Melbourne Bioinformatics

BIOINFORMATICS + DATA SERVICES + INFRASTRUCTURE, FOR LIFE SCIENCES TODAY

THE UNIVERSITY OF MELBOURNE

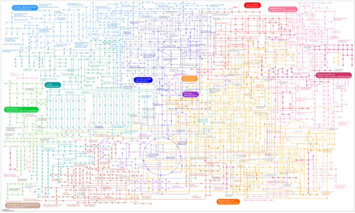
COMP90014

Algorithms for Bioinformatics

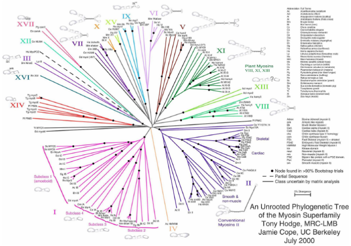
Week 3A - Introduction to Graph Theory

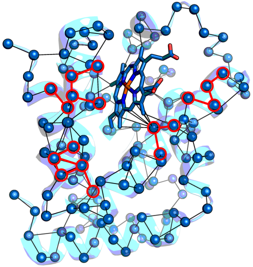
Introduction to graph theory

Graphs in biology



- graphs model objects and their interactions
- biology is full of interactions
- graphs are everywhere
- mathematically **describe and analyse** relationships

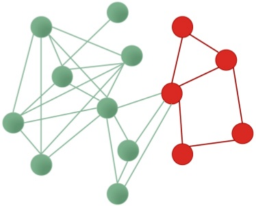




Graphs

A ***graph*** G is represented by a pair $G = (V, E)$

- set of vertices V and a set of edges E
- a ***subgraph*** contains subsets of V and E



Nodes (or vertices)

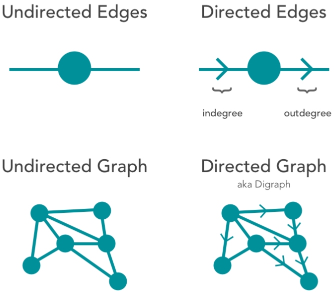
- represent physical objects, concepts, locations, *etc.*
- can have ***labels***

Edges

- represent relationships between pairs of objects
- can have ***labels***
- can be ***directional***
- can have ***weights***

- node and edge properties define different ***graph types***

Graph types



Undirected graphs

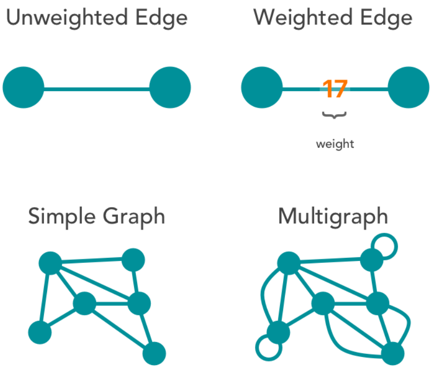
- edges are ***unordered*** pairs of vertices
- edge $\{u, v\}$
- *e.g.* protein-protein interactions

Directed graphs (digraph)

- edges (arcs) are ***ordered*** pairs of vertices
- (u, v) is different from (v, u)
- *e.g.* metabolic pathway

- u and v are ***adjacent***
- how many nodes are adjacent to u?
  - node degree (neighbourhood)
  - out-degree & in-degree for directed graphs

Graph types



Edges and nodes can have costs/weights

- weighted graphs
  - *e.g.* gene similarity
- unweighted graphs (binary relationships)
  - *e.g.* homologous genes

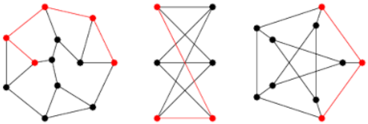
Simple graph

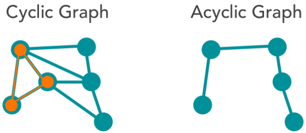
- only one edge between any two nodes

Multigraph

- allows multiple edges connecting two nodes

Graph types

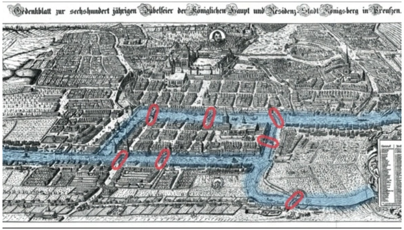


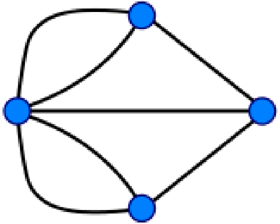


Walk: a connected sequence of edges
Path: a walk without repeated nodes
Cost: the sum of the edge costs/weights
e.g. indirect relationships
Shortest path: Path with minimum cost
Cycle/Circuit: a ***closed path*** (first and last nodes are the same)

