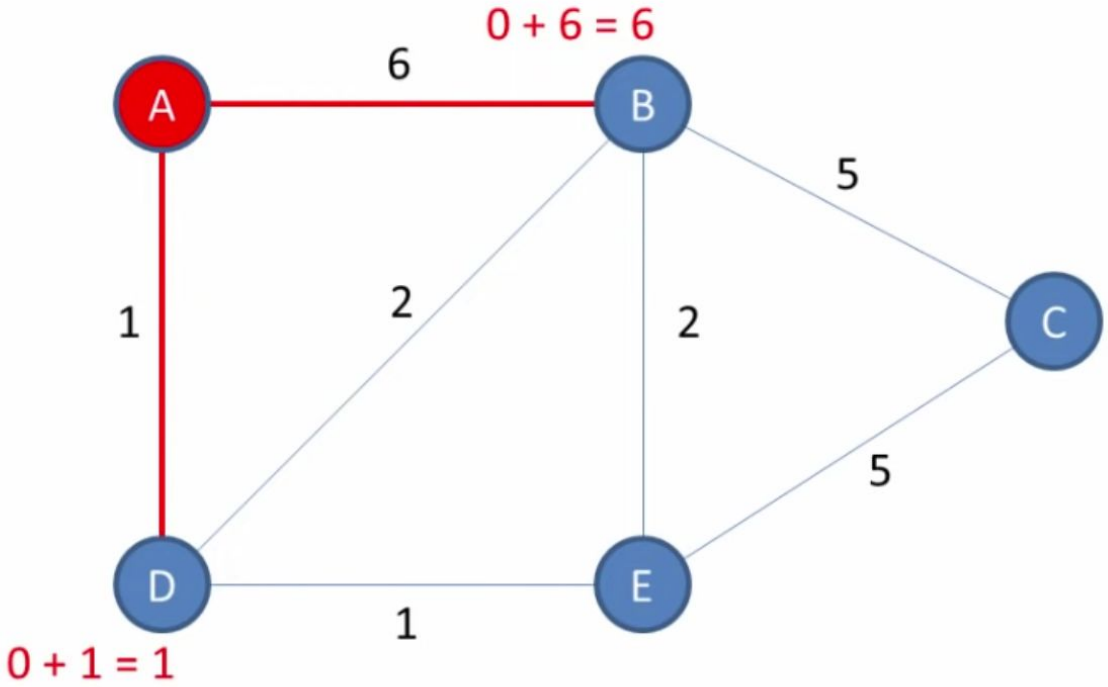
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	
C	∞	
D	1	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

Update the previous vertex for each of the updated distances

In this case we visited B and D via A

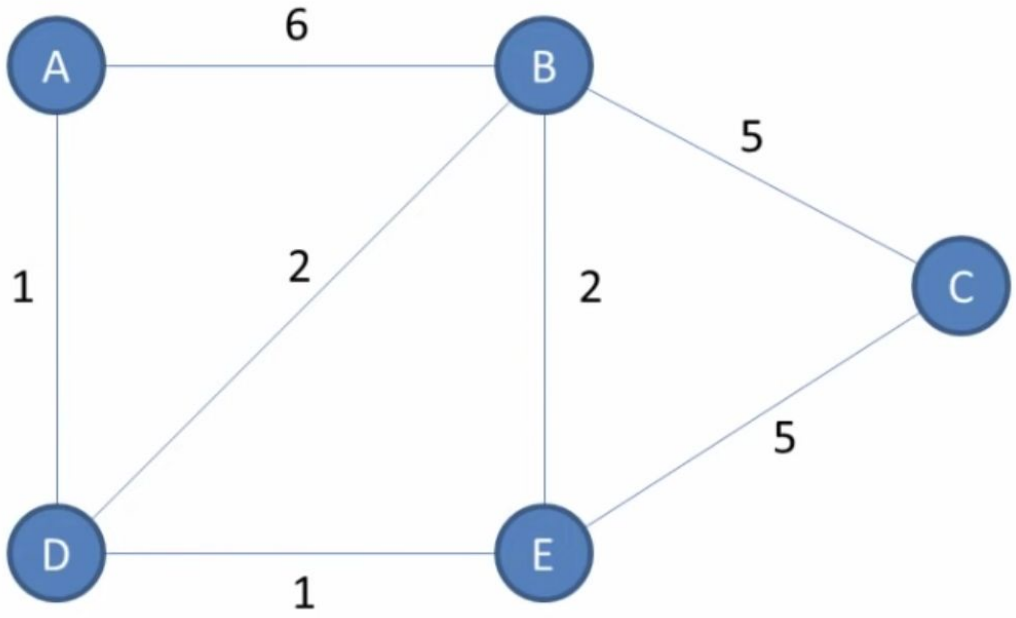


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	
C	∞	
D	1	
E	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E]

Add the current vertex to the list of visited vertices



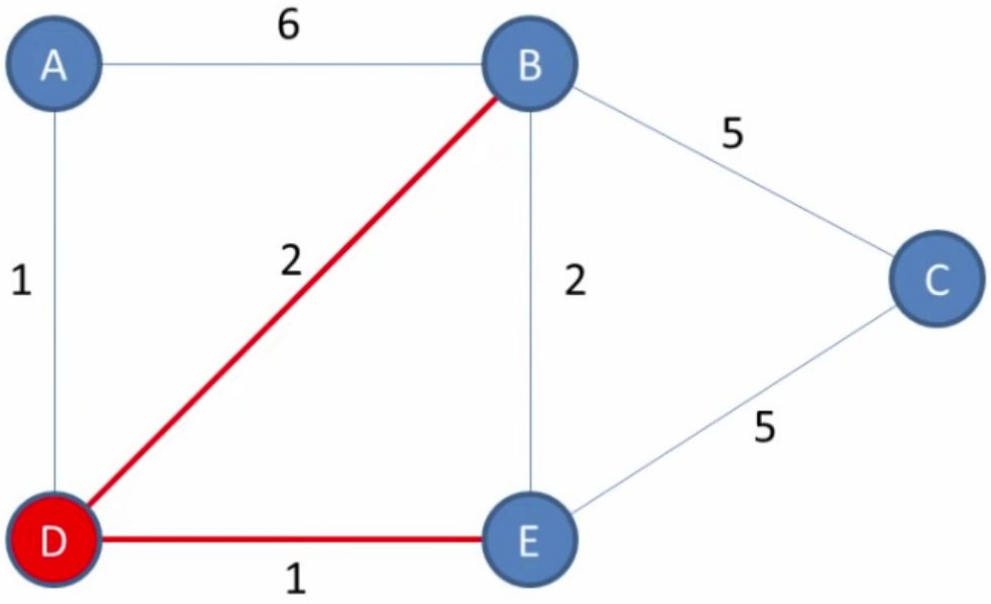
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting D and its unvisited neighbours are B and E

