Department of Computer Science

Graph Representation in Java

Hooman Khosravi

CPSC 319 - Data Structures, Algorithms, and Their Applications

UNIVERSITY OF CALGARY

Requirments

- Please read the lecture material first and then continue this tutorial:
- http://pages.cpsc.ucalgary.ca/~hudsonj/CPSC319W20/Slides/CPSC319-8-1-Graphs.pdf

UNIVERSITY OF CALGARY

Tree vs Graph

Quick question:

- What are the differences between graphs and trees?

UNIVERSITY OF CALGARY

Tree vs Graph

- Well there are a number of differences, but one important one in my opinion is that:
  - In a tree there are no cycles, meaning that if start moving from a node you cannot come back to that node with passing each “edge” only once. But in graphs you can have loops.

- https://www.geeksforgeeks.org/difference-between-graph-and-tree/

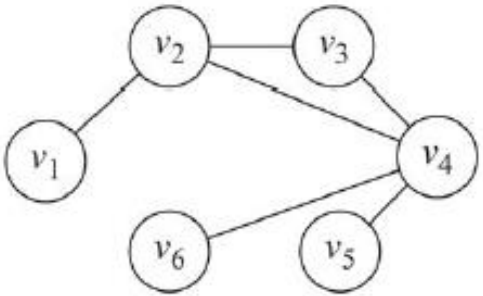
UNIVERSITY OF CALGARY

Representation

- Now we're going to talk about two Graph Representation techniques:
  - Adjacency Matrix
  - Adjacency List
- Graph representation techniques allows us to store a graph into the memory therefore we will be able to work with them using programming languages like Java for example.
- First let's have a quick reminder of what these two techniques are.

Adjacency Matrix

- Vertices represent row and columns
- Edges are defined by values in the matrix.
- For more detailed info please checkout the lectures if you haven't done so already



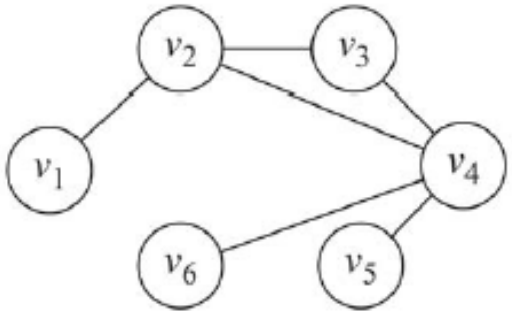
(a) Graph

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	0	0	0	0
v_2	1	0	1	1	0	0
v_3	0	1	0	1	0	0
v_4	0	1	1	0	1	1
v_5	0	0	0	1	0	0
v_6	0	0	0	1	0	0

(b) Adjacency Matrix

Adjacency List

- We have a table (or an array) with the same size as the number of vertices.
- In that table, each element represents one vertex and is a list of variable size, listing all the other vertices that are connected to that vertex.



(a) Graph

Table 2.1: Adjacency List

Node	Connected To
v_1	v_2
v_2	v_1, v_3, v_4
v_3	v_2, v_4
v_4	v_2, v_3, v_5, v_6
v_5	v_4
v_6	v_4

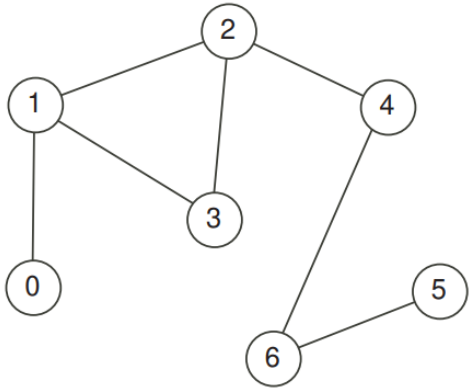
UNIVERSITY OF CALGARY

- Now the question is:
  - What are the practical differences between these two method?
  - When would use one and not the other?
- I will try to answer these by asking you a series of question.
- Please think about the answers before checking the answer slide!