- Graphs can be cyclic or acyclic

Special paths





Eulerian walk/cycle

- visits every edge exactly once
- *e.g. de novo* genome assembly
- Leonhard Euler in 1736
- seven bridges of Königsberg problem
- walk through the city crossing each of those bridges ***once and only once***
- ***no Eulerian walks*** on this graph

Hamiltonian path/cycle

- visits every node exactly once
- ***NP-complete***
- travelling salesperson problem:
  - find a Hamiltonian cycle of minimum cost

This is not how to solve the bridges of Königsberg problem

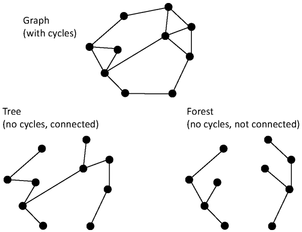
Trees (树)

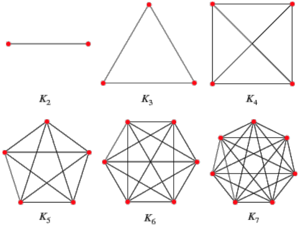
acyclic, undirected, connected graphs: 树是不包含任何循环的图，它的边没有方向，并且图是连通的，即任意两个顶点之间都存在路径。
connected: path between any nodes: 连通意味着任意两个节点之间都有路径。
binary trees: 二叉树是树的一种，其中每个节点最多有两个子节点。
usually sparse: 树通常是稀疏的，意味着它们的边数相对较少。
how many edges in a tree of size k?: 大小为k的树含有k-1条边。
Forest (森林)

no cycles, not connected: 森林是由多个不相交的树组成的图，它们不包含循环，且各个树之间没有连接。

edges between all pairs of nodes: 完全图是图中的每对节点之间都有一条边相连的图。
how many edges in a clique of size k?: 大小为k的团（或完全子图）中有
k(k-1)/2
条边。

Graph types





Trees

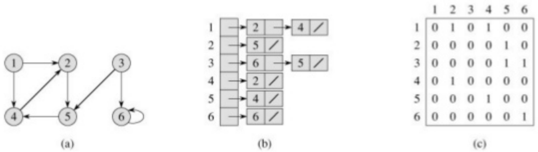
- acyclic, undirected, **connected** graphs
- connected: path between any nodes
- binary trees
- usually ***sparse***
- how many edges in a tree of size k?

Complete graphs & cliques

- edges between all pairs of nodes
- how many edges in a clique of size k?
- *e.g.* modules in protein–protein interaction networks

Introduction to graph theory

Graph representation



graph   Adjacency list   Adjacency matrix

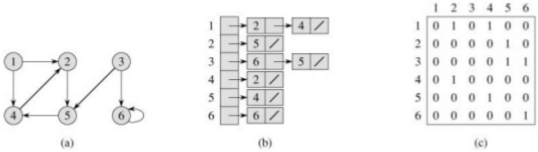
Adjacency matrix

- Rows and columns correspond to vertices
  - $A_{ij} = 1$ (***unweighted***)
  - $A_{ij} = A_{ji}$ (***undirected***)
  - $A_{ij} = w$ (***weighted***)
  - $A_{ij} = 0$

Adjacency list

- Node ***neighbourhood*** is represented by a list
- Good for sparse graphs

Adjacency matrix *vs.* list



graph | Adjacency list | Adjacency matrix

- Check if node *i* and *j* are connected
- Calculate node degree
  - is it different for a directed graph?

Storing graphs in a file:

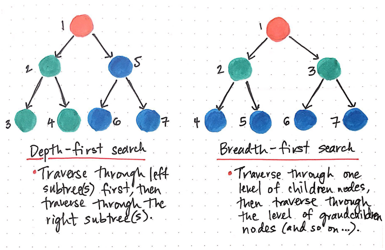
- represent the whole adjacency matrix
- edge list:

```
1,2
1,4
2,5
3,5
4,2
5,4
6,6
```

Introduction to graph theory

1. Concepts
2. Graph representation
3. Graph traversals
4. Depth-first search
5. Breadth-first search

Graph traversals



- process of visiting each vertex in a graph
  - **order vertices are visited**
- tree traversals will visit each node once
- two main algorithms