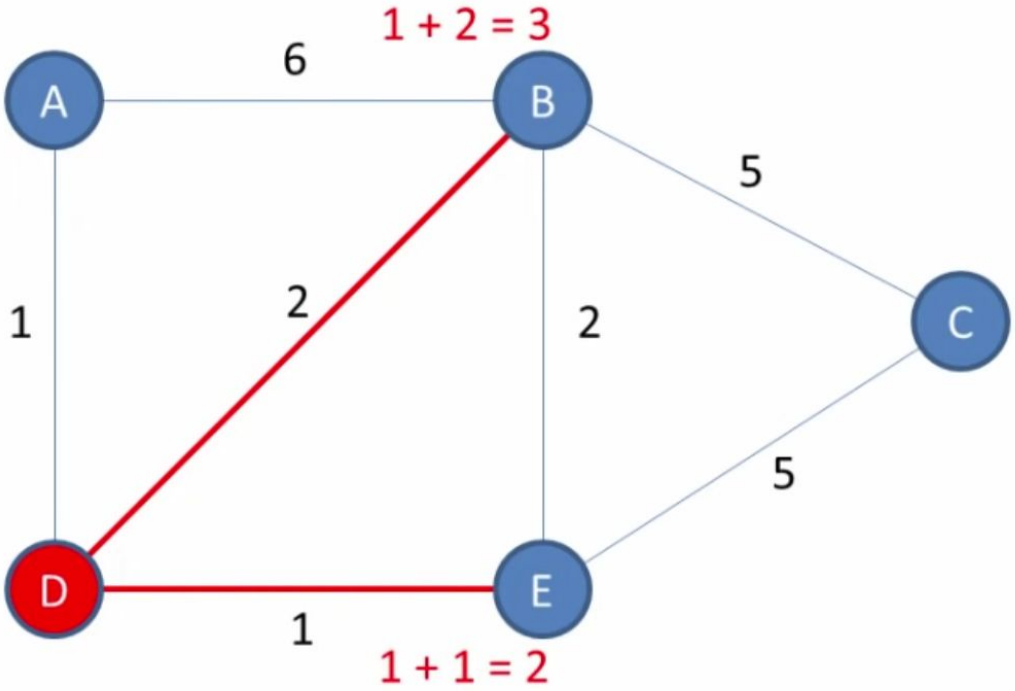


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

For the current vertex, calculate the distance of each neighbour from the start vertex

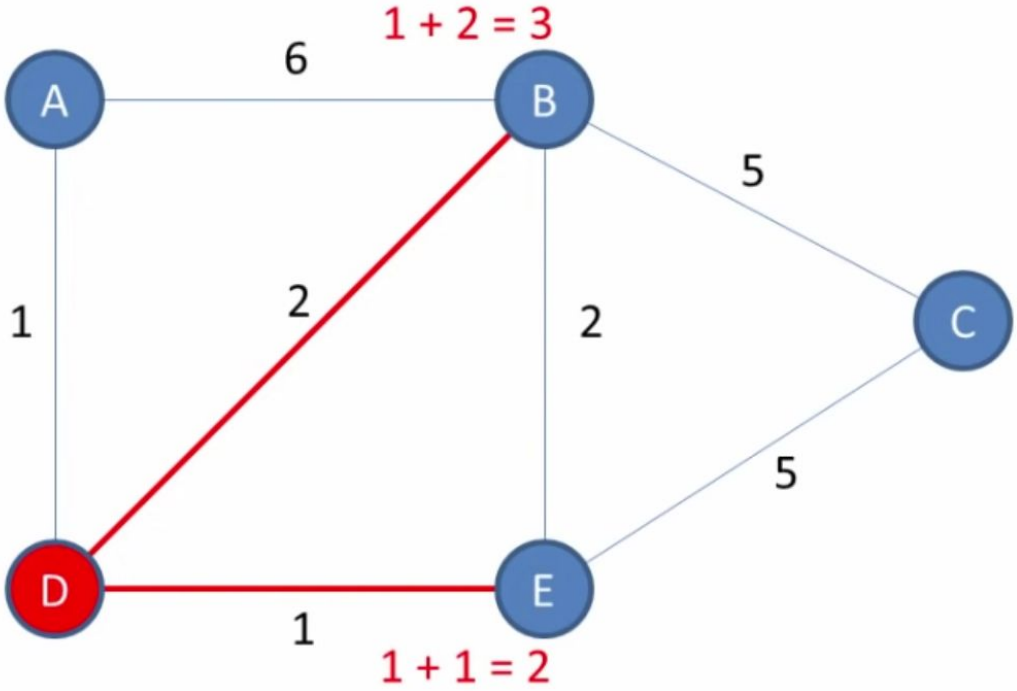


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

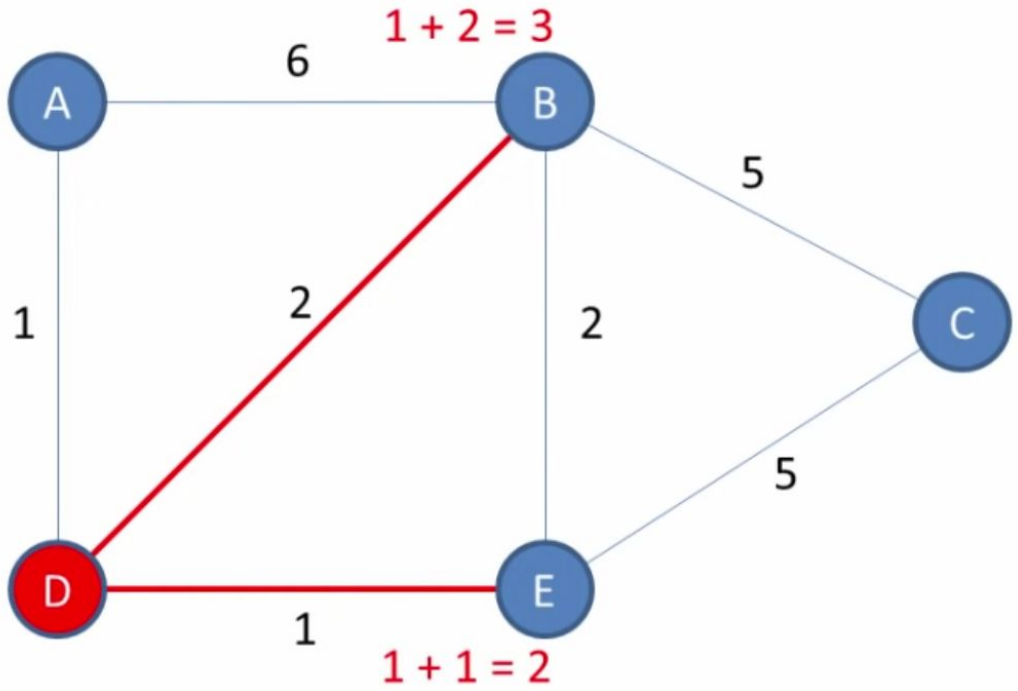


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

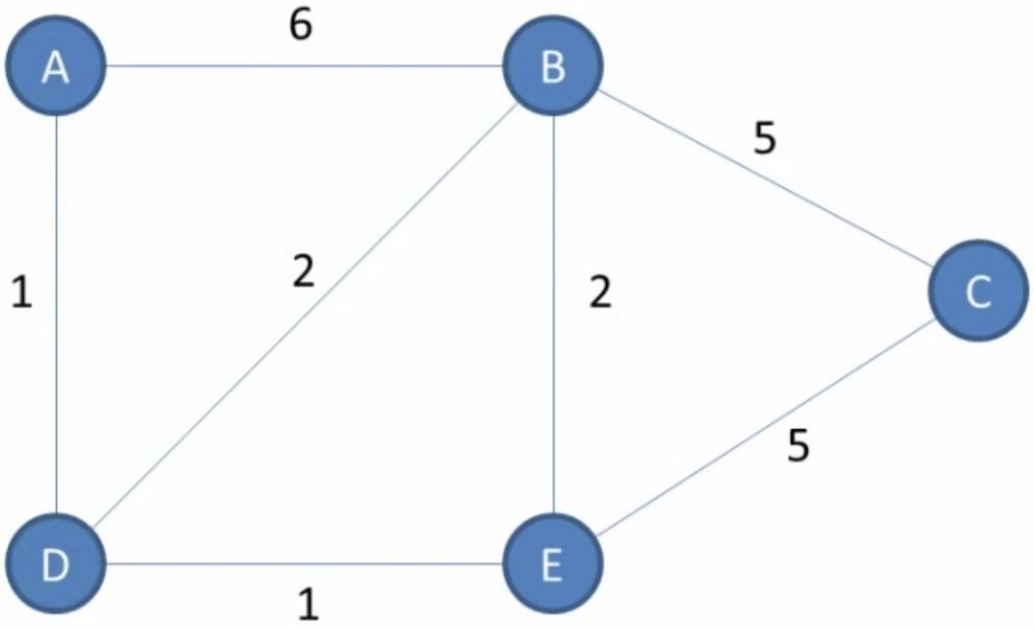


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	A
C	∞	
D	1	A
E	2	

Visited = [A]

