Design and Analysis of Algorithms

Lecture 05

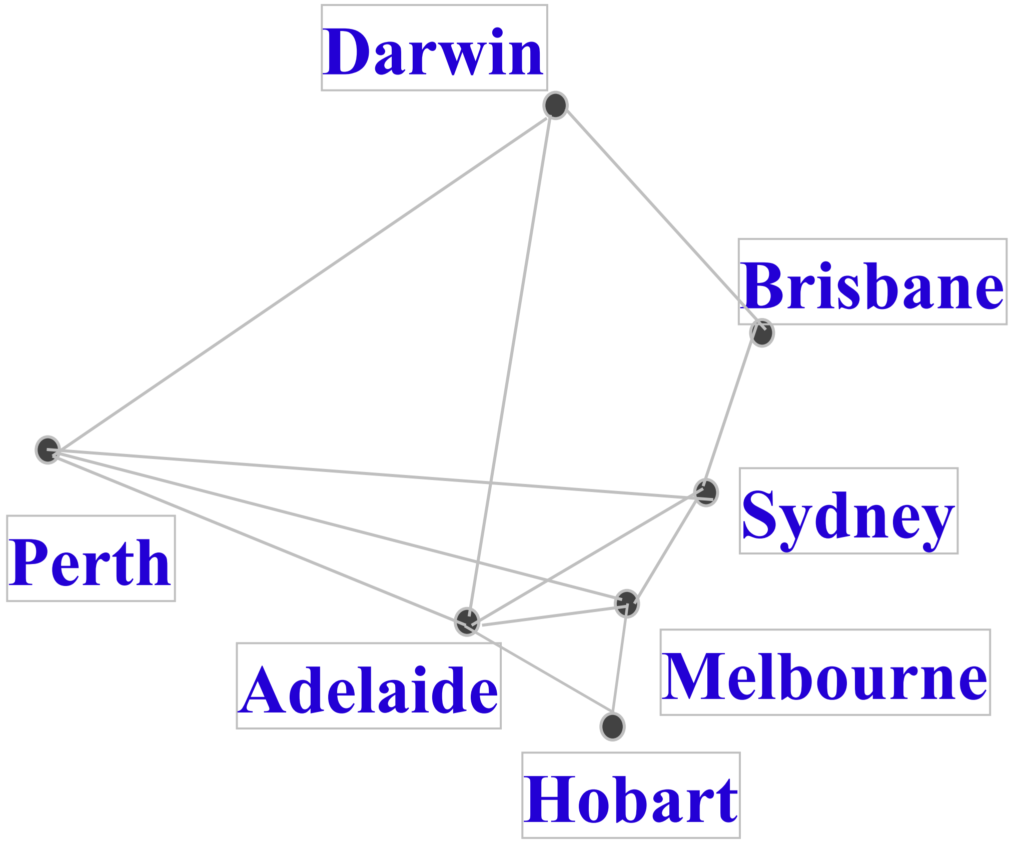
Graphs

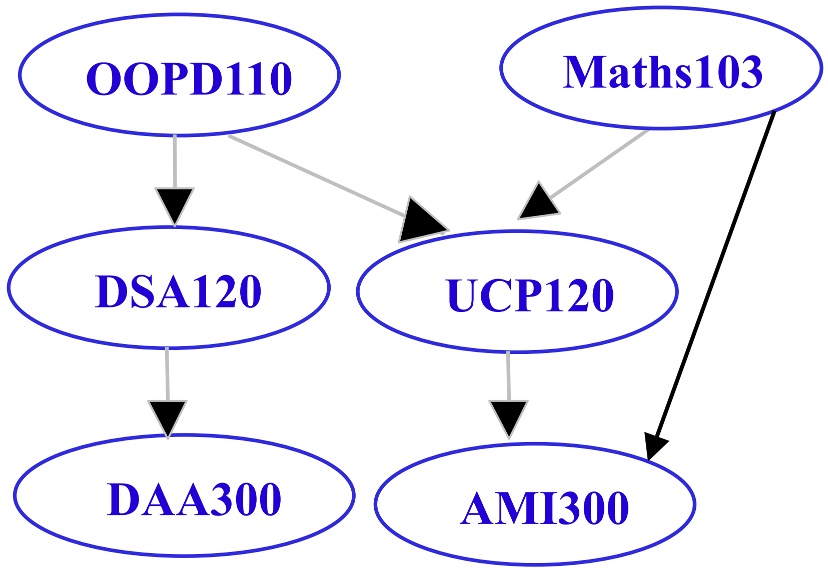
Topics

- Graph terminology
- Breadth-first-search
- Depth-first-search
- Connected Components
- Analysis of BFS and DFS Algorithms