Depth-first search (DFS)

- ***Does a path between*** u, v ***exist?***
- visit child nodes before visiting the sibling/neighbour nodes

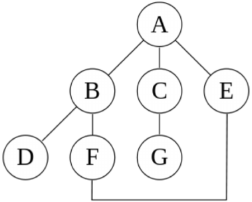
Breadth-first search (BFS)

- ***Find the shortest path between*** u, v
- visit sibling/neighbour nodes before visiting the children nodes

Introduction to graph theory

1. Concepts
2. Graph representation
3. Graph traversals
4. Depth-first search
5. Breadth-first search

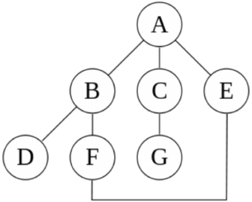
Depth-first search (DFS)



Procedure `DFS`(G, v)**:**
  Label v as discovered
  foreach *node* $w \in G.\text{adjacency}(v)$ **do**
    if *w is not discovered* **then**
      `DFS`(G, w)

- Nodes visited:
  - A, B, D, F, E, C, G
- Complexity is $O(|E|)$
  - $|E|$ is the number of edges of the graph G
  - uses a stack

Depth-first search (DFS)



Procedure `DFS-iterative`(G, v):
  Let S be a stack
  $S.\text{push}(v)$
  while *S is not empty* **do**
    $v \leftarrow S.\text{pop}()$
    if *v is not discovered* **then**
      label v as discovered
      foreach *node $w \in G.\text{adjacency}(v)$* **do**
        $S.\text{push}(w)$

- Nodes visited:
  - A, B, D, F, E, C, G
- Complexity is $O(|E|)$
  - $|E|$ is the number of edges of the graph G
  - uses a stack

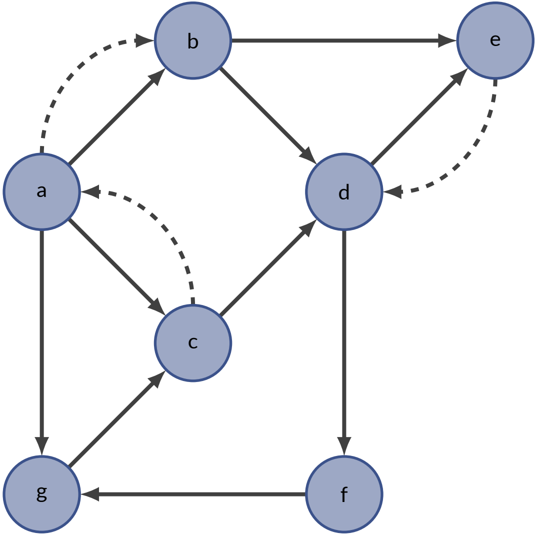
Example: Depth-first search

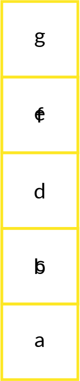
- The analogy for DFS algorithm is solving a maze:
  1. Work through the maze until reach a dead end.
  2. When we find a dead end, backtrack until you find an alternative route you haven't walked yet.
  3. Repeat until you find an exit (finding a path)

```
Procedure DFS(G, v):
    Label v as discovered
    foreach node w ∈ G.adjacency(v) do
        if w is not discovered then
            DFS(G, w)
```

Example: Recursive depth-first search

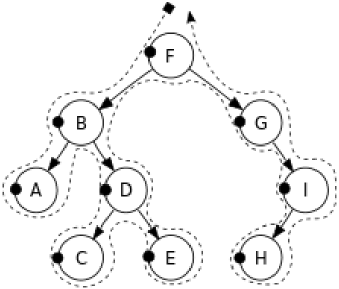
```
Procedure DFS(G, v):
    Label v as discovered
    foreach node w ∈ G. adjacency(v)
      do
        if w is not discovered then
            DFS(G, w)
```





See base$_{CS}$. Interactive examples on visualgo.net

DFS for binary trees



Procedure `Pre-order`(tree):
  Visit the current node
  Traverse the left subtree, `Pre-order`(left_subtree)
  Traverse the right subtree, `Pre-order`(right_subtree)

- F, B, A, D, C, E, G, I, H

Procedure `Post-order`(tree):
  Traverse the left subtree, `Post-order`(left_subtree)
  Traverse the right subtree, `Post-order`(right_subtree)
  Visit the current node