Unvisited = [B, C, D, E]

Add the current vertex to the list of visited vertices



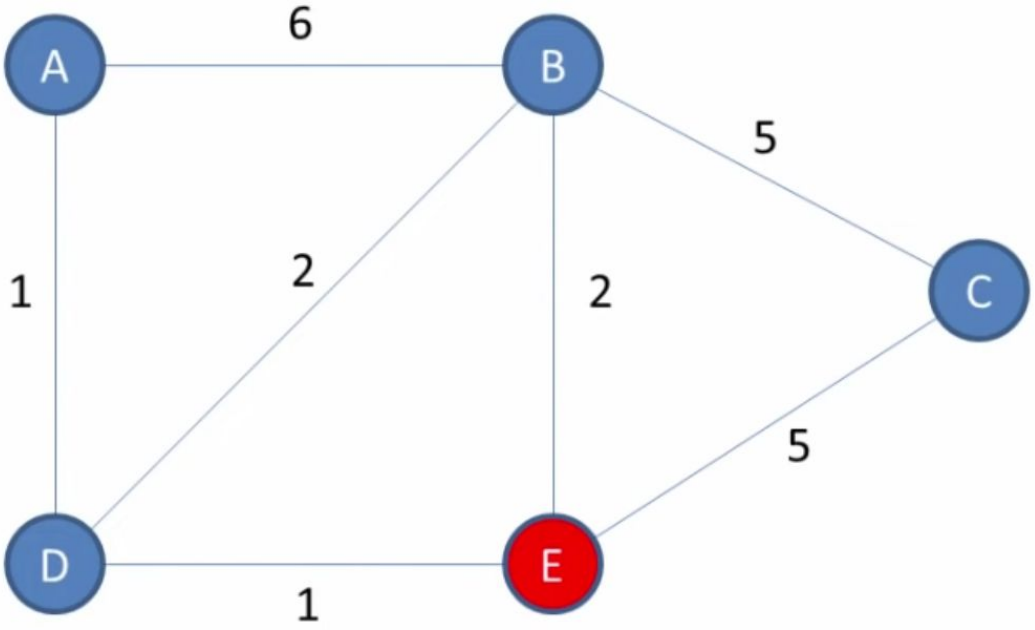
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex E



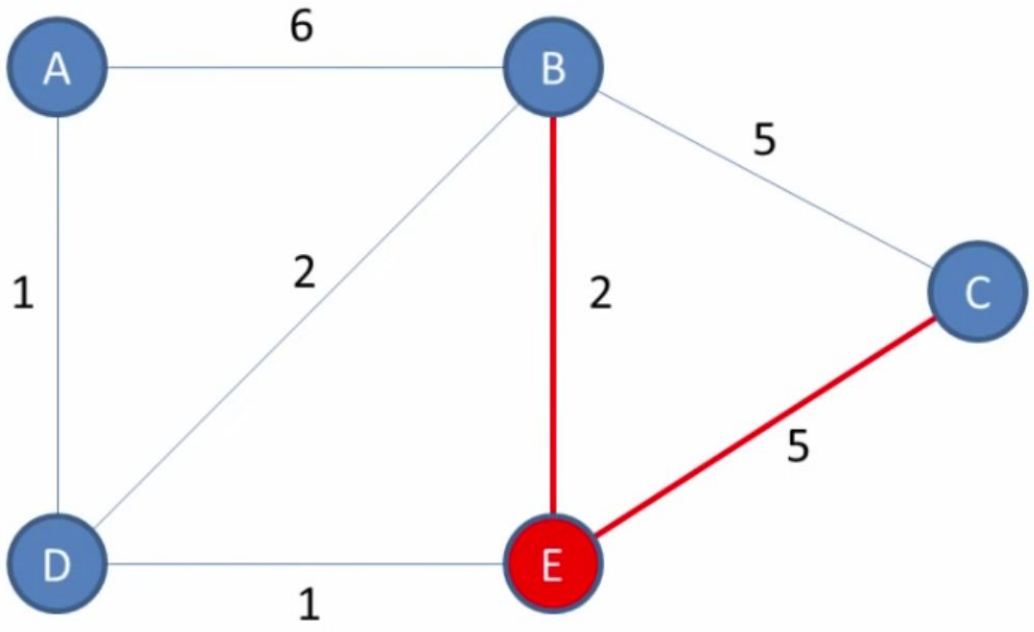
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting E and its unvisited neighbours are B and C

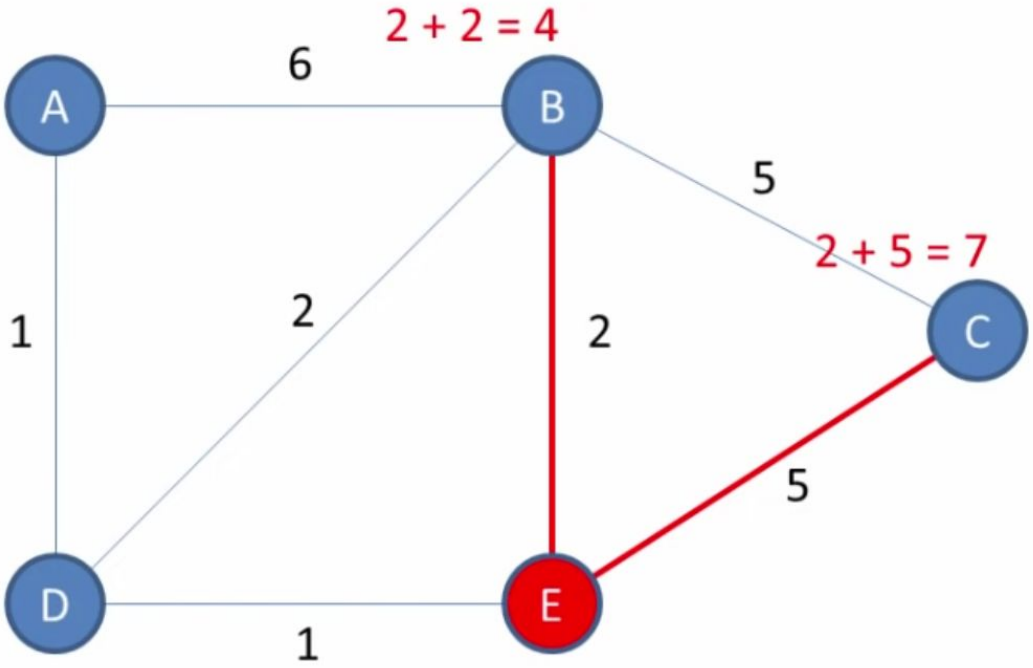


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

For the current vertex, calculate the distance of each neighbour from the start vertex