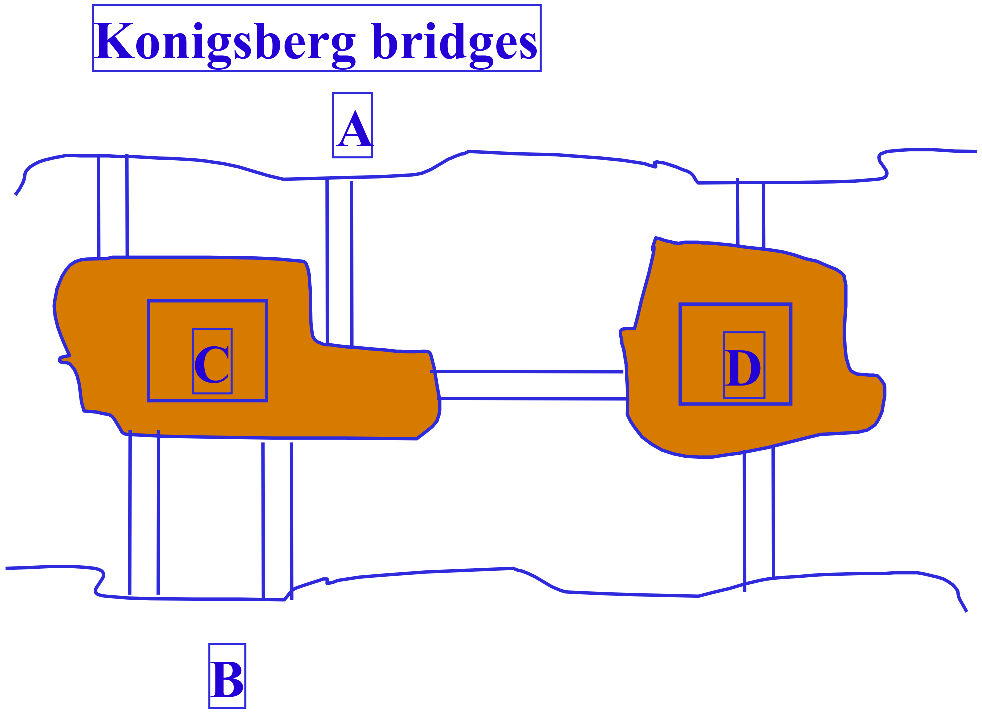
Curtin University of Technology

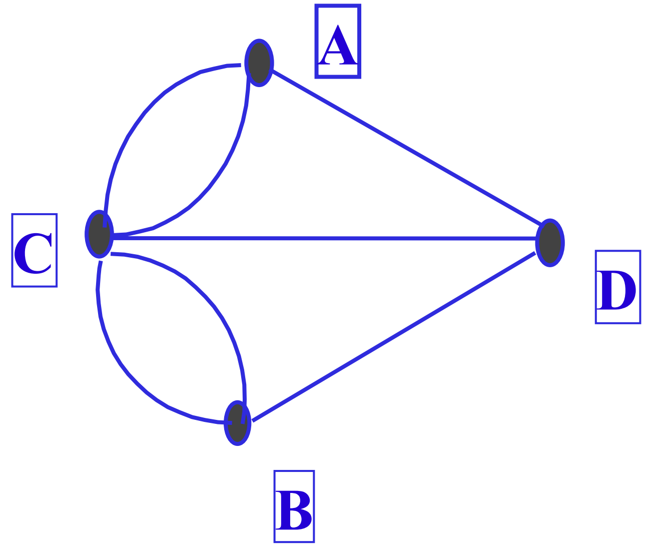
What is a graph?





Konigsberg Bridges





The town of Konigsberg (now Kaliningrad) lay on the banks and on two islands of the Pregel river. The city was connected by 7 bridges.

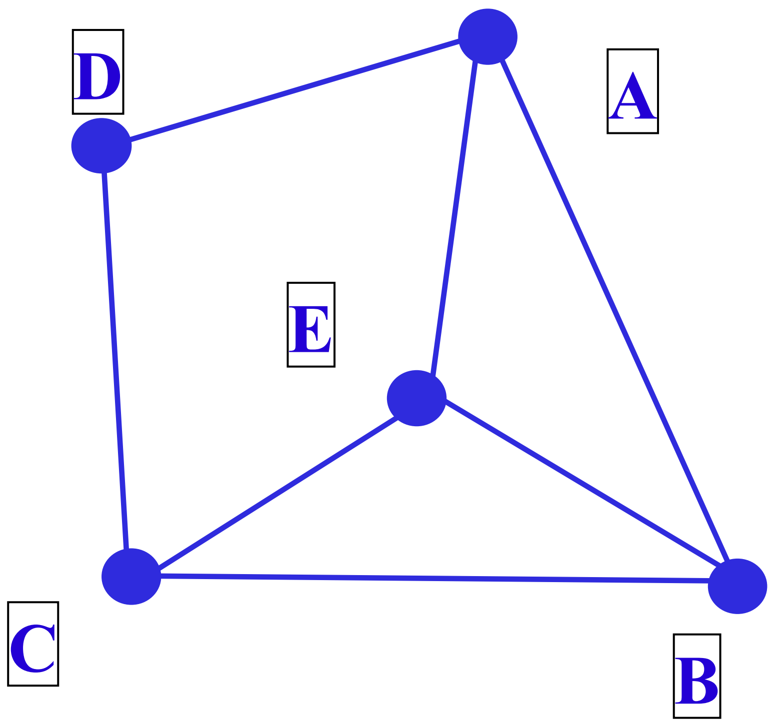
The puzzle (as encountered by Leonhard Euler in 1736) :
Whether it was possible to start walking from anywhere in town and return to the starting point by crossing all bridges exactly once.