UNIVERSITY OF CALGARY

Adjacency Matrix

1. Consider an ***adjacency matrix*** representation of an undirected graph $G = (V, E)$. Let $n = |V|$ be the number of vertices in this graph and let $m = |E|$ be the number of edges in this graph.

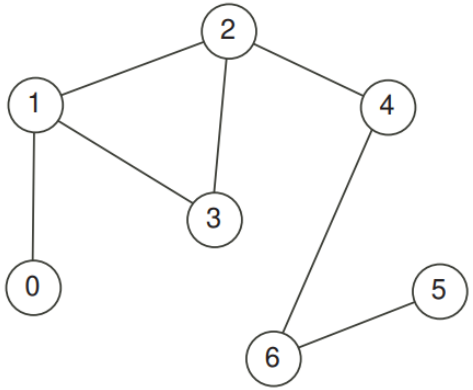


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

UNIVERSITY OF CALGARY

Adjacency Matrix

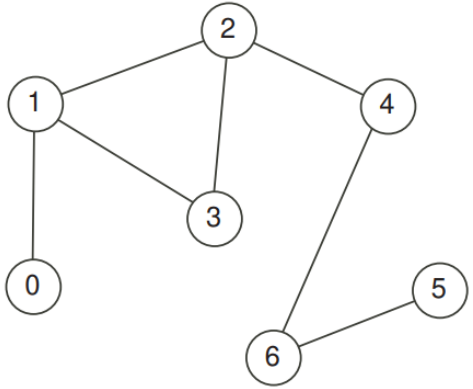
(a) Describe the amount of ***storage space*** used by this representation of the graph G. This — and the answer to later questions — might depend on n, m, or both.



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix

Solution: Assuming the uniform cost criterion — and that vertices have names $0, 1, \ldots, n-1$, where $n = |V|$, $\Theta(n^2)$ storage locations are needed. Indeed, if the only data represented is the number n of vertices, and the matrix used to store information about edges, then $n^2 + 1$ storage locations are used.

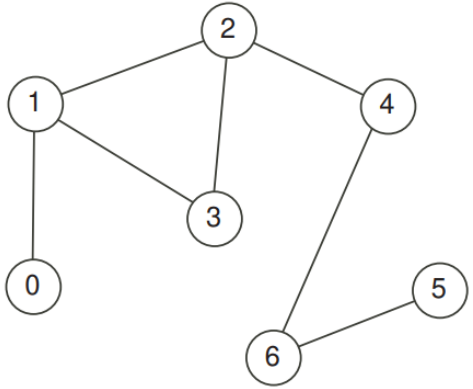


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

UNIVERSITY OF CALGARY

Adjacency Matrix

(b) Describe how to decide whether a given pair of vertices are ***neighbours*** in G. Describe the number of steps that an algorithm would use to decide this, using this representation of G, in the worst case.

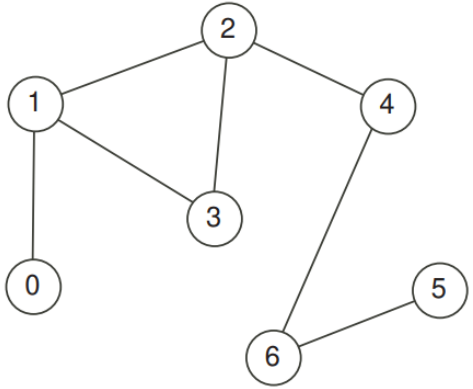


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix

Solution: A `NoSuchElementException` should (probably) be thrown if the inputs are integers i and j that are *not* both between 0 and $n-1$, inclusive. Otherwise, if `A` is the matrix used to store information about edges, then `true` should be returned if $\texttt{A}[i,j] = 1$, and `false` should be returned otherwise.

This operation can be performed using a constant number of steps in the worst case.

