COMTEK3,CCT3 & ESD3: ALGORITHMS

INTRODUCTION TO GRAPH ALGORITHMS, BFS & DFS

Ramoni Adeogun
Associate Professor and Head of the AI for Communication Group
Wireless Communication Networks Section (WCN)
Email: ra@es.aau.dk
October, 2023

AALBORG UNIVERSITET

Outline

- Introduction to Graphs
  - Representation of Graphs
  - Walks, Cycles, Paths and Loops
  - Degrees of vertices
  - Operations on Graphs

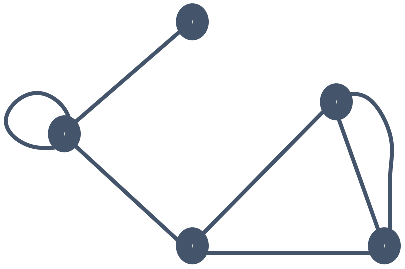
- BFS and DFS
  - Length of Paths and Distances on Graphs
  - Breadth-First-Search
  - Depth-First-Search
  - Topological Sort

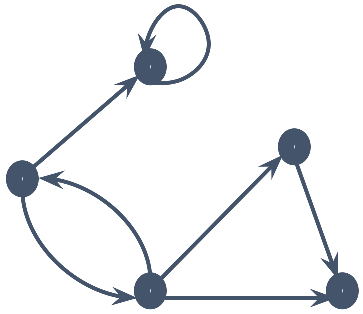
- Summary

I: INTRODUCTION TO GRAPHS

Introduction to Graphs

- A graph is a mathematical construct used to describe connections/relations between objects of all kinds
- Graphs can be:
  - undirected (graph): vertices are connected by edges without orientation
  - directed (digraph): vertices are connected by edges with orientation





Introduction to Graphs 2

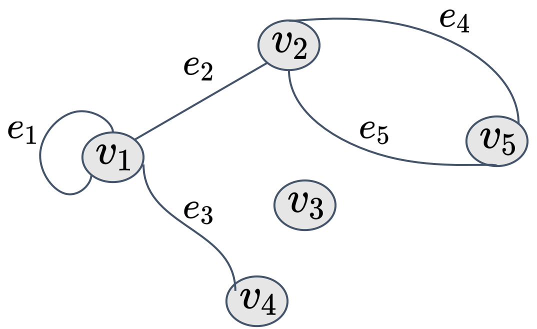
- Graphs can represent many different situations, e.g.,
  - Computer networks
  - Social networks
  - Cities and roads on a map
  - Systems of equations with many variables
  - Factorization of functions, e.g.:

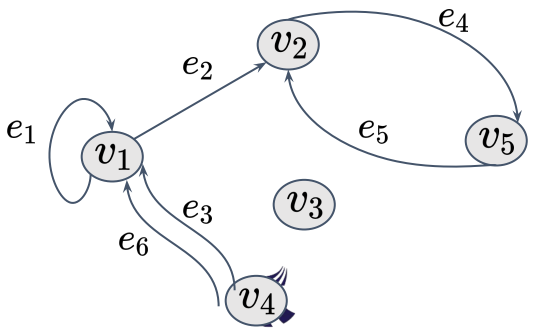
$$f(x_1, x_2, x_3, x_4) = f_a(x_1, x_2) f_b(x_1, x_3) f_c(x_2, x_3, x_4) f_d(x_4)$$

- **In the situations above, what would the edges and vertices represent?**
- Can you sketch a graph representing the factorization of function example?

Representation of Graphs

- Mathematically, definitions:
  - A graph $G = (V, E)$ is a structure consisting of a set of **vertices** $V = (v_1, v_2, \ldots)$ and a set of edges $E = (e_1, e_2, \ldots)$; each edge $e \in E$ has two endpoints which are vertices in V.
  - A digraph is a graph $G = (V, E)$ in which each edge have ordered endpoints: a start vertex and an end vertex





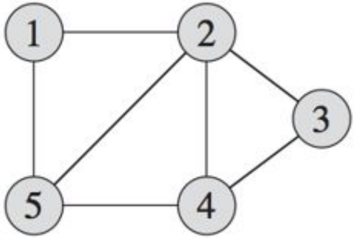
Can you identify the following kind of edges in each of the graphs?