Graph Terminology

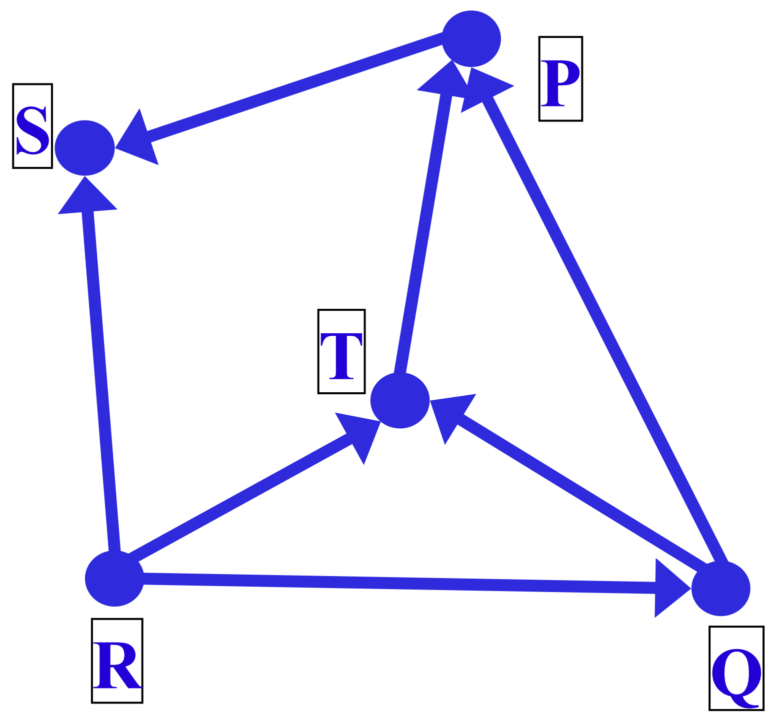
- A Graph consists of a set V of vertices (or nodes) and a set E of edges (or links)
- A graph can be directed or undirected
- Edges in a directed graph are ordered pairs
  - The order between the two vertices is important.
    Example: (S, P) is an ordered pair because the edge starts at S and terminates at P
  - The edge is **unidirectional**
- Edges of an undirected graph form unordered pairs, written as $\{S, P\}$
- A multigraph is a graph with possibly several edges between the same pair of vertices
- Graphs that are not multigraphs are called simple graphs

Curtin
University of Technology

Graph Terminologies (cont.)



G1 :Undirected Graph



G2: Directed Graph

Curtin University of Technology

Graph Terminologies (cont.)

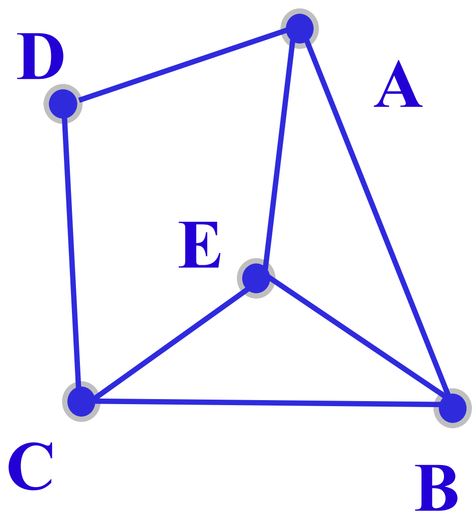
- The **degree** ***d*(*v*)** of a vertex v is the number of edges incident to v

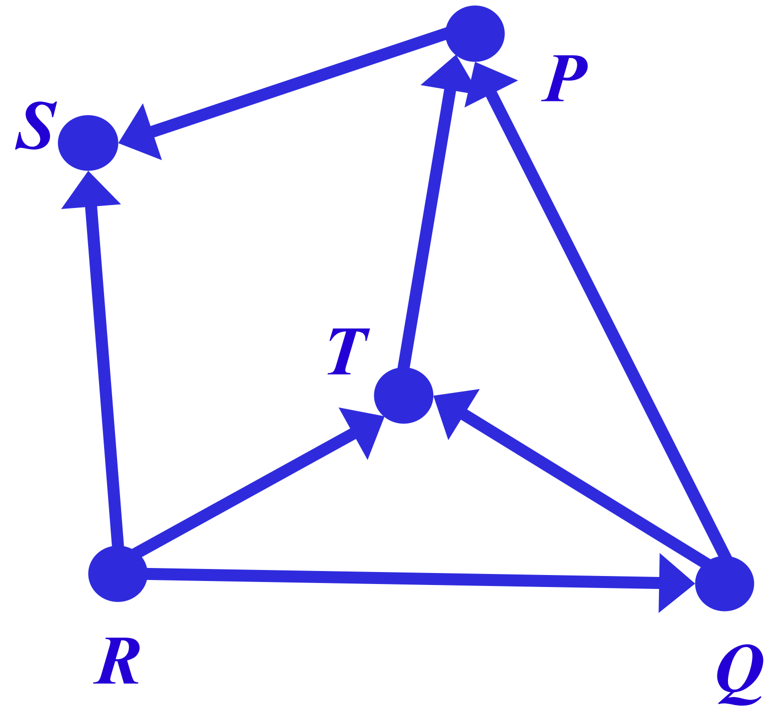
  $d(A) = 3$, $d(D) = 2$

- In directed graphs, **indegree** is the number of **incoming** edges at the vertex and **outdegree** is the number of **outgoing** edges from the vertex.

  Node P: indegree = 2, outdegree = 1.