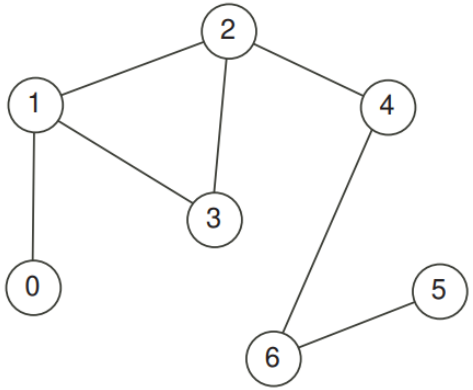


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

UNIVERSITY OF CALGARY

Adjacency Matrix

(c) Describe how to ***add an edge*** between a given pair of vertices in G, using this representation. How many steps are needed to do this in the worst case?

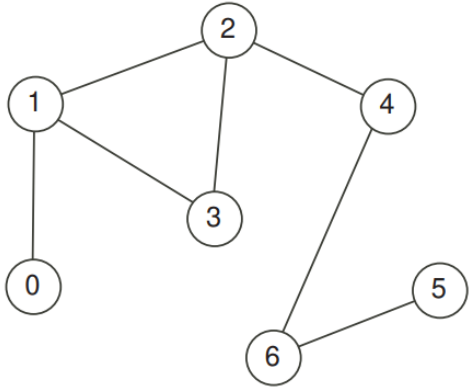


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix

Solution: Once again, a `NoSuchElementException` should (probably) be thrown if the inputs are not a pair of integers i and j such that $0 \leq i, j \leq n-1$ where $n = |V|$. An `IllegalArgumentException` should be thrown if $i = j$. Another exception — possibly called an `EdgeFoundException` should be thrown if $\mathtt{A}[i, j] = 1$ already. In the only remaining case $\mathtt{A}[i, j]$ and $\mathtt{A}[j, i]$ should both set to be 1.

This operation can also be performed using a constant number of steps in the worst case.

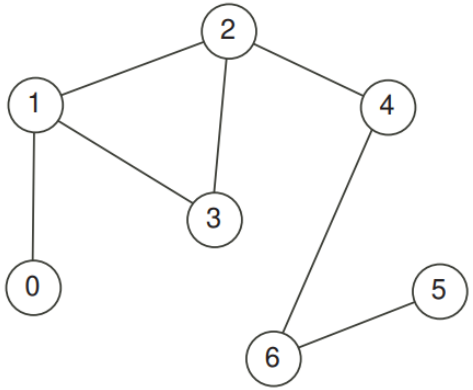


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

UNIVERSITY OF CALGARY

Adjacency Matrix

(d) Describe how to ***add a vertex*** in G. How many steps are needed in the worst case?

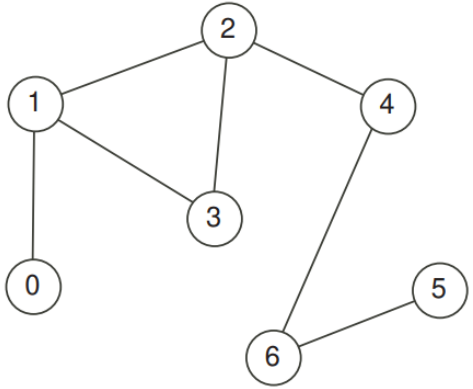


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix

Solution: If an array is used then it is necessary to make a copy of the matrix, with one more row and column each filled with 0's. $\Theta(n^2)$ steps will be required in this case, if the graph originally had n vertices.

If an `ArrayList` is used instead of an array then it is possible to use the `add` operation to extend rows, and to add a new one. It can be shown that the amortized cost will be smaller (indeed, an amortized cost linear in n can be established) but number of steps used in the worst case will still be in $\Theta(n^2)$.

