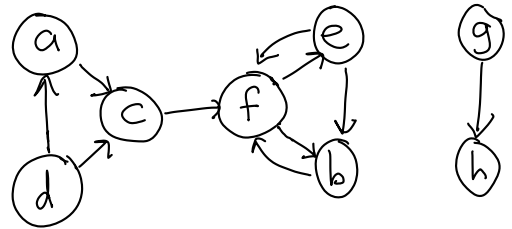


lecture 32

graphs

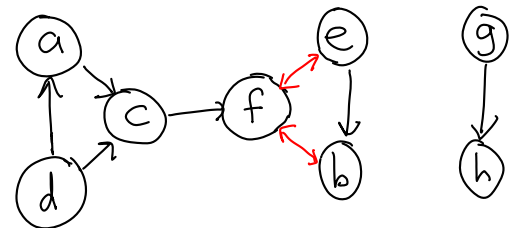
Example



A graph G is a set of "vertices" V (also called "nodes") and a set of edges E .

An edge e is an ordered pair of vertices (v_1, v_2) .

Same Example - different notation



→ one edge
↔ two edges

Examples of Graphs

vertices

edges

airports

flights

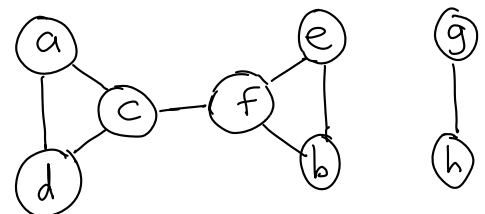
web pages

links (URLs)

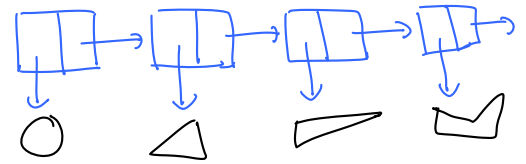
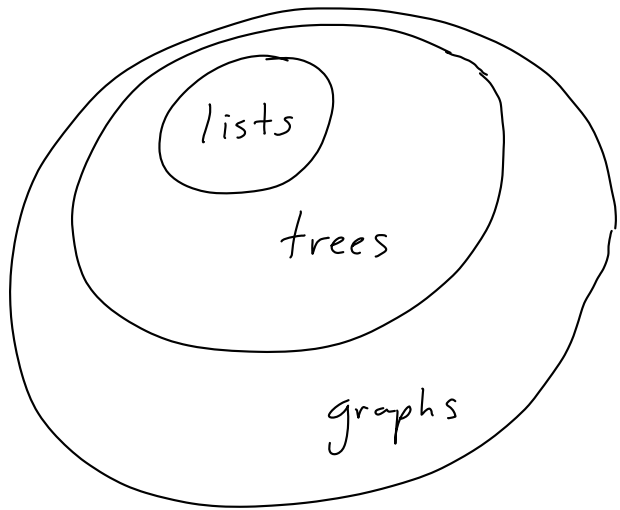
Java objects

references

Undirected Graphs



Edge (v, w) is equivalent to edge (w, v) .



```

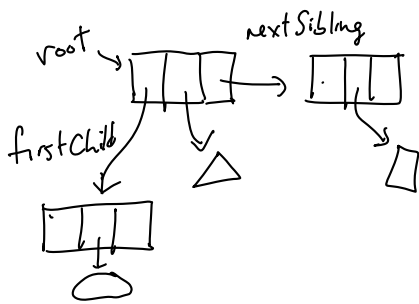
class SLinkedList <E> {
    SNode <E> head;
    :
}

```

```

class Tree {
    TNode <E> root;
}

```



```

class Graph {
    ?
}

```

There is no special "head" or "root" node.
 There is no standard way
 Let's look at a few examples.