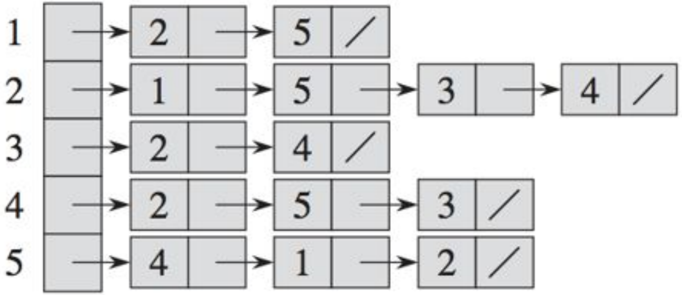
1. **Loop (self-loop)**
2. **Parallel edges**
3. **Antiparallel edges**

Representation of Graphs

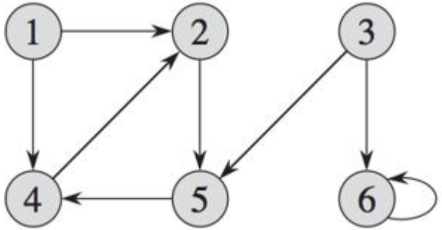
- Graphs can be represented in many ways:
  - Diagram
    - Human readable, but not good for computers
    - Best for small graphs
  - Adjacency list
    - Lookup requires linear search in a list $O(|V|)$
    - Memory efficient for sparse graphs $|E| << |V|^2$
  - Adjacency matrix
    - Easy lookup with $O(1)$ time
    - Large memory requirement $O\left(|V|^2\right)$
    - Best for dense graphs with $|E| \approx |V|^2$
    - Cannot represent parallel edges (but loops are ok)

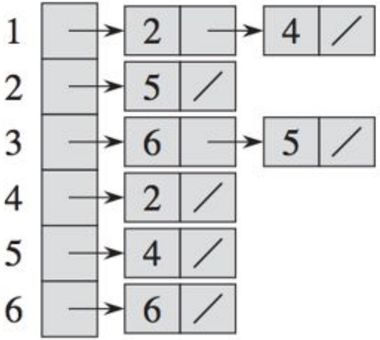
Representation of Graphs





$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

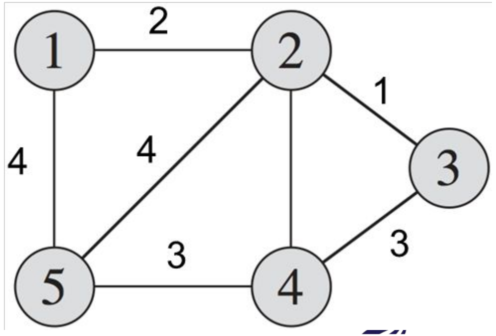




$$
\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
$$

Weighted Graphs

- Vertices have an associated numeric value (weight)
  - Can be used to represent: available/used link capacity, geographical distance, cost of decision, etc.
- Representation with adjacency-list and adjacency matrix is straight-forward:
  - Weight of edge between vertices u and v: w(u,v)



Simple Graphs

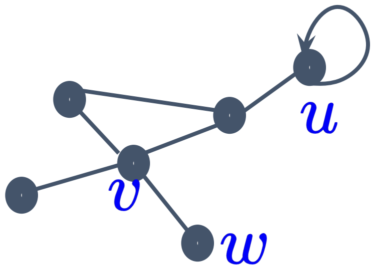
- A graph with no parallel edges is called a **simple graph.**
- Edges in a simple graph are taken to be pair of start and end vertices, i.e., $e = (v_{start}, v_{end})$
  - For simple graphs, we write $E \subset V^2$
- For undirected simple graphs, $(v_1, v_2) = (v_2, v_1)$