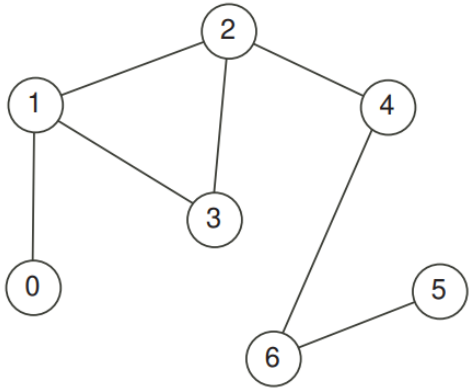


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

UNIVERSITY OF CALGARY

Adjacency Matrix

(e) Describe how to ***list the neighbours*** of a given vertex. How many steps are needed in the worst case?

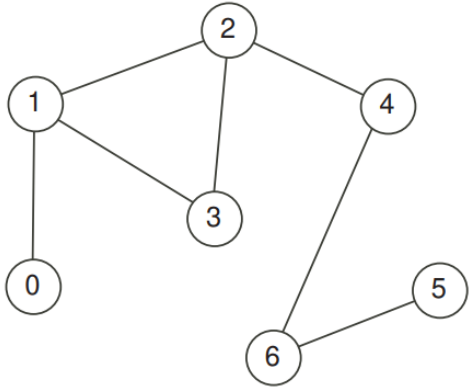


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix

Solution: If the input is not an integer between 0 and $n-1$ (inclusive), where n is the current size of V, then a `NoSuchElementException` should be thrown. Otherwise the entries in row i should be checked: For $0 \leq j \leq n-1$, j should be listed as a neighbour if and only if $\mathtt{A}[i,j] = 1$.

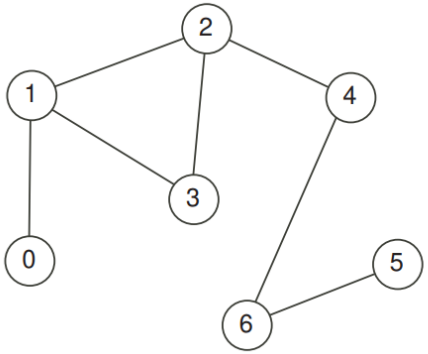
$\Theta(n)$ steps are used in the worst case.

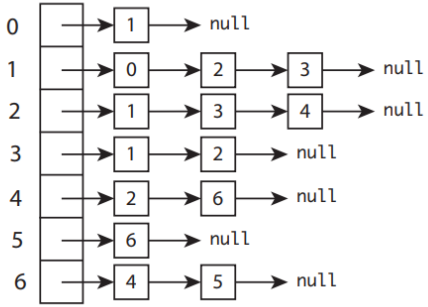


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency List

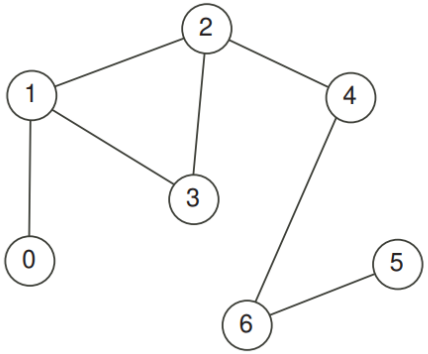
2. Repeat the above question, for an ***adjacency list*** representation instead of an adjacency matrix representation of an undirected graph G.

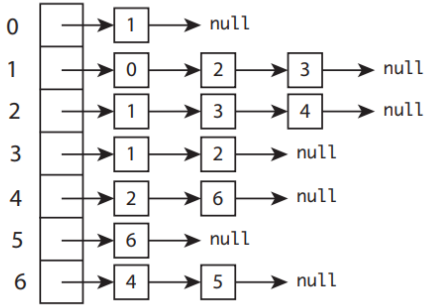




Adjacency List

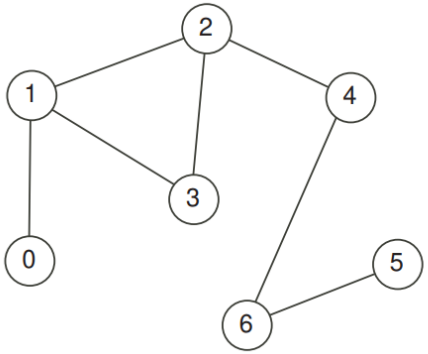
(a) Describe the amount of ***storage space*** used by this representation of the graph G. This — and the answer to later questions — might depend on n, m, or both.