  Node Q: indegree = 1, outdegree = 2.





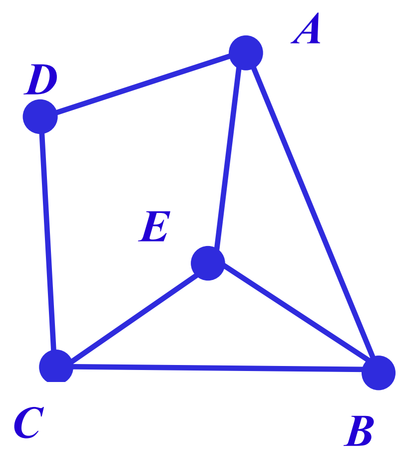
Curtin University of Technology

Paths and Cycles

- A **path** from vertex v_1 to v_k is a sequence of vertices $v_1, v_2, \ldots, v_k$ that are connected by edges $(v_1,v_2), (v_2,v_3), \ldots, (v_{k-1},v_k)$.

  Path from D to E: (D,A,B,E)

  Edges in the path: $(D,A), (A,B), (B,E)$

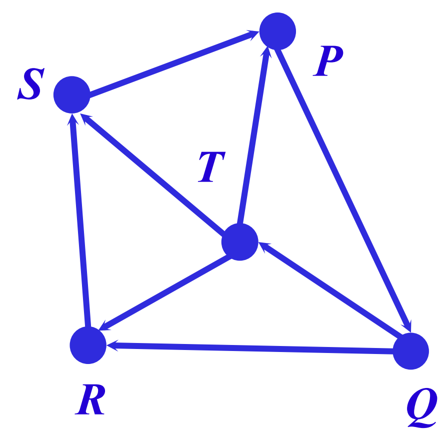


- A path is **simple** if each vertex in it appears only once.

  D A B E is a simple path

  A B C D A E is not a simple path

- Vertex u is said to be **reachable** from v if there is a path from v to u



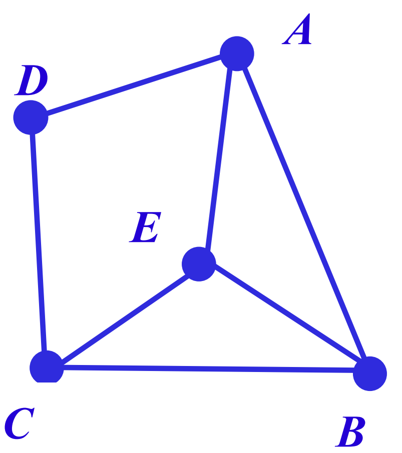
- A **circuit** is a path whose first and last vertices are the same;

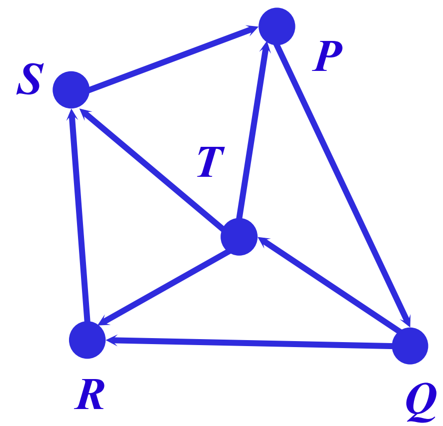
  DAEBCEAD, *ABEA*, *DABECD*, *SPQRS*, *SPQTRS* are circuits

Paths and Cycles

- A simple circuit is a **cycle** if except for the first (and last) vertex, no other vertex appears more than once.
  ABEA, *DABECD*, *SPQRS*, and *STRS* are cycles.

- A **Hamiltonian cycle** of a graph *G* is a cycle that contains all the vertices of *G*
  DABECD is a Hamiltonian cycle of *G*1
  PQRSTP is a Hamiltonian of *G*2