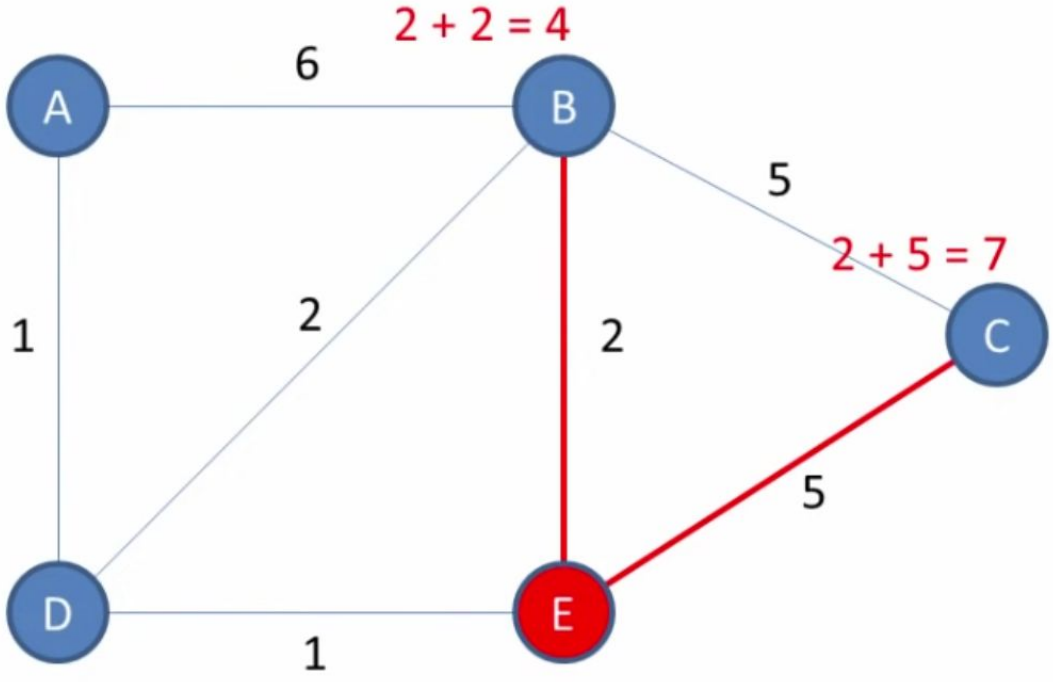
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to B



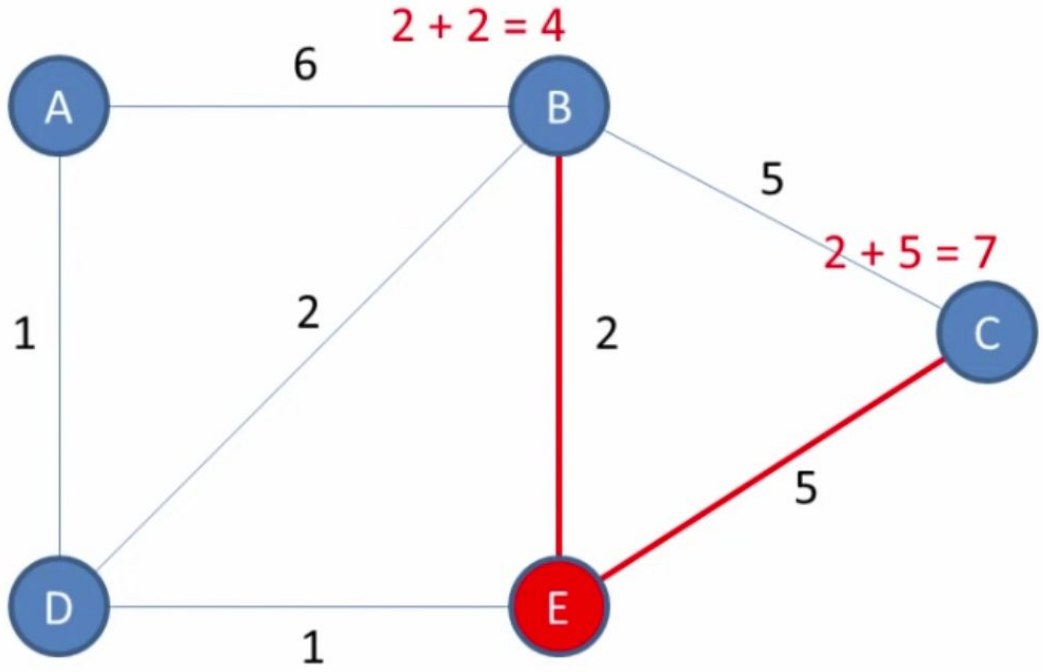
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Update the previous vertex for each of the updated distances

In this case we visited C via E

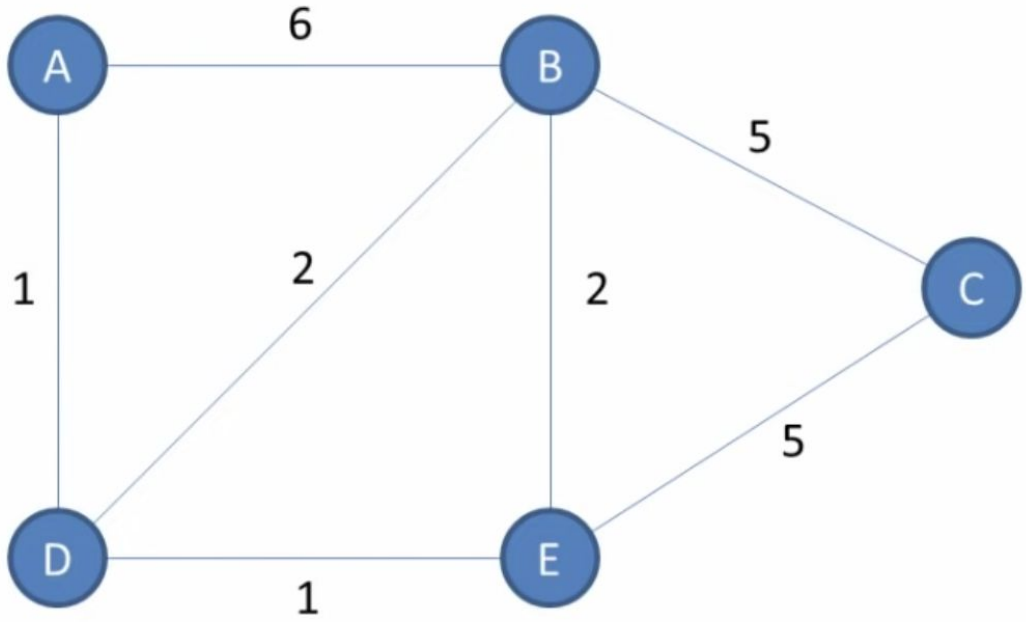


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Add the current vertex to the list of visited vertices