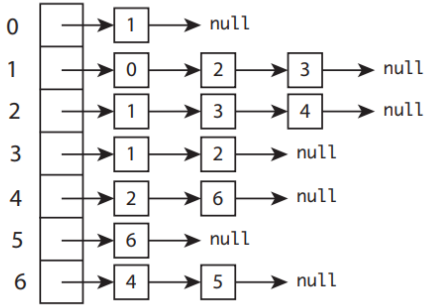




Adjacency List

(a) Assuming the uniform cost criterion and, once again, that vertices are named $01, 2, \ldots n-1$ where $n = |V|$, $\Theta(n+m)$ storage locations are used for this representation: One is needed to store n, n are used to represent the one-dimensional array with indices $0, 1, 2, \ldots, n-1$ and storing the heads of linked lists, and (including space for references to nodes) slightly less than another $2m$ locations are needed to store the linked lists of neighbours of nodes.

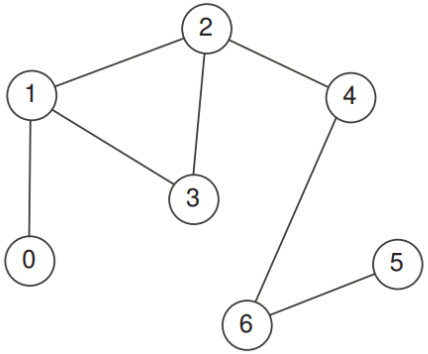


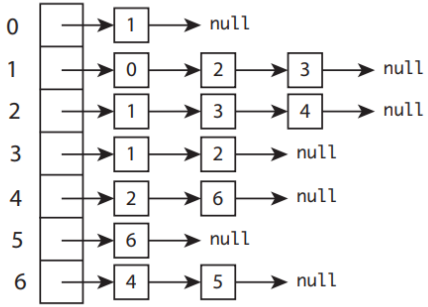


UNIVERSITY OF CALGARY

Adjacency List

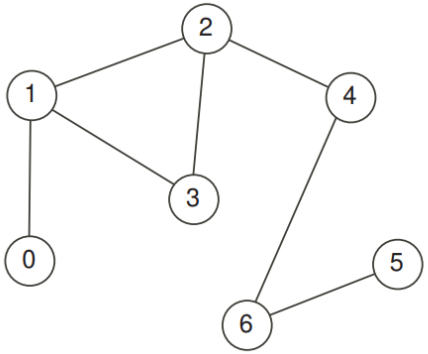
(b) Describe how to decide whether a given pair of vertices are ***neighbours*** in G. Describe the number of steps that an algorithm would use to decide this, using this representation of G, in the worst case.

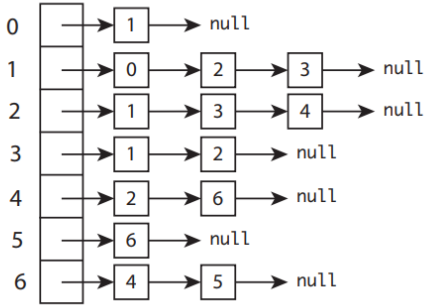




Adjacency List

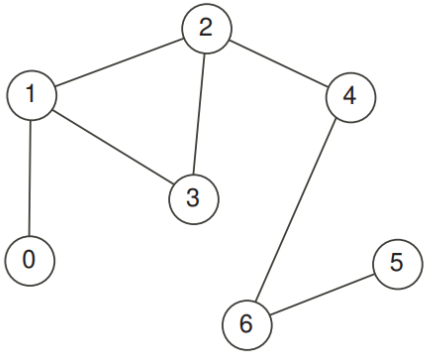
(b) As for part (b) above, a `NoSuchElementException` should be thrown if the inputs (`i j`) are not integers between 0 and $n-1$, inclusive. Otherwise, if `A` is the (one-dimensional) array to store heads of linked lists, then either the linked list at `A`[i] should be traversed to search for j, or the linked list at `A`[j] should be traversed to search for i — returning `true` if the desired integer is found, and `false` otherwise.. In the worst case the number of steps used will be linear in the maximum degree of any node in the graph — in $O(n)$, in the worst case.