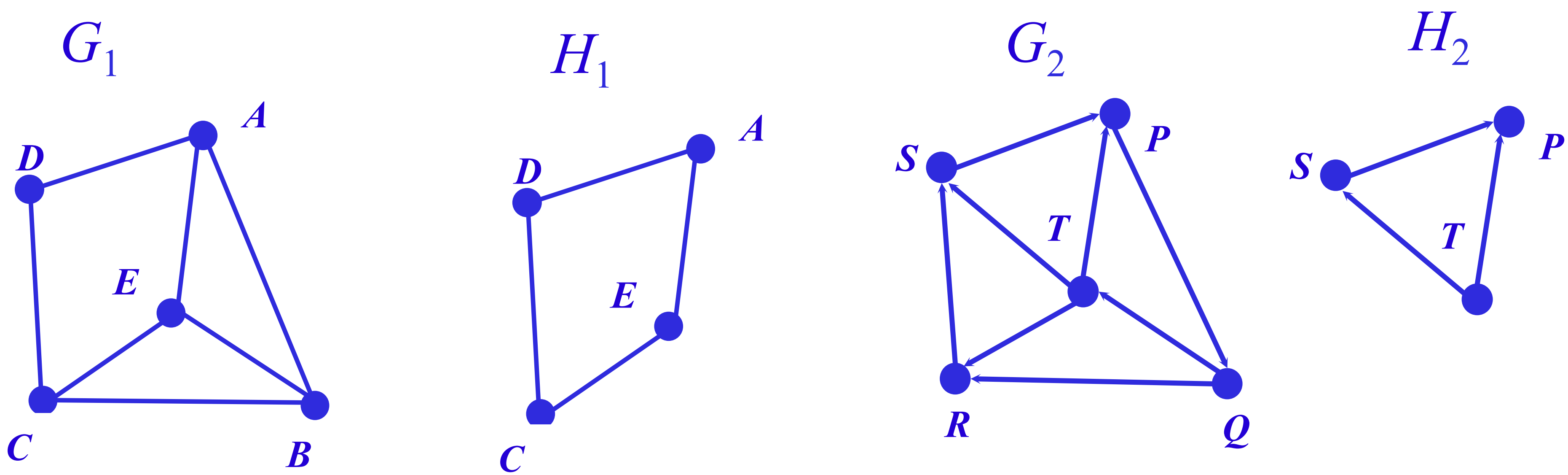
Curtin
University of Technology

Subgraph

- A **subgraph** of a graph $G=(V,E)$ is a graph $H=(U,F)$ such that $U \subseteq V$ and $F \subseteq E$.

$H_1=\{[U_1:A,E,C,D], [F_1: (A,E),(E,C),(C,\mathrm{D}),(\mathrm{D},\mathrm{A})]\}$ is subgraph of G_1

$H_2=\{[U_2:S,P,T],[F_2:(S,P),(S,T),(T,P)]\}$ is a subgraph of G_2



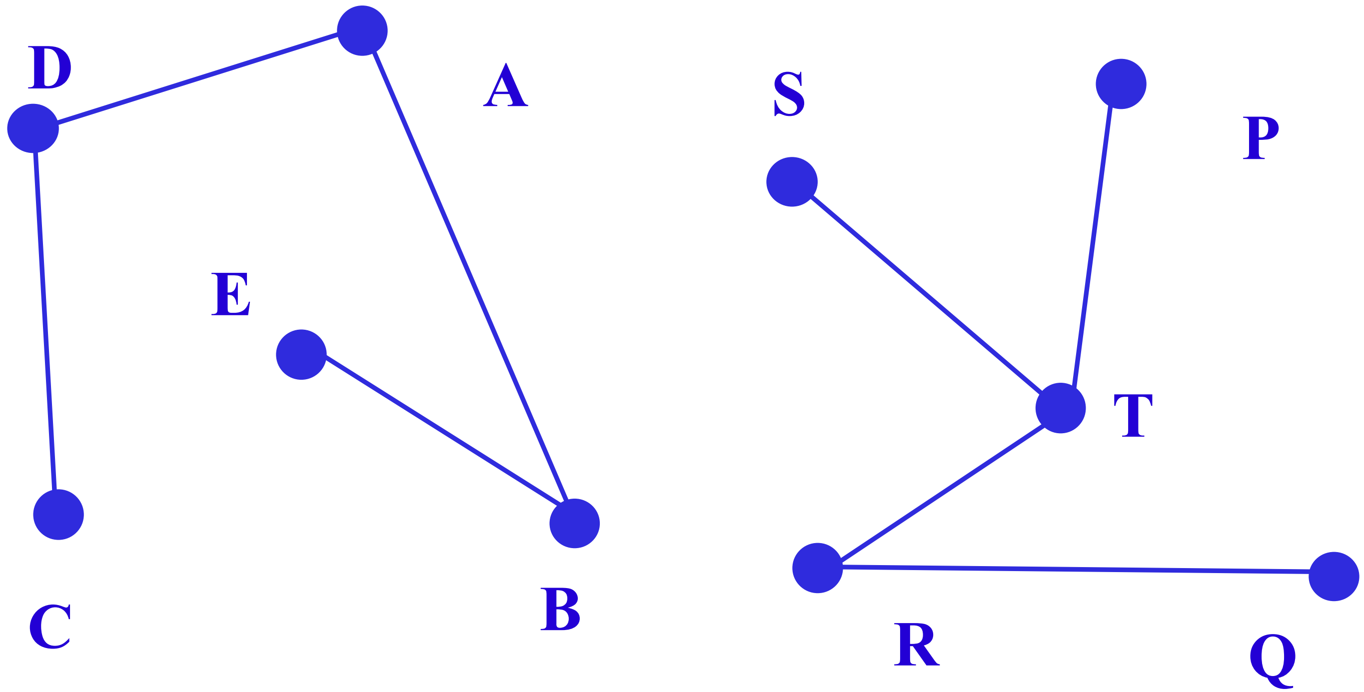
Curtin University of Technology

Graph Connectivity

- A graph is said to be **connected** if there is a path from any vertex to any other vertex in the graph → $G1$ and $G2$ are both connected graphs.
- A **forest** is a graph that does not contain a cycle.
- A **tree** is a connected forest.
- A **spanning forest** of an undirected graph G is a subgraph of G that is a forest and contains all the vertices of G.
- If a graph $G(V,E)$ is not connected, then it can be partitioned in a unique way into a set of connected subgraphs called **connected components**.
- A **connected component** of G is a connected subgraph of G such that no other connected subgraph of G contains it.