Walks, Cycles, Paths and Loops

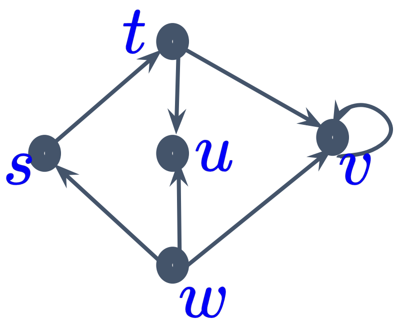
- A **walk** in a graph $G = (V, E)$ is an alternating sequence of vertices and edges:'
$$w = (u_1, e_1, u_2, e_2, \ldots, u_k, e_k, u_{k+1});\ e_k = (u_k, u_{k+1})$$
  - w is referred to as a $u_1 - u_{k+1}$ walk
- A walk is often written as a list of only the edges or vertices it traverses.
- A **path** is a walk in which no vertices are repeated
- A **walk** forms a cycle if $u_1 = u_{k+1}$ and $u_1, u_2, \ldots, u_k$ are all distinct
- A **loop** is a cycle containing only one edge

Degrees of Vertices

- The degree of a vertex v is the number of edges incident to v



$\deg(u) = 3$
$\deg(v) = 4$
$\deg(w) = 1$



vertex	deg	$\deg_i$	$\deg_o$
s	2	1	1
t	3	1	2
u	2	2	0
v	4	3	1
w	3	0	3

Handshaking Lemma:

For graph $G = (V, E)$, $\sum_{v \in V} \deg(v) = 2|E|$

Useful Graph Properties

- **Adjacency matrix** can be used to calculate
  - Degree
    - Summing the ith row gives the out degree of vertex i: $\deg_o(i)$
    - Summing the jth column gives the in degree of vertex j: $\deg_i(j)$
    - Sum of the ith column and ith row gives $\deg(i)$
  - Number of i - j walks
    - Theorem: Let A be the adjacency matrix of a simple graph $G = (V, E)$, then the (i,j)th entry of A^k, $k \geq 1$ is the number of different $v_i - v_j$ walks
      - Proof: we use mathematical induction

Operations on a Graph

- Many operations can be defined to transform a graph $G = (V, E)$ into another $G' = (V', E')$, e.g.,
  - Add edge e to G: $G' = G + e = (V,\ E\, u\, \{e\})$
  - Delete edge e from G: $G' = G - e = (V,\ E \setminus \{e\})$
  - Add vertex v to G: $G' = G + v = (V\, u\, \{v\}, E)$
  - Delete vertex v from G: $G' = G - v = (V \setminus \{v\}, E')$
  - Contraction of edge e: $G' = G \div e = (V', E')$
    - Step 1: $G - e$
    - Step 2: Replace v and u in the result

II: BFS and DFS

AALBORG UNIVERSITET

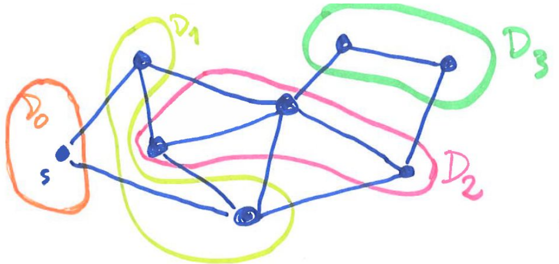
Length of Path, Distance & Shortest Path

- The **length of a path** is the number of edges in the path
  - A **path of length** k has k edges and at most k+1 vertices.
  - A **simple path** has exactly k+1 vertices

