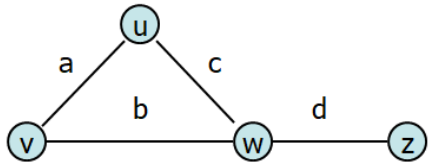
CS 225

**#35: Adjacency List + BFS**

November 16, 2020 · *G Carl Evans*

## Graph Implementation #3: Adjacency List




| Vertex List | Edges |
|---|---|
| u | a |
| v | b |
| w | c |
| z | d |

**Operations on an Adjacency Matrix implementation:**

insertVertex(K key):

removeVertex(Vertex v):

incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

insertEdge(Vertex v1, Vertex v2, K key):

## Running Times of Classical Graph Implementations

| | Edge List | Adj. Matrix | Adj. List |
|---|---|---|---|
| **Space** | **n+m** | **$n^2$** | **n+m** |
| **insertVertex** | **1** | **n** | **1** |
| **removeVertex** | **m** | **n** | **deg(v)** |
| **insertEdge** | **1** | **1** | **1** |
| **removeEdge** | **1** | **1** | **1** |
| **incidentEdges** | **m** | **n** | **deg(v)** |
| **areAdjacent** | **m** | **1** | **min( deg(v), deg(w) )** |

## Big Picture Ideas: Comparing Implementations

**Q:** If we consider implementations of simple, connected graphs, what relationship between n and m?

- On connected graphs, is there one algorithm that underperforms the other two implementations?

...what if our graph is sparse and not connected?

**Q:** Is there clearly a single best implementation?

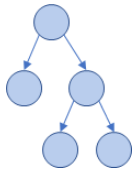
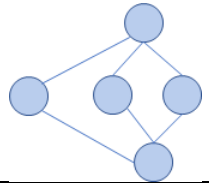
- Optimized for fast construction:

- Optimized for areAdjacent operations:

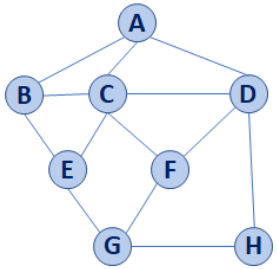
## Graph Traversal

**Objective:** Visit every vertex and every edge in the graph.
**Purpose:** Search for interesting sub-structures in the graph.

We've seen traversal before – this is different:

| BST | Graph |
|---|---|
|  |  |

### BFS Graph Traversal:




**Pseudocode for BFS**

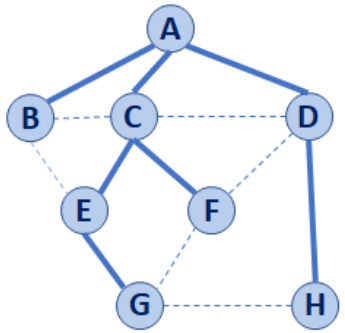
```
1   BFS(G):
2     Input: Graph, G
3     Output: A labeling of the edges on
4         G as discovery and cross edges
5
6     foreach (Vertex v : G.vertices()):
7       setLabel(v, UNEXPLORED)
8     foreach (Edge e : G.edges()):
9       setLabel(e, UNEXPLORED)
10    foreach (Vertex v : G.vertices()):
11      if getLabel(v) == UNEXPLORED:
12         BFS(G, v)
13
14  BFS(G, v):
15    Queue q
16    setLabel(v, VISITED)
17    q.enqueue(v)
18
19    while !q.empty():
20      v = q.dequeue()
21      foreach (Vertex w : G.adjacent(v)):
22        if getLabel(w) == UNEXPLORED:
23           setLabel(v, w, DISCOVERY)
24           setLabel(w, VISITED)
25           q.enqueue(w)
26        elseif getLabel(v, w) == UNEXPLORED:
27           setLabel(v, w, CROSS)
```

| Vertex (v) | Distance (d) | Prev. (p) | Adjacent |
|---|---|---|---|
| A | | | |
| B | | | |
| C | | | |
| D | | | |
| E | | | |
| F | | | |
| G | | | |
| H | | | |

### BST Graph Observations

1. Does our implementation handle disjoint graphs? How?

   a. How can we modify our code to count components?




2. Can our implementation detect a cycle? How?

**CS 225 – Things To Be Doing:**

1. mp_mazes due today
2. Project goals due Wednesday
3. POTD ongoing