Curtin University of Technology

Forest

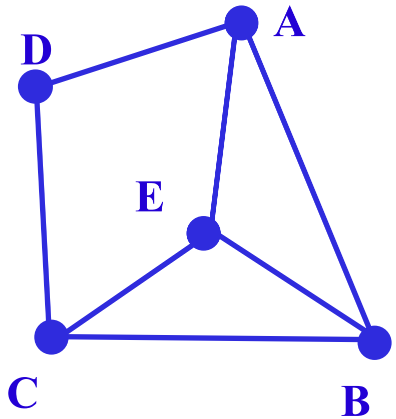


- *G* (A,B,C,D,E,P,Q,R,S,T) is a **forest**
- *G* (A,B,C,D,E) is a **tree**
- (A,B,C,D,E) and (P,Q,R,S,T) are **connected components**

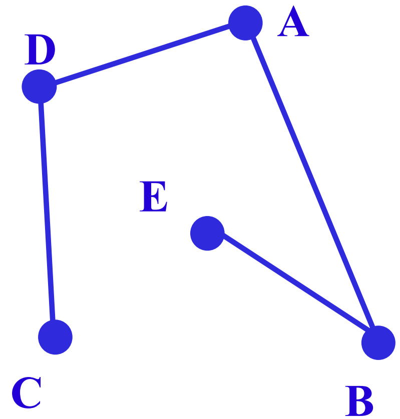
Curtin University of Technology

Spanning Tree

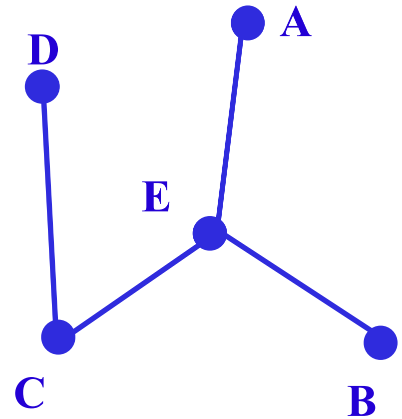
- A **spanning tree** of a graph *G* is a subgraph of *G* that is a tree and contains all the vertices of G.