Data structures for graphs?

vertices:

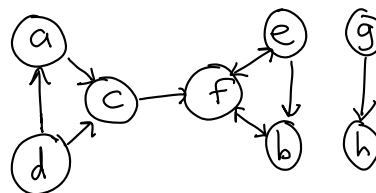
- a class V

edges:

- list of vertices V
- boolean [][]

"Adjacency List"

Example: the vertices are characters

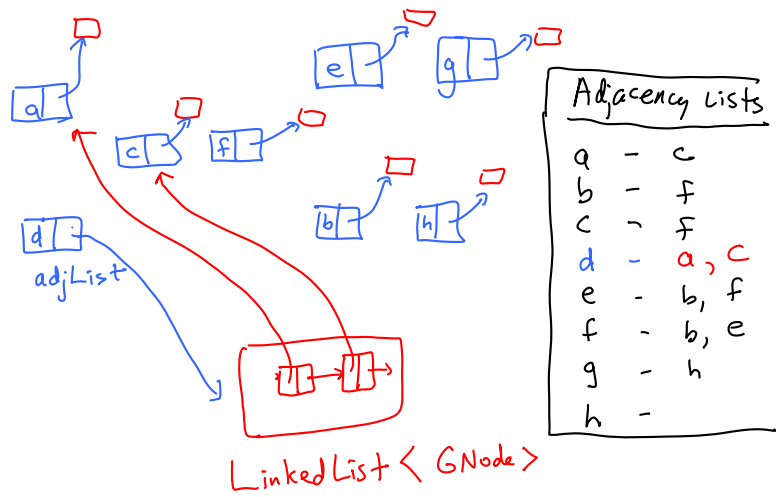


- a - c
- b - f
- c - f
- d - a, c
- e - b, f
- f - b, e
- g - h
- h -

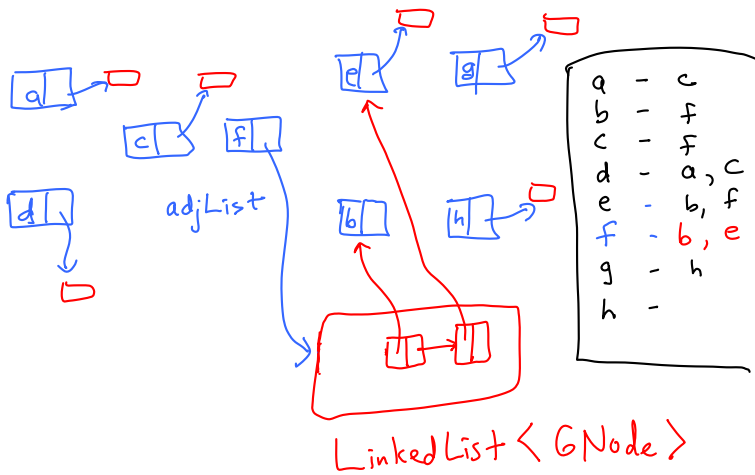
Example (Java)

```

class GNode {
    char vertex
    LinkedList<GNode> adjList
}
    
```



*Note: each vertex (GNode) has an adjacency list



* each vertex (GNode) has an adjacency list

More generally

a vertex will be an object of some class V .

```

class GNode<V> {
    V vertex
    LinkedList<GNode<V>> adjList
}
    
```

How to represent the graph?

```

class Graph {
    LinkedList<GNode<V>> nodeList
    :
}
    
```

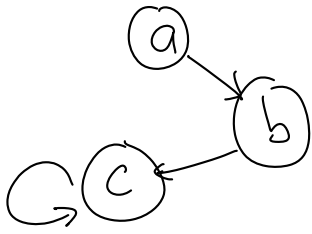
Alternatively (better)

```

class Graph<K, V> {
    HashMap<K, GNode<V>> map
    ↑      ↑
    key   value
}
    
```

The key K might be the vertex itself or some field in the vertex eg. name of airport.

Another example



	a	b	c
a	0	1	0
b	0	0	1
c	0	0	1

Adjacency Lists vs. Adjacency Matrix

- space efficiency ?
- time efficiency ?

See lecture notes for brief discussion

Terminology

- in degree
- out degree
- path
- cycle

see lecture notes for definition

eg. (e, b, f, e)
(e, f, e).

