# 13.1 graph algorithms

Monday, November 28, 2022 15:21

Reminder: pick a partner for lab 9! Your lab 9 top-down designs are due WEDNESDAY!
This week in lab: test 3!

TODAY: more graph algorithms
- Shortest Length Path BFS
- single Source Shortest Path

Recap/warm up:

Graphs get used for many applications.

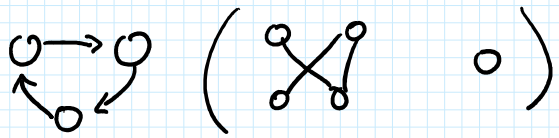
Q: What is a graph?
A way to model relationships

V = set of vertices
E = set of edges

graph: $G = (V, E)$

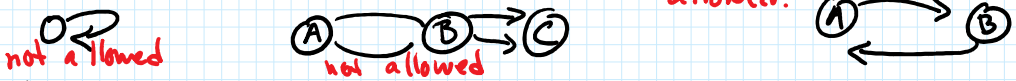
Edges can have weights and labels.
Edges can be directed or undirected.

Q: Give an example graph.
the internet
representing a maze

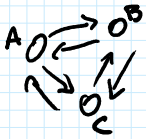


Q. What is the difference between a directed graph and an undirected graph?
Directed graph: edges have source & destination A → B
Undirected graph: edges just connect 2 vertices A — B

Q. What is a SIMPLE graph?
A graph with no loops and no duplicate edges.




Q. Assume a simple, directed graph has n vertices.
What is the maximum number of edges it could have?




Each vertex can have n-1 edges outgoing.
There are n vertices. Total # of edges = (# vertices)(outgoing edges from each vertex)
$= n \cdot (n-1)$
$= n^2 - n$

Q. Assume a simple, undirected graph has n vertices.
What is the maximum number of edges it could have?

Half as many as the previous question! $\frac{n(n-1)}{2}$

# GRAPH ADT

templated on 3 things:
- V vertex label type (must be unique)
- E edge label type (can have duplicates)
- W edge weight type (usually numerical)

methods:

```
void insertVertex(V vertex)
void removeVertex(V vertex)
void insertEdge(V source, V destination, E label, W weight)
void removeEdge(V source, V destination)
vector<Edge<V,E,W>> getEdges()
vector<V> getVertices()
bool containsVertex(V vertex)
bool containsEdge(V source, V destination)
Edge<V,E,W> getEdge(V source, V destination)
vector<Edge<V,E,W>> getOutgoing(V vertex)
vector<Edge<V,E,W>> getIncoming(V vertex)
vector<V> getNeighbors(V vertex)
```

Last time we discussed these, we saw pseudocode for

reachableDFS(V start, V end, Graph<V,E,W>* g)

to answer the question, "Is it possible to go from the start vertex to the end vertex in this graph?"

Idea:

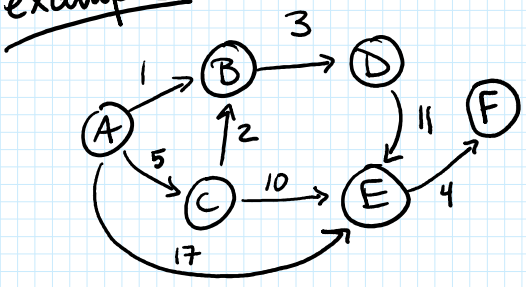
```
have a stack
push start onto stack
while stack not empty:
    pop from stack
    if we reached end, done! return T
    else: push neighbors onto stack
return F
```

(Note: it is also possible to answer this question with BFS: reachableBFS ...)

The plan: we'll do three graph algorithms:

1. bool reachableDFS(V start, V end, Graph<V,E,W>* g)
2. vector<V> shortestLengthPathBFS(V start, V end, Graph<V,E,W>* g)
   // ignores weights and returns the path with fewest number of edges
3. Dictionary<V,W>* singleSourceShortestPath(V start, Graph<V,E,W>* g)
   // finds the length of the shortest path from start to every vertex in graph

example:

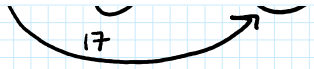



Q: What does reachableDFS(A,E) return?
TRUE

Q: What does reachableDFS(D,C) return?
FALSE

Q: What should shortestLengthPath(A,F) return?
vector containing A, E, F

Q: What should shortestLengthPath(F,B) return?




vector containing A, E, F

Q: What should shortestLengthPath(E, B) return?

empty vector because it's not possible to go from E to B

algorithm 2:

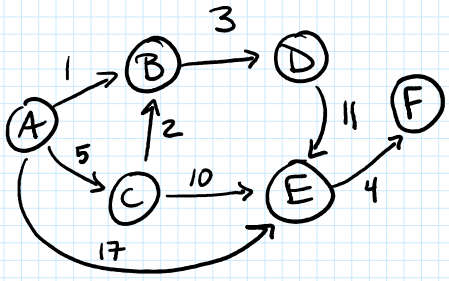
vector<V> shortestLengthPathBFS(V start, V end, Graph<V,E,W>* g)

//ignores weights and returns the path with fewest number of edges

## pseudocode to find shortest path

```
vector<V> shortestLengthPathBFS(V start, V end, Graph<V,E,W>* g)
    vector<V> path  // empty path
    Queue<V> queue
    Dictionary<V,V> previous
    enqueue start vertex
    insert start, start into previous dictionary
    While queue is not empty
    |   current = dequeue from queue
    |   if current == end  // we found the goal!
    |   |   use previous dictionary to fill the path vector
    |   |   return path
    |   else
    |   |   for every neighbor of current
    |   |   |   if previous does not contain that neighbor
    |   |   |   |   insert neighbor, current into previous dictionary
    |   |   |   |   enqueue neighbor
    return path  // if we reach this line, return the empty path
```

## example run:




shortestLengthPath("A", "F", graph)

queue: ~~A~~ ~~B~~ ~~C~~ ~~E~~ ~~D~~ ~~F~~

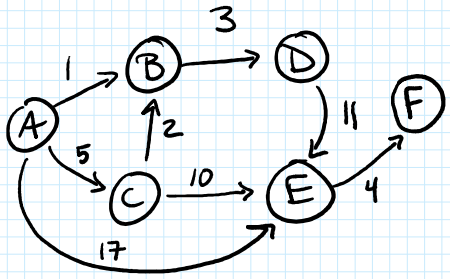
current: ~~A~~ ~~B~~ ~~C~~ ~~E~~ ~~D~~ F

path: A, E, F

previous: "A" → "A"
"B" → "A"
"C" → "A"
"E" → "A"
"D" → "B"
"F" → "E"

"C" → "A"
"E" → "A"

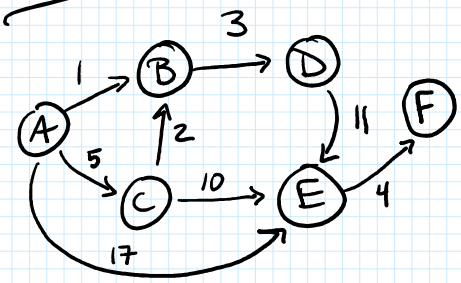
example run:




shortestLengthPath("C", "D", graph)

queue: ~~C~~ ~~B~~ ~~E~~ ~~D~~ F
current: ~~C~~ ~~B~~ ~~E~~ D
path: C, B, D
previous: "C" → "C" "E" → "C" "F" → "E"
"B" → "C" "D" → "B"

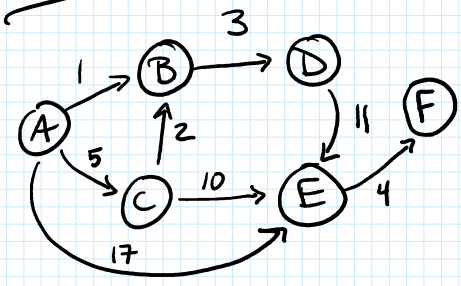
example run:




shortestLengthPath("B", "C", graph)

queue: ~~B~~ ~~D~~ ~~E~~ ~~F~~
current: ~~B~~ ~~D~~ ~~E~~ F
path: ← returns empty vector!
previous: "B" → "B" "E" → "D"
"D" → "B" "F" → "E"

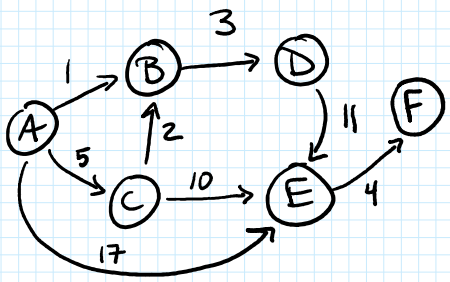
example run:




shortestLengthPath("D", "D", graph)

queue: ~~D~~
current: D
path: D
previous: "D" → "D"

③ Dictionary<V, W>* singleSourceShortestPath (V start, Graph<V,E,W>* g)
// finds the length of the shortest path from start to every vertex in graph




singleSourceShortestPath("A", graph)

Should return:

"A" → 0
"B" → 1
"C" → 5
"D" → 4
"E" → 15
"F" → 19