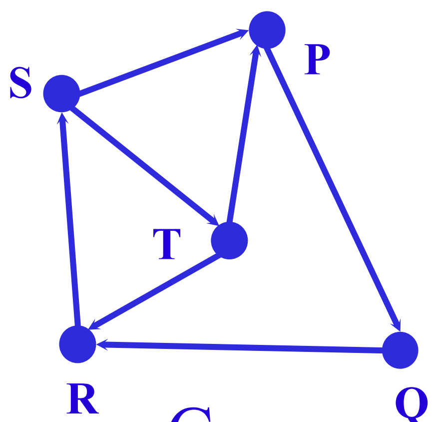
G_1



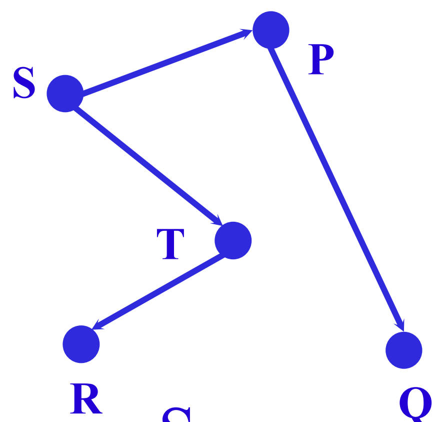
$S_{1,1}$



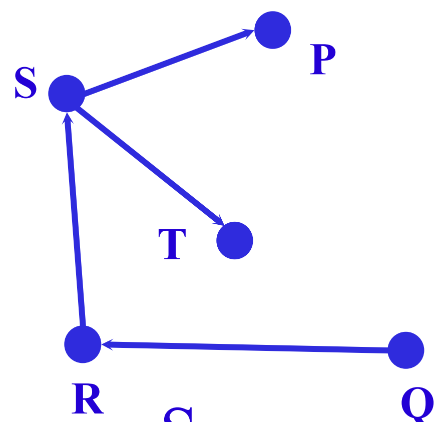
$S_{1,2}$



G_2



$S_{2,1}$



$S_{2,2}$

Curtin University of Technology

Graph Representations

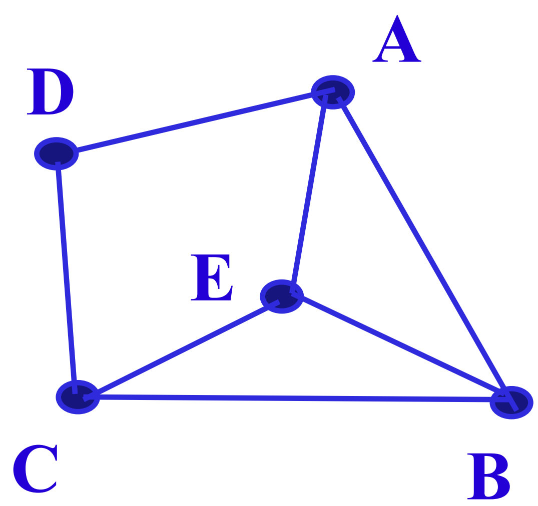
G1: undirected graph

Adjacency Matrix

	A	B	C	D	E
A	0	1	0	1	1
B	1	0	1	0	1
C	0	1	0	1	1
D	1	0	1	0	0
E	1	1	1	0	0

Adjacency list

A	B	D	E
B	A	C	E
C	B	D	E
D	A	C	\
E	A	B	C

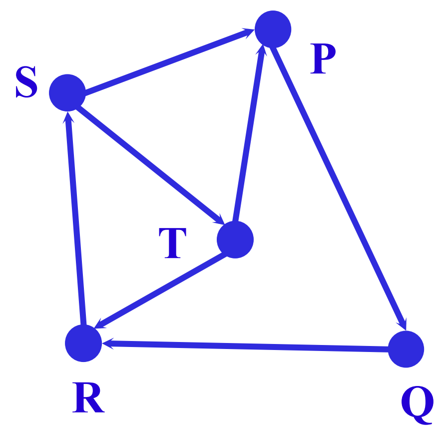


Directed Representation

G2: DirectedGraph

Adjacency matrix

	P	Q	R	S	T
P	0	1	0	0	0
Q	0	0	1	0	0
R	0	0	0	1	0
S	1	0	0	0	1
T	1	0	1	0	0



Adjacency list

P	Q	/
Q	R	/
R	S	/
S	P	T
T	P	R

Curtin University of Technology

Depth First Search

DFS_Tree *G*(*V*,*E*)

Input: $G = (V,E)$ in adjacency list format

x = value on top of stack S

$L[x]$ refers to the adjacency list of x

Output: The DFS tree T

1. Mark all vertices *new* and set $T = \{\ \}$ .... $\Theta(V)$
2. Mark any one vertex $v = old$ .... $\Theta(1)$
3. push (S, v) .... $\Theta(1)$
4. **while** S is nonempty **do** .... $\Theta(1)$
5. **while** exists a *new* vertex w in $L[x]$ **do** .... $\Theta(1)$
6. $T = T \cup (x,w)$ .... $\Theta(1)$
7. $w = old$ .... $\Theta(1)$
8. push w onto S .... $\Theta(1)$
9. pop S .... $\Theta(1)$

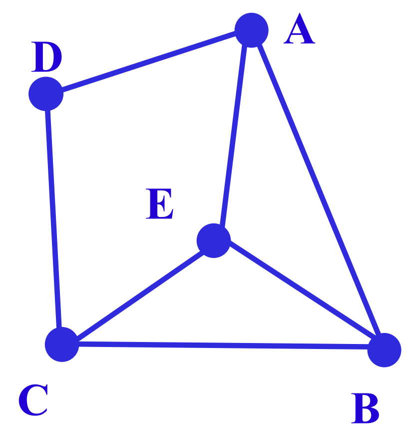
V **times** (lines 5–8)

V **times** (lines 4–9)

$$O(V + 1 + V^2) = O(V^2)$$

Curtin University of Technology

Depth First Search



DFS_Tree G(V,E)

Input: G = (V,E) in adjacency list format.

x = value on top of stack

L[x] refers to the adjacency list of x

Output : The DFS tree T

1. Mark all vertices *new* and set $T = \{\ \}$ .... $\Theta(V)$
2. Mark any one vertex $v = old$ .... $\Theta(1)$
3. push (S, v) .... $\Theta(1)$
4. **while** S is nonempty **do** .... $\Theta(V)$
5. **while** exists a *new* vertex w in $L[x]$ **do** .... $\Theta(1)$
6. $T = T \cup (x,w)$ .... $\Theta(1)$
7. $w = old$ .... $\Theta(1)$
8. push w onto S .... $\Theta(1)$
9. pop S .... $\Theta(1)$

At most 2*E times over the whole algorithm

$$O(V + V + E) = O(V + E)$$

Curtin University of Technology

DFS - Example

1

new = {A,B,C,D,E}
old = {}
T = {}

2

A

new = {B,C,D,E}
old = {A}
L[A] = {B,D,E}
T = {}

3

new={C,D,E}

B
A

L[B] = {A,C,E}
old = {A,B}
T = {(A,B)}

4

new={D,E}

C
B
A

L[C] = {B,D,E}
old = {A,B,C}
T = {(A,B),(B,C)}

5

New ={E}

D
C
B
A

L[D] ={A,C}
old = {A,B,C,D}
T={(A,B),(B,C),(C,D)}

6

new ={E}

C
B
A

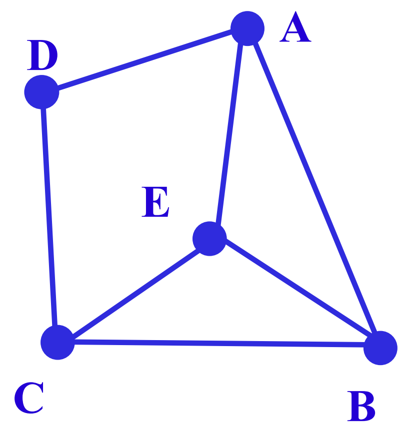
L[C] = {B,D,E}
old = {A,B,C,D}
T={(A,B),(B,C),(C,D), (C,E)}