- The **distance** $\delta(s, v)$ from vertex s to v in a graph $G = (V, E)$ is the minimum length of a path from s to v in G.
  - Note: The shortest s-v path has length $\delta(s, v)$ and may not be unique
- A vertex v is reachable from s if there is a finite-length path from s to v.
  - If v is not reachable from s, we set $\delta(s, v) = \infty$

Distance Classes

- A distance class is the set of vertices with same distance from a particular vertex s.
  - the distance class $\mathcal{D}_i(s)$ is the set of vertices in a graph such that $\delta(s,v) = i \; \forall v \in \mathcal{D}_i(s)$
- A graph can be divided according to distances:



- A vertex $v \in \mathcal{D}_i(s)$ can be reached in i steps from vertex s
  - There is a length i path from s to v

Searching in Graphs

- The two fundamental search approaches are:

  - Breadth-first
    - Explores all nodes in 1-step distance, then 2-step distance, …

  - Depth-first
    - Explores a deep path when possible

- Other graph algorithms can be seen as extensions/hybrids of these

Breadth-First-Search (BFS)

- Given a graph or digraph, BFS finds
  - all vertices reachable from a "source" vertex s.
  - all distances from s to all reachable vertices
  - a "breadth- first tree" with root, s that contains all reachable vertices
- BFS is called so because
  - <u>it searches a graph breadth- wise</u>,discovering all vertices of distance k from s before moving on to those at distance k+1.
  - Thus it finds all vertices in distance class $\mathcal{D}_k(s)$ and then moves on to distance class $\mathcal{D}_{k+1}(s)$

Breadth-First-Search (BFS) 2

- BFS tracks its progress by coloring and recoloring vertices as the algorithm moves forward:
  - all vertices start as **white**
  - when a vertex is discovered it is colored **gray**
  - when done visiting a vertex it is colored **black**
- BFS uses on a FIFO queue with operations Enqueue & Dequeue

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each vertex v in G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

Breadth-First-Tree

For a connected graph $G = (V, E)$ with source $s \in V$, a predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is defined as:

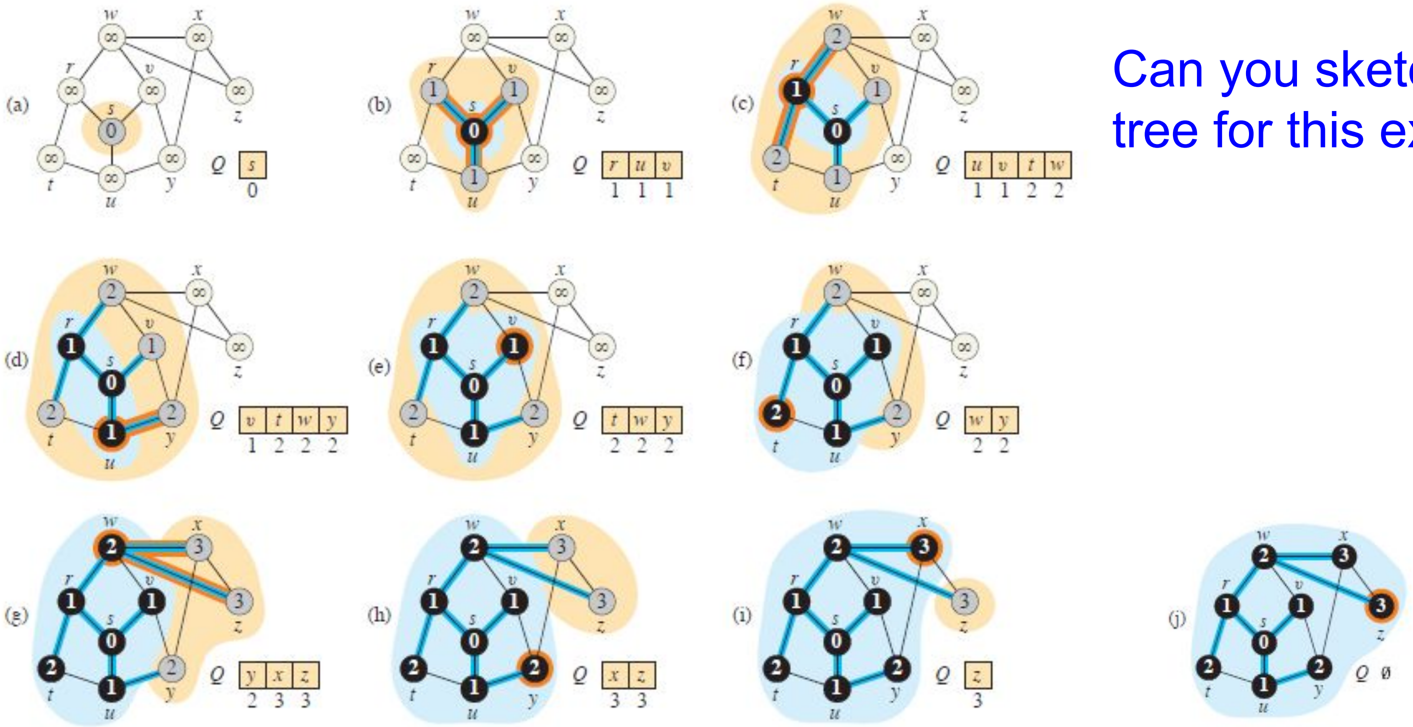
$$V_\pi = \{v \in V : v.\pi \neq NIL\} U \{s\}$$