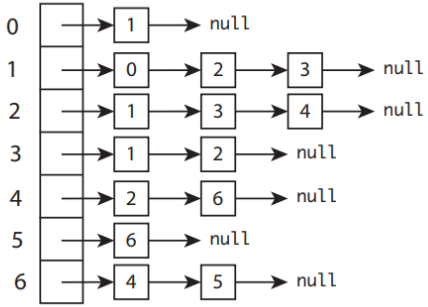




Adjacency List

(c) Describe how to ***add an edge*** between a given pair of vertices in G, using this representation. How many steps are needed to do this in the worst case?

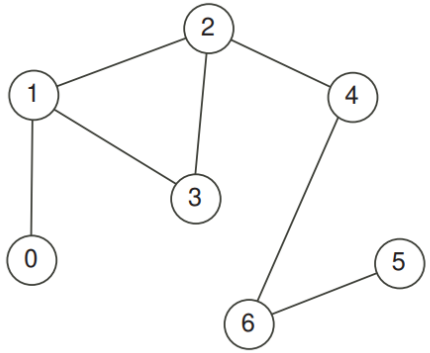


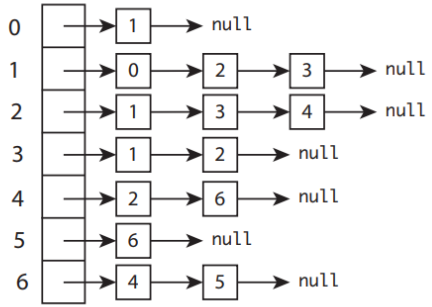


Adjacency List

(c) As in part (c), above, a `NoSuchElementException` should be thrown if the inputs are not a pair of integers i and j such that $0 \leq i, j \leq n-1$ where $n = |V|$. An `IllegalArgumentException` should be thrown if $i = j$. Either the linked list at `A[`i`]` should be traversed in search for j, or the linked list at `A[`j`]` should be traversed in search for i — with an `EdgeFoundException` thrown if the integer is found. Otherwise, j should be inserted into the linked list `A[`i`]` and i should be inserted into the linked list at `A[`j`]`.

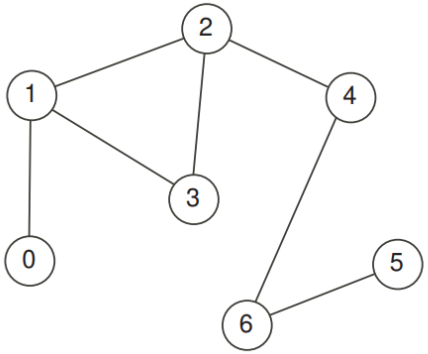
The number steps used by this process is at most linear in the maximum degree of any node — in $O(n)$, in the worst case.

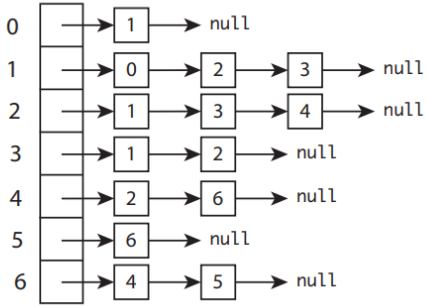




Adjacency List

(d) Describe how to ***add a vertex*** in G. How many steps are needed in the worst case?