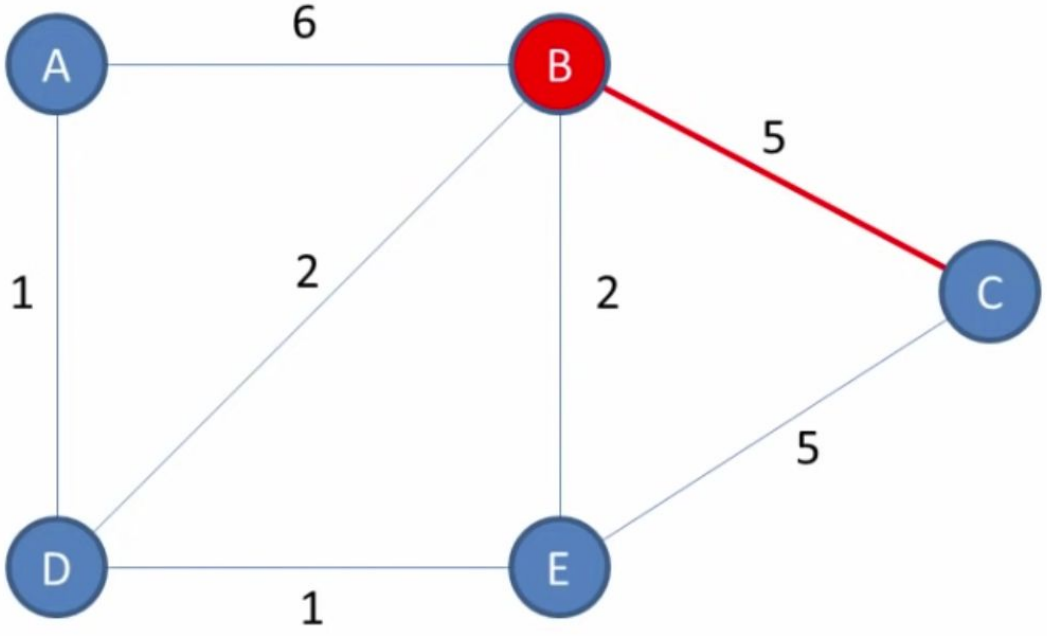
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

For the current vertex, examine its unvisited neighbours

We are currently visiting B and its only unvisited neighbour is C

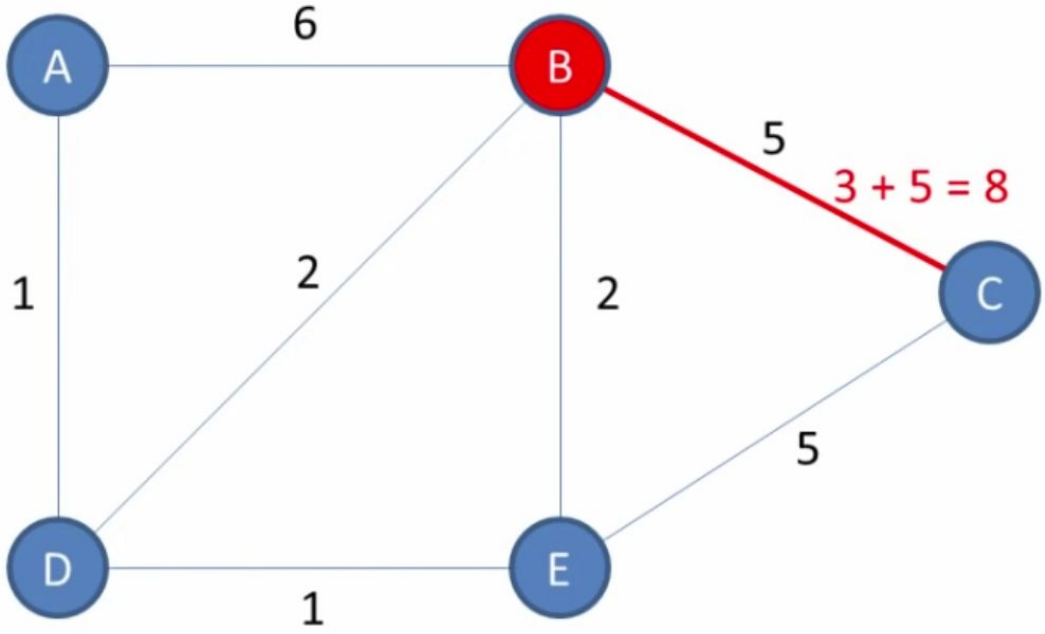


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

For the current vertex, calculate the distance of each neighbour from the start vertex



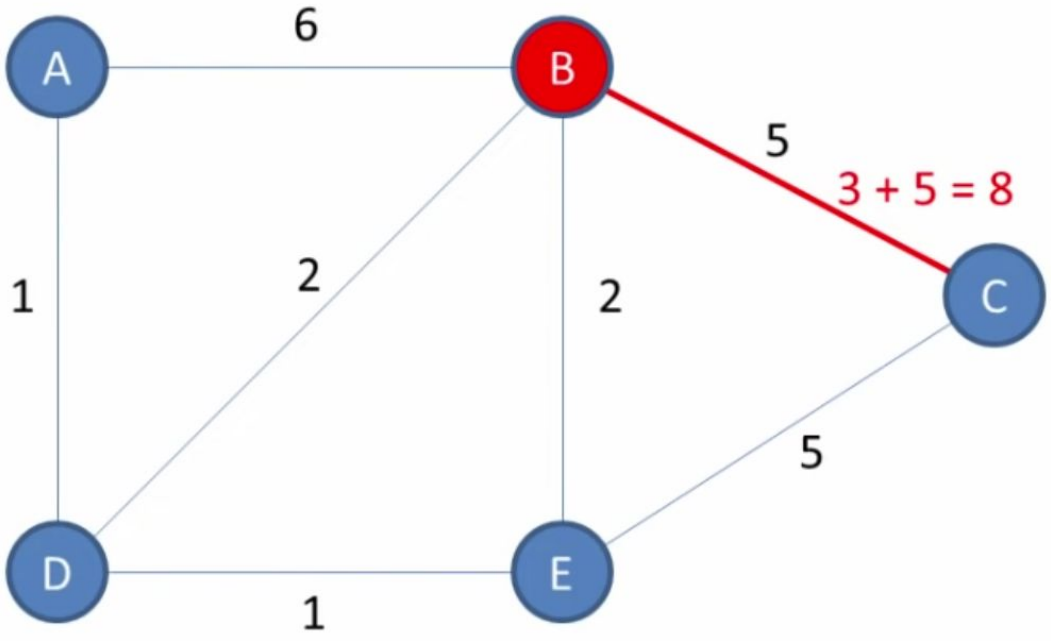
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

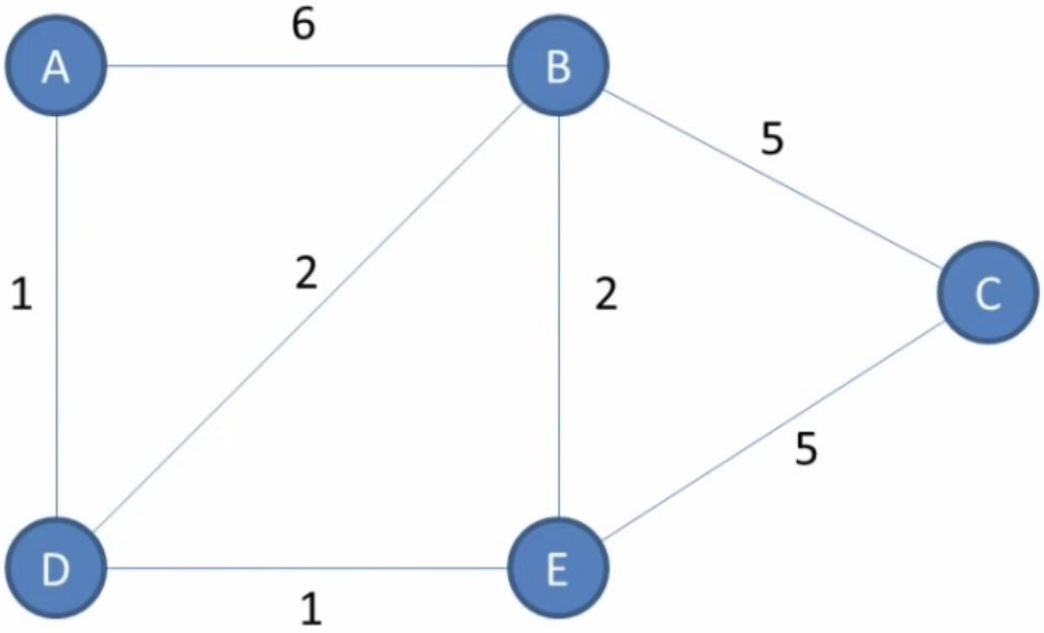
We do not need to update the distance to C