$$E_\pi = \{(v\pi, v)\, v \neq V_\pi \backslash \{s\}\}$$

- A predecessor subgraph is a **breadth-first-tree** if
  - V_π consists of vertices reachable from s, and
  - for all $v \in V_\pi$, G_π contains a unique simple path from s to v, which is also a shortest path from s to v in G
- <u>G_π **is a tree**</u> since it is connected and $|E_\pi| = |V_\pi| - 1$

Note: BFS returns a breadth first tree

BFS - Example



Can you sketch the BFS tree for this example?

Depth-First-Search (DFS)

- DFS
  - explores a graph by searching “deeper” into it whenever possible
  - explores edges out of the most recently discovered vertex v which still has unexplored edges leaving it
    - when all v’s edges have been explored, DFS “backtracks” to explore edges leaving the vertex from which v was discovered
  - continues until all reachable vertices are discovered.
    - If any undiscovered vertices remain
      - DFS selects one and repeats with that as source, until all vertices have been discovered

Note: Since DFS is run from (possibly) multiple sources, its predecessor graph may consist of more than one tree => forms a **depth-first-forest**.

Depth-First-Search (DFS)

DFS gives the following attributes to vertices:

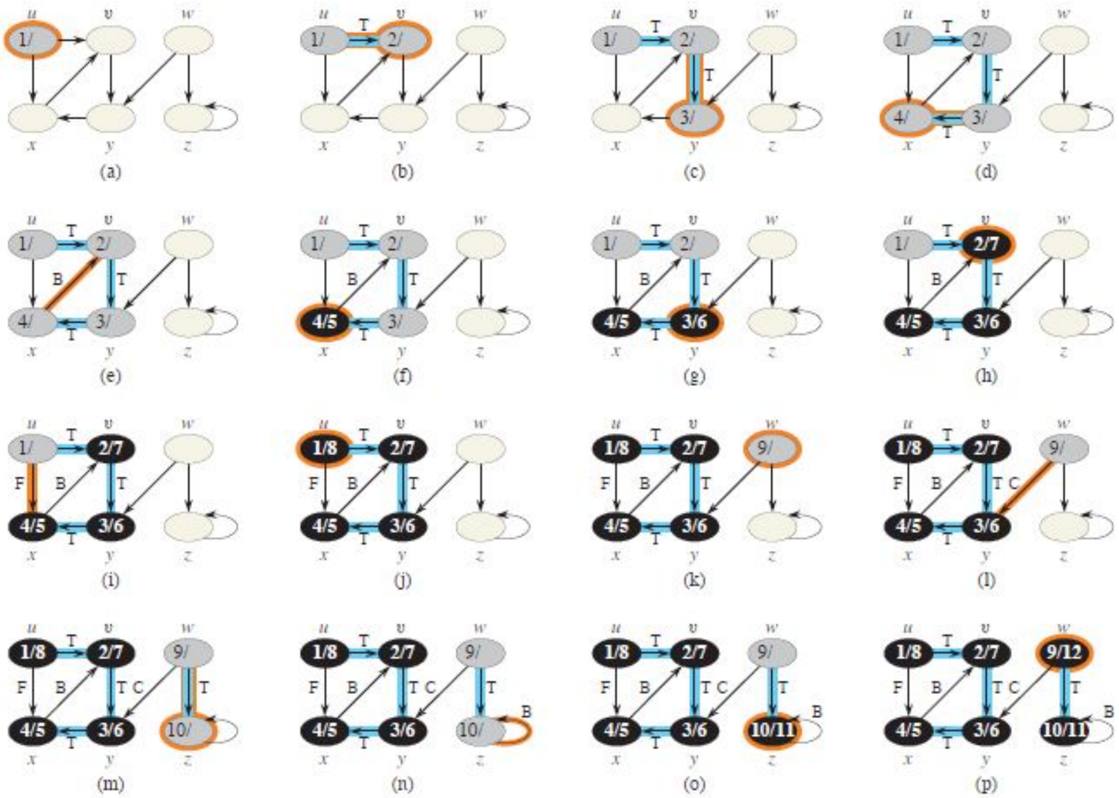
- v.color:
  - All vertices starts as **white**
  - when a vertex is discovered, it is **grayed**
  - When a vertex is finished, it is **blacked**