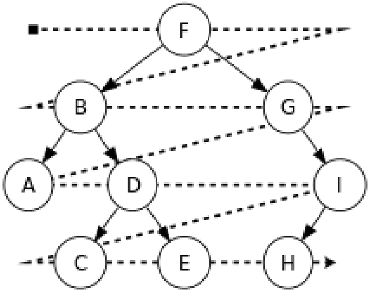
- A, C, E, D, B, H, I, G, F

Procedure `In-order`(tree):
  Traverse the left subtree, `In-order`(left_subtree)
  Visit the current node
  Traverse the right subtree, `In-order`(right_subtree)

- A, B, C, D, E, F, G, H, I

Introduction to graph theory

Breadth-first search (BFS)



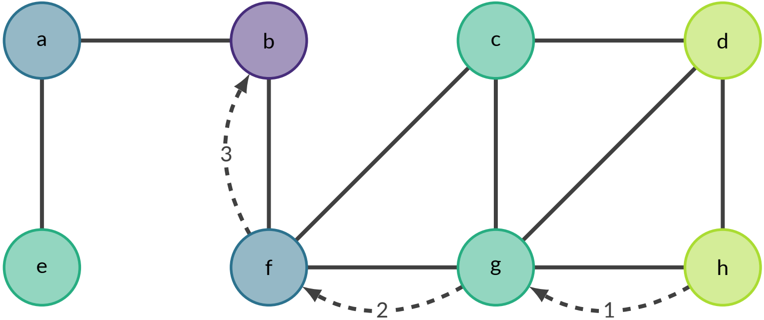
Procedure BFS(G, v):
  Let Q be a queue
  Q.enqueue(v)
  Label v as discovered
  while *Q is not empty* **do**
    $v \leftarrow Q$.dequeue()
    foreach *node $w \in G$*.adjacency(v) **do**
      if *w is not discovered* **then**
        Q.enqueue(w)
        w.parent $= v$
        Label w as discovered

- find the shortest path from one vertex to another
- produces a breadth first tree

- Nodes visited:
  - F, B, G, A, D, I, C, E, H
- Time complexity is $O(|E|)$
  - $|E|$ is the number of edges of the graph G
  - uses a queue

Example: Breadth-first search (BFS)

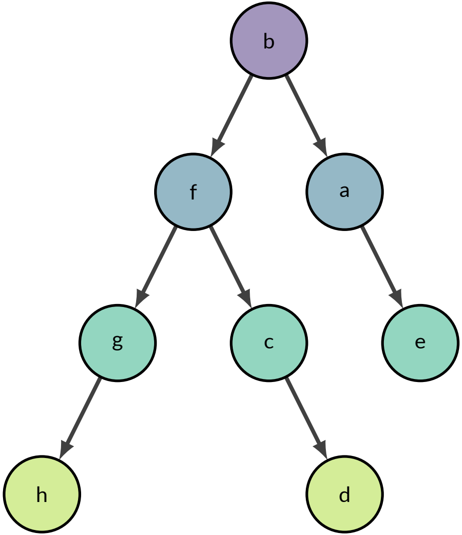
```
Procedure BFS(G, v):
    Let Q be a queue
    Q.enqueue(v)
    Label v as discovered
    while Q is not empty do
        v ← Q.dequeue()
        foreach node
          w ∈ G.adjacency(v) do
            if w is not discovered
              then
                Q.enqueue(w)
                w.parent = v
                Label w as
                  discovered
```



b	f	a	c	g	e	d	h

See base$_{cs}$. Interactive examples on visualgo.net

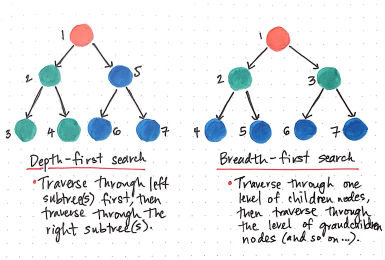
Example: Breadth-first search (BFS)



- ***Shortest paths*** for unweighted graphs
- Breadth-first tree

See base$_{CS}$. Interactive examples on visualgo.net

Graph traversals



- process of visiting each vertex in a graph
  - **order vertices are visited**
- tree traversals will visit each node once
- two main algorithms

Depth-first search (DFS)

- ***Does a path between*** u, v ***exist?***
- visit child nodes before visiting the sibling/neighbour nodes

Breadth-first search (BFS)

- ***Find the shortest path between*** u, v
- visit sibling/neighbour nodes before visiting the children nodes

Introduction to graph theory