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

Add the current vertex to the list of visited vertices

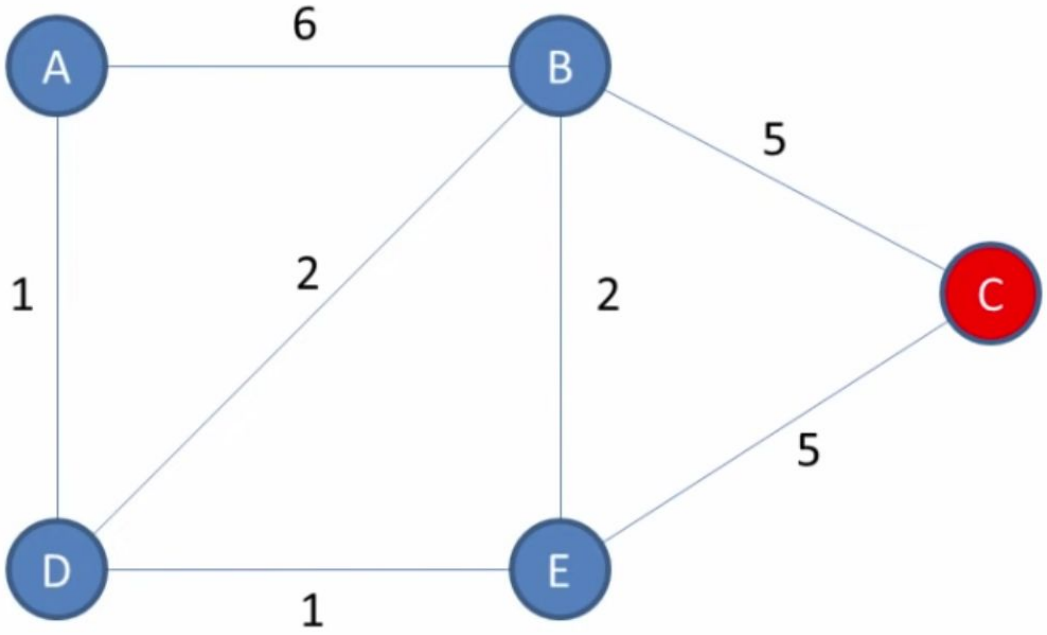


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

For the current vertex, examine its unvisited neighbours

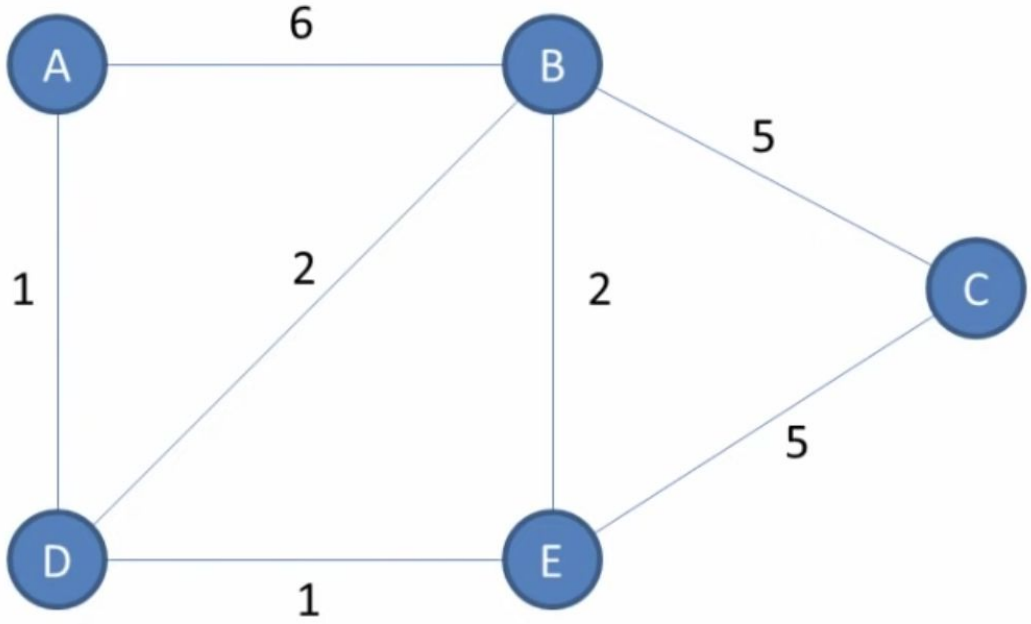
We are currently visiting C and it has no unvisited neighbours



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

Add the current vertex to the list of visited vertices



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B, C] Unvisited = []

Dikstras Algo: greedy algo (make locally optimal decisions)

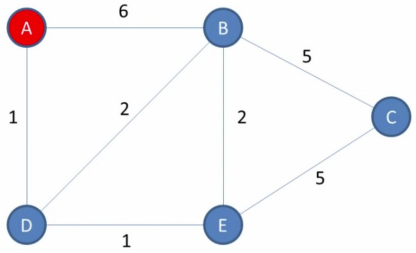
Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat
- Visit the unvisited vertex with the smallest known distance from the start vertex
- For the current vertex, examine its unvisited neighbours
- For the current vertex, calculate distance of each neighbour from start vertex
- If the calculated distance of a vertex is less than the known distance, update the shortest distance
- Update the previous vertex for each of the updated distances
- Add the current vertex to the list of visited vertices

Until all vertices visited

```python
# Single thread Dijskra's implementation
def dijkstra(graph, initial):
    """
    GOAL: Shortest distances from node A to all other nodes
    A node can be one of three states
           Visited
           Queue
           Unvisited
    """
    visited = {initial: 0}
    heap = [(0, initial)] # our priority queue
    path = {}

    nodes = set(graph.nodes) # unvisited state

    while nodes and heap: #priority queue

        current_weight, min_node = heapq.heappop(heap)
        try:
            while min_node not in nodes:
                current_weight, min_node = heapq.heappop(heap)
        except IndexError:
            break

        nodes.remove(min_node)

        for v in graph.edges[min_node]:
            weight = current_weight + graph.distances[min_node, v]
            if v not in visited or weight < visited[v]:
                visited[v] = weight
                heapq.heappush(heap, (weight, v))
                path[v] = min_node
    return visited
```

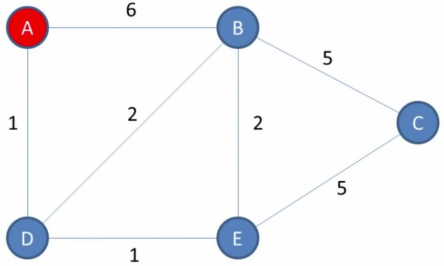
Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)



Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ] Unvisited = [A, B, C, D, E]

Visit the unvisited vertex with the smallest known distance from the start vertex
First time around, this is the start vertex itself, A