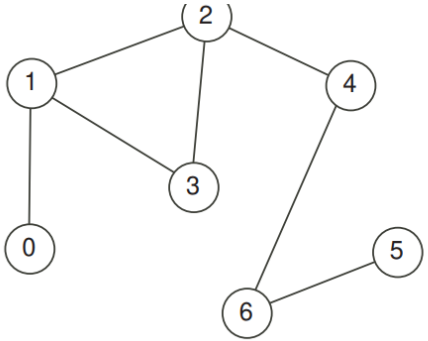


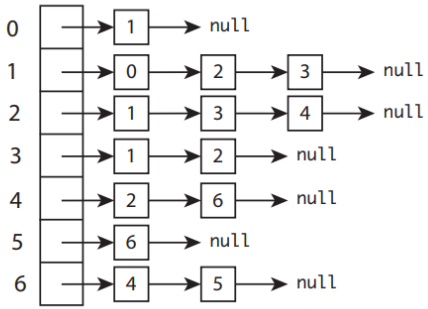


Adjacency List

(d) If an array is used to store linked lists then it is necessary to copy this array, with one more entry added. References to linked lists can the copied from the existing array to corresponding positions in the new array, and the final entry can be set to `null`. $\Theta(n)$ steps are required in the worst case.

If an `ArrayList` is used to store linked lists then the `add` operation can be used to extend the `ArrayList` instead. The amortized cost of a sequence of operations will be reduced — indeed, a constant bound can be established — but the number of steps used will still be in $\Theta(n)$ in the worst case.

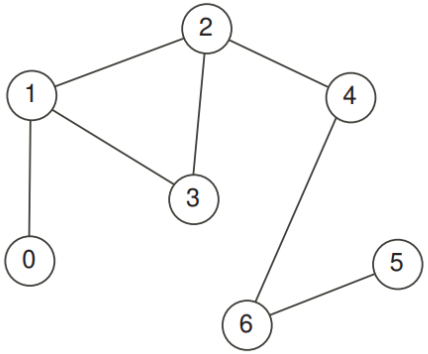


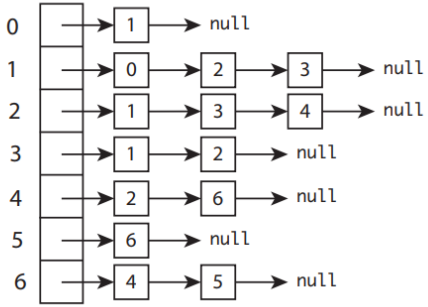


UNIVERSITY OF CALGARY

Adjacency List

(e) Describe how to ***list the neighbours*** of a given vertex. How many steps are needed in the worst case?

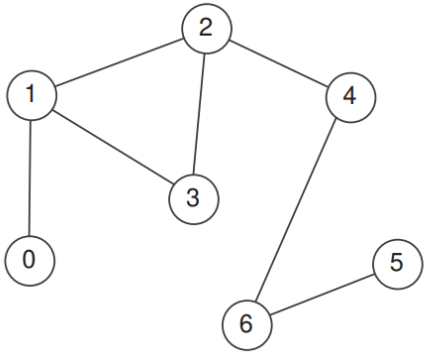


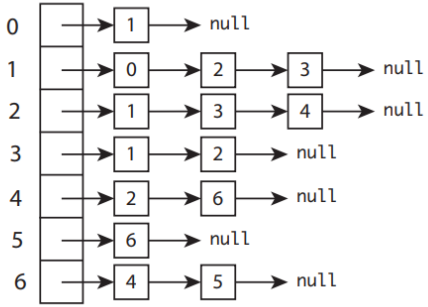


Adjacency List

(e) As for (e) above, a `NoSuchElementException` should be thrown if the input is not an integer i such that $0 \leq i \leq n-1$, where $n = |V|$. Otherwise the linked list at `A[`i`]` should be traversed, with each integer stored in the list included as a neighbour of i.

The number of steps used in the worst case is linear in the largest degree of any node in the graph — in $\Theta(n)$, in the worst case.