- v.d: discovery time-stamp
- v.f: finishing time-stamp

  A time-stamp is an integer between 1 and 2|V|
- $v.\pi$: predecessor of v

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1                 // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each vertex v in G.Adj[u]   // explore each edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
time = time + 1
u.f = time
u.color = BLACK                 // blacken u; it is finished
```

DFS - Example



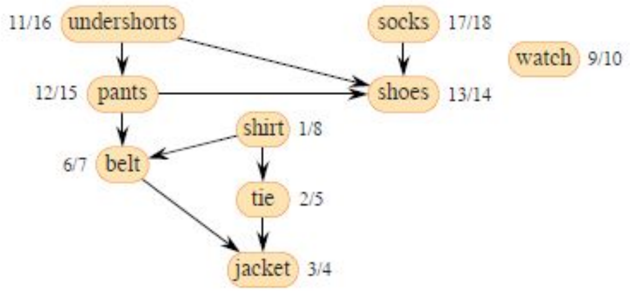
Topological Sort

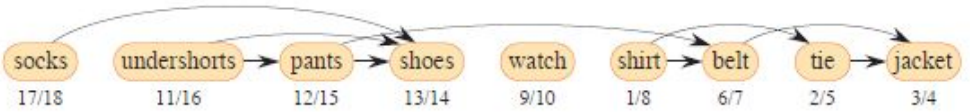
- DFS can be used to linearly order vertices of a directed acyclic graph (DAG), such that if $(u, v) \in E$,then *u* comes before *v*.
  - Such an ordering is called topological sort and can be obtained used a simple algorithm

TOPOLOGICAL-SORT(G)

call DFS(G) to compute finish times $v.f$ for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices

- Example: Bumstead Getting Dressed





Summary

- Graphs are useful in many applications
  - can be represented using
    - diagrams, adjacency lists, adjacency matrices, etc
  - different operations can be defined from graph transformation

- BFS searches all vertices that are reachable from a specified source
  - its output is the so called **Breadth-First-Tree**
- DFS combines deep exploration with backtracking
  - it result in a **Depth-First-Forest**