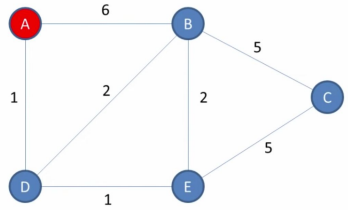


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

```python
# Single thread Dijskra's implementation
def dijkstra(graph, initial):
    """
    GOAL: Shortest distances from node A to all other nodes
    A node can be one of three states
           Visited
           Queue
           Unvisited
    """
    visited = {initial: 0}
    heap = [(0, initial)] # our priority queue
    path = {}

    nodes = set(graph.nodes) # unvisited state

    while nodes and heap: #priority queue

        current_weight, min_node = heapq.heappop(heap)
        try:
            while min_node not in nodes:
                current_weight, min_node = heapq.heappop(heap)
        except IndexError:
            break

        nodes.remove(min_node)

        for v in graph.edges[min_node]:
            weight = current_weight + graph.distances[min_node, v]
            if v not in visited or weight < visited[v]:
                visited[v] = weight
                heapq.heappush(heap, (weight, v))
                path[v] = min_node
    return visited
```

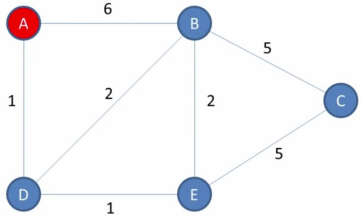
Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)



Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

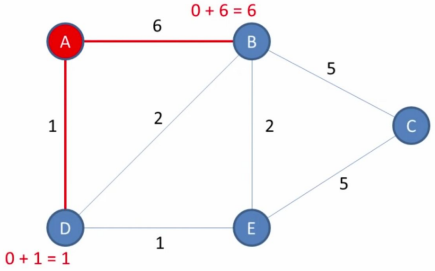
Visited = [ ] Unvisited = [A, B, C, D, E]

Visit the unvisited vertex with the smallest known distance from the start vertex
First time around, this is the start vertex itself, A



Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

For the current vertex, calculate the distance of each neighbour from the start vertex



Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ] Unvisited = [A, B, C, D, E]

Algorithm

Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat
- Visit the unvisited vertex with the smallest known distance from the start vertex
- For the current vertex, examine its unvisited neighbours
- For the current vertex, calculate distance of each neighbour from start vertex
- If the calculated distance of a vertex is less than the known distance, update the shortest distance
- Update the previous vertex for each of the updated distances
- Add the current vertex to the list of visited vertices

Until all vertices visited

Algorithm

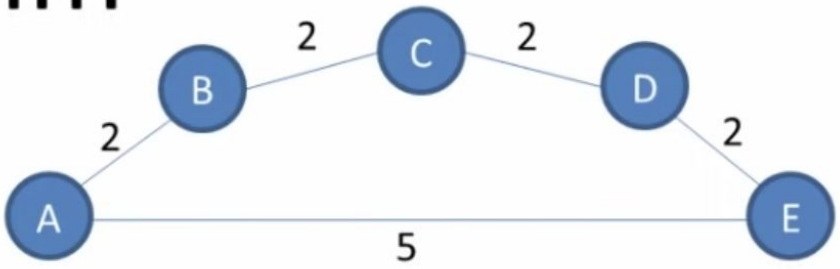
Greedy Algo: get to the end more quickly; makes locally optimal choices

Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat
- Visit the unvisited vertex with the smallest known distance from the start vertex
- For the current vertex, examine its unvisited neighbours
- For the current vertex, calculate distance of each neighbour from start vertex
- If the calculated distance of a vertex is less than the known distance, update the shortest distance
- Update the previous vertex for each of the updated distances
- Add the current vertex to the list of visited vertices

Until all vertices visited

Algorithm



Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat

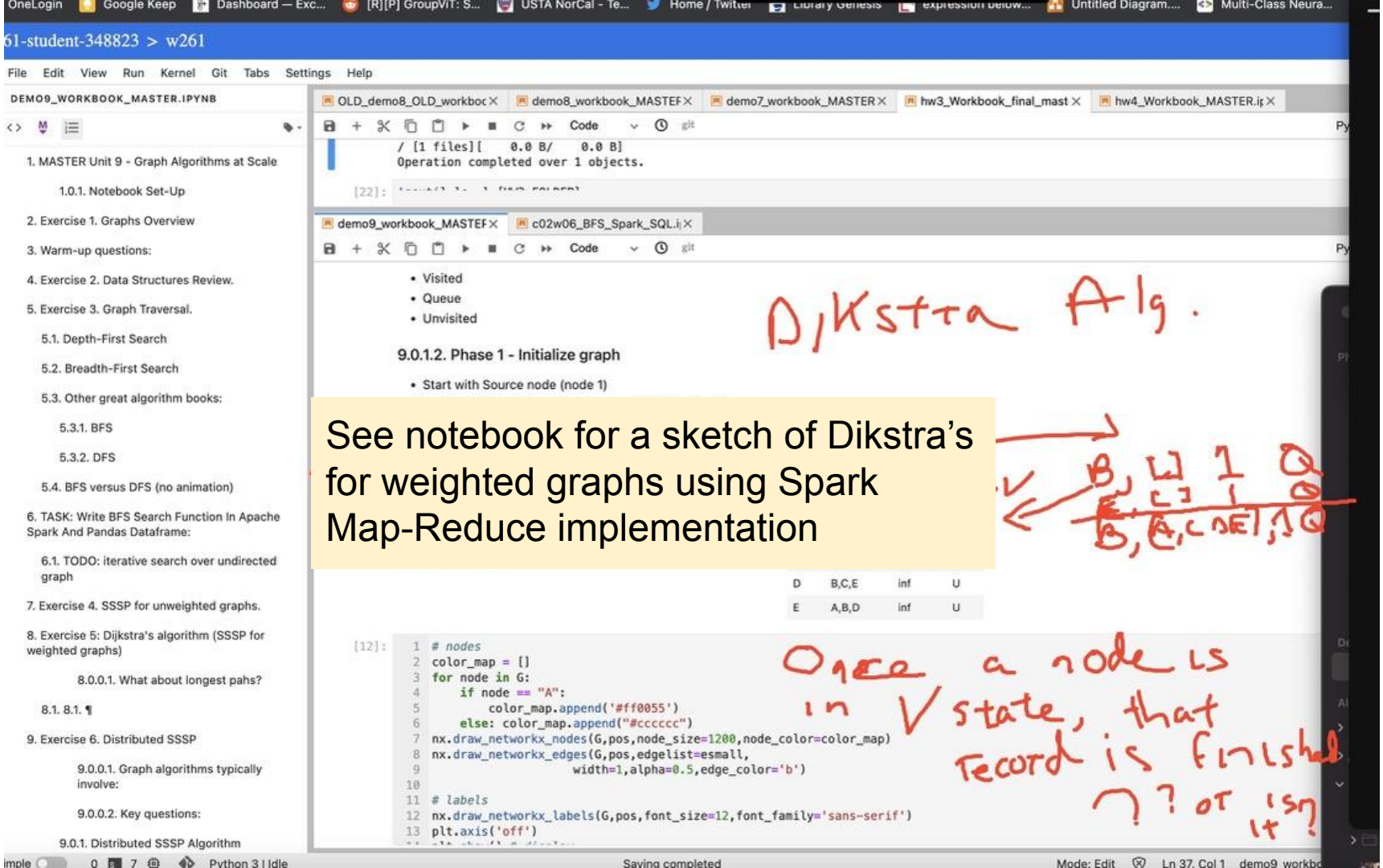
- Visit the unvisited vertex with the smallest known distance from the start vertex
- For the current vertex, examine its unvisited neighbours
- For the current vertex, calculate distance of each neighbour from start vertex
- If the calculated distance of a vertex is less than the known distance, update the shortest distance
- Update the previous vertex for each of the updated distances
- Add the current vertex to the list of visited vertices