UNIVERSITY OF CALGARY

Adjacency List

- Allright! Let's do some coding now:
- Implementing the Adjacency List representation of graph in Java.
- Implement a weighted graph
  - Each element in the linked list should have a vertex and a weight.
  - It is better to have a subclass edge that has contains both destination vertex and the weight
- Construct a graph with n vertecies
  - new graph(5) //five vertecies
- Implement add edge. It should take two vertex index and a weight.
- Implement remove edge. It should take two vertex index.
- Implement getWeight. It should take two vertices and return the weight
- Implement a print function to see all the edges connected to each vertex. Use this for testing.

Adjacency List

```java
public static void main(String[] args) {

    GraphAdjacencyList G1 = new GraphAdjacencyList(3);

    G1.addEdge(0,2,0.5);
    G1.addEdge(0,1,1.5);
    G1.addEdge(1,2,1.3);


    G1.print();
    G1.addEdge(1,2,1.4);
    G1.removeEdge(0, 1);

    G1.print();

    try {
        System.out.println("Edge weight between v1 and v2 is:" + G1.getWeight(1,2));
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

```
V0 is connected to V2(0.5) V1(1.5)
V1 is connected to V0(1.5) V2(1.3)
V2 is connected to V0(0.5) V1(1.3)

V0 is connected to V2(0.5)
V1 is connected to V2(1.4)
V2 is connected to V0(0.5) V1(1.3)

Edge weight between v1 and v2 is:1.4
```

- Your code should work with this main function and produce the same result.

UNIVERSITY OF CALGARY

Adjacency List

- First try your best to impalement this yourself.
- I will include my code in the website so you can compare afterwards.

UNIVERSITY OF CALGARY

Adjacency Matrix

- Also, Implement the Adjacency Matrix representation of graph in Java.
- Implement a weighted graph
  - You can use either a two-dimensional array list or array.
- Construct a graph with n vertecies
  - new graph(5) //five vertecies
- Implement add edge. It should take two vertex index and a weight.
- Implement remove edge. It should take two vertex index.
- Implement getWeight. It should take two vertices and return the weight
- Implement a print function to see all the edges connected to each vertex. Use this for testing.

UNIVERSITY OF CALGARY

Adjacency Matrix

```
public static void main(String[] args) {

    GraphMatrixList G1 = new GraphMatrixList(3);

    G1.addEdge(0,2,0.5);
    G1.addEdge(0,1,1.5);
    G1.addEdge(1,2,1.3);

    G1.print();
    G1.addEdge(1,2,1.4);
    G1.removeEdge(0, 1);

    G1.print();

    try {
        System.out.println("Edge weight between v1 and v2 is:" + G1.getWeight(1,2));
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

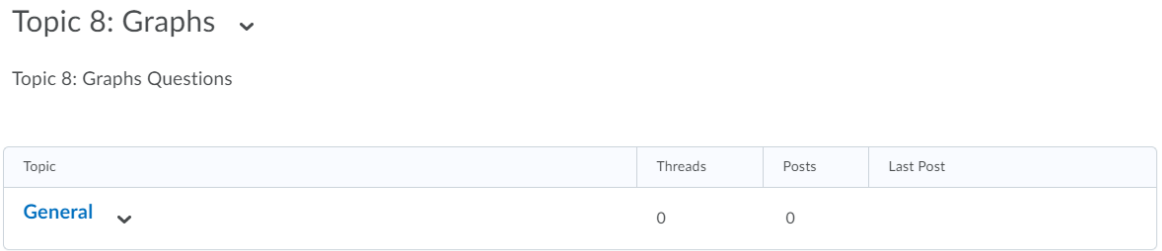
```
V0 is connected to V2(0.5) V1(1.5)
V1 is connected to V0(1.5) V2(1.3)
V2 is connected to V0(0.5) V1(1.3)

V0 is connected to V2(0.5)
V1 is connected to V2(1.4)
V2 is connected to V0(0.5) V1(1.3)

Edge weight between v1 and v2 is:1.4
```

- Your code should work with this main function and produce the same result.

- If you have any questions about this tutorial, please post it here:



- Or email me: hooman.khosravi1@ucalgary.ca