Until all vertices visited

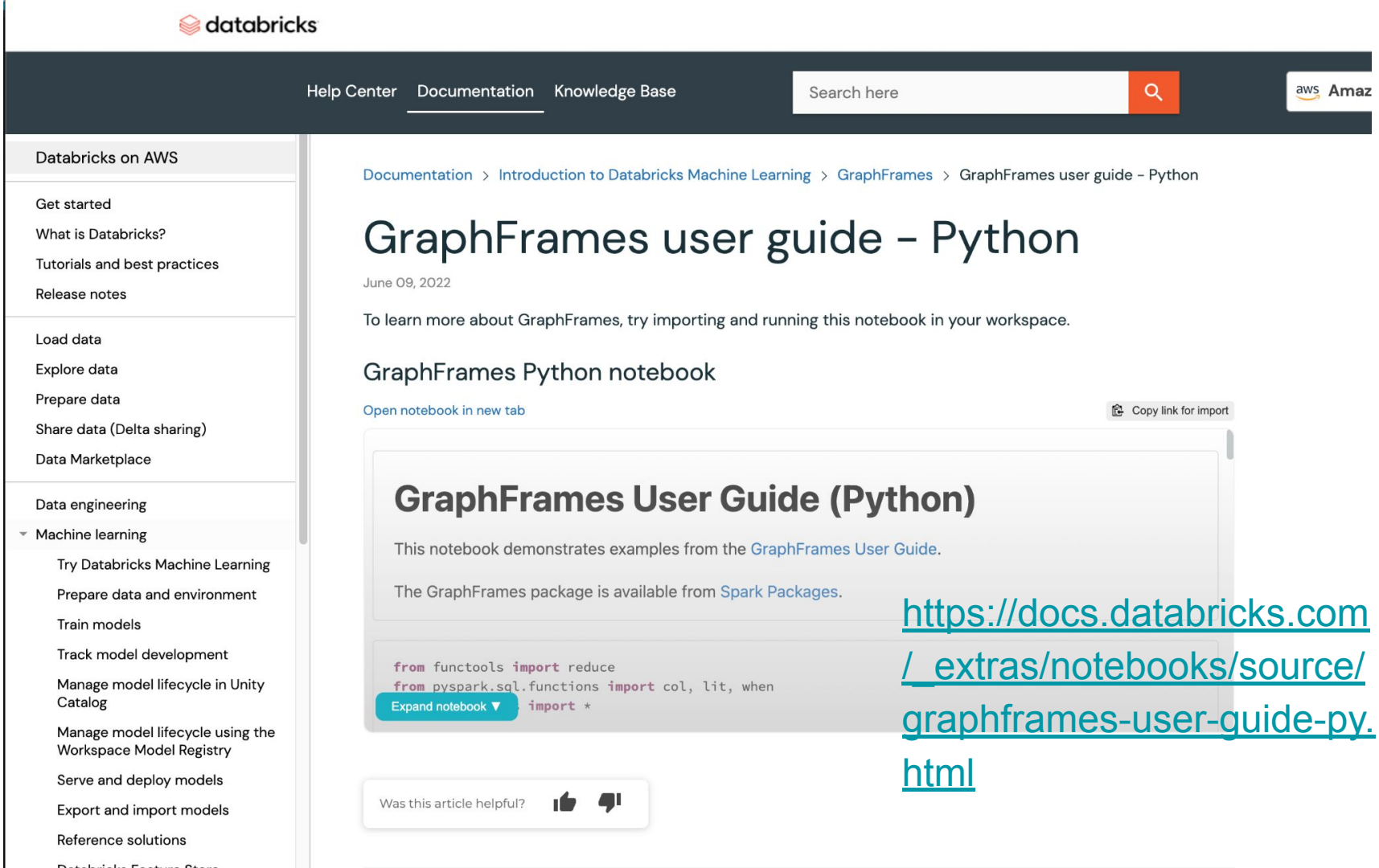
Algorithm

Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

WHILE vertices remain unvisited
    Visit unvisited vertex with smallest known distance from start vertex (call this 'current vertex')
    FOR each unvisited neighbour of the current vertex
        Calculate the distance from start vertex
        If the calculated distance of this vertex is less than the known distance
            Update shortest distance to this vertex
            Update the previous vertex with the current vertex
        end if
    NEXT unvisited neighbour
    Add the current vertex o the list of visited vertices
END WHILE


61-student-348823 > w261
File Edit View Run Kernel Git Tabs Settings Help
DEMO9_WORKBOOK_MASTER.IPYNB
1. MASTER Unit 9 - Graph Algorithms at Scale
1.0.1. Notebook Set-Up
2. Exercise 1. Graphs Overview
3. Warm-up questions:
4. Exercise 2. Data Structures Review.
5. Exercise 3. Graph Traversal.
5.1. Depth-First Search
5.2. Breadth-First Search
5.3. Other great algorithm books:
5.3.1. BFS
5.3.2. DFS
5.4. BFS versus DFS (no animation)
6. TASK: Write BFS Search Function In Apache Spark And Pandas Dataframe:
6.1. TODO: iterative search over undirected graph
7. Exercise 4. SSSP for unweighted graphs.
8. Exercise 5: Dijkstra's algorithm (SSSP for weighted graphs)
8.0.0.1. What about longest pahs?
8.1. 8.1. ¶
9. Exercise 6. Distributed SSSP
9.0.0.1. Graph algorithms typically involve:
9.0.0.2. Key questions:
9.0.1. Distributed SSSP Algorithm
OLD_demo8_OLD_workboc
demo8_workbook_MASTEF
demo7_workbook_MASTER
hw3_Workbook_final_mast
hw4_Workbook_MASTER.ip
Code
/ [1 files][0.0 B/ 0.0 B]
Operation completed over 1 objects.
demo9_workbook_MASTEF
c02w06_BFS_Spark_SQL.i
Code
• Visited
• Queue
• Unvisited
9.0.1.2. Phase 1 - Initialize graph
• Start with Source node (node 1)
Dikstra Alg.
See notebook for a sketch of Dikstra's for weighted graphs using Spark Map-Reduce implementation
D B,C,E inf U
E A,B,D inf U
[12]:
# nodes
color_map = []
for node in G:
if node == "A":
color_map.append('#ff0055')
else: color_map.append("#cccccc")
nx.draw_networkx_nodes(G,pos,node_size=1200,node_color=color_map)
nx.draw_networkx_edges(G,pos,edgelist=esmall,
width=1,alpha=0.5,edge_color='b')
# labels
nx.draw_networkx_labels(G,pos,font_size=12,font_family='sans-serif')
plt.axis('off')
Once a node is in V state, that record is finished ?? or isn't it?
Python 3 | Idle
Saving completed
Mode: Edit
Ln 37, Col 1
demo9_workbo

Spark's GraphFrames library



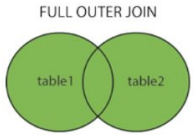
https://docs.databricks.com/_extras/notebooks/source/graphframes-user-guide-py.html

Full outer join

Tip: `FULL OUTER JOIN` and `FULL JOIN` are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



Note: `FULL OUTER JOIN` can potentially return very large result-sets!

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

https://www.w3schools.com/sql/sql_join_full.asp

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

A selection from the result set may look like this:

CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

Note: The `FULL OUTER JOIN` keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

https://www.w3schools.com/sql/sql_join_full.asp