7

new = {}

E
C
B
A

L[E]={A,B,C}
old = {A,B,C,D,E}
T = {(A,B),(B,C), (C,D),(C,E)}



Curtin University of Technology

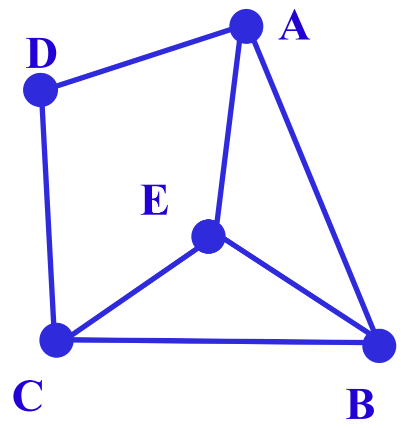
Breadth-first Search

BFS_Tree_G(*V*,*E*)
Input: $G = (V,E)$. $L[x]$ refers to the adjacency list of x.
Output: The BFS tree T;

1. Mark all vertices *new* and set $T = \{\ \}$ $\Theta(V)$
2. Mark the start vertex $v = old$ $\Theta(1)$
3. insert (Q,v) // Q is a queue
4. **while** Q is nonempty **do**
5. $x = dequeue(Q)$
6. **for** each vertex w in $L[x]$ marked *new* **do**
7. $T = T \cup \{x,w\}$
8. Mark $w = old$
9. insert (Q,w) $\Theta(1)$

V times

At most 2*E times over the whole algorithm



$$O(V+V+E) = O(V+E)$$

Curtin University of Technology

BFS - Example

1

Q: (empty)

new = {A,B,C,D,E}
old = {}
T = {}

2

Q: A

L[A]={B,D,E}
new = {B,C,D,E}
old = {A}
T = {}

3

Q: B | D | E

L[B] = {A, C, E}
new = {C}
old = {A,B,D,E}
T = {(A,B),(A,D),(A,E)}

4

Q: D | E | C

L[D]={A,C}
new = {}
old = {A,B,D,E,C}
T = {(A,B),(A,D),(A,E),(B,C)}

5

Q: E | C

L[E]={A,B,C}
new = {}
old = {A,B,D,E,C}
T = {(A,B),(A,D),(A,E),(B,C)}

6

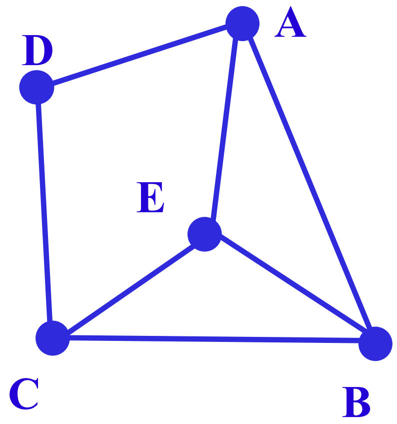
Q: C

L[C]={B,D,E}
new = {}
old = {A,B,D,E,C}
T = {(A,B),(A,D),(A,E),(B,C)}

7

Q: (empty)

new = {}
old = {A,B,D,E,C}
T = {(A,B),(A,D),(A,E),(B,C)}



Curtin University of Technology

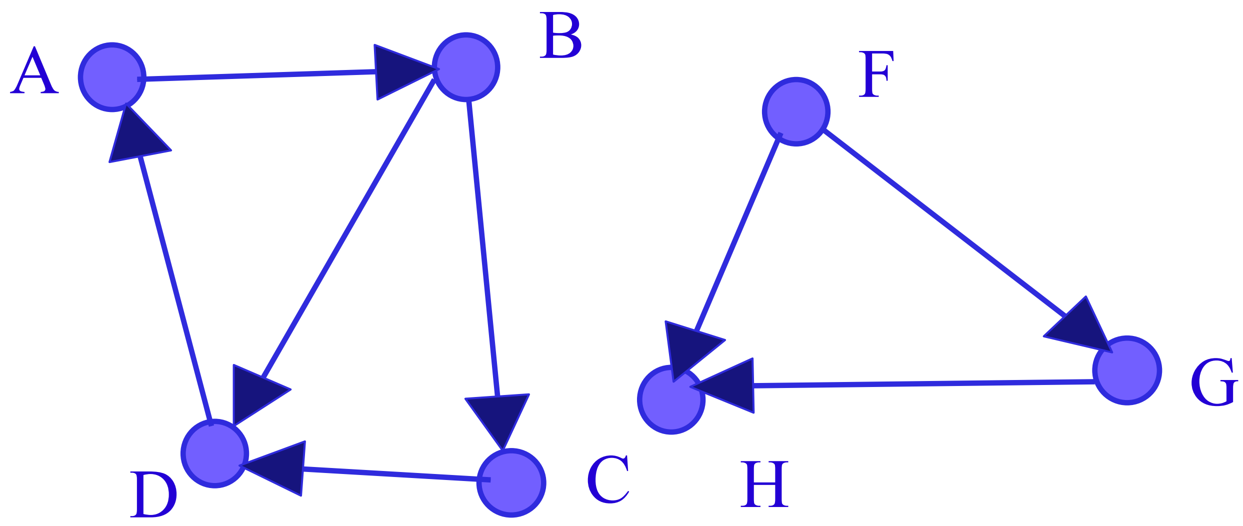
Connected Components

- The connected component of a graph $G = (V,E)$ is a maximal set of vertices $U \subseteq V$ such that for every pair of vertices u and v in U, we have both u and v reachable from each other. In the following we give an algorithm for finding the connected components of an undirected graph.

Connected_Components_G(V,E)
Input: G (V,E)
Output: Number of Connected Components and $G1$, $G2$ etc, the connected components

1. $V' = V$
2. $c = 0$
3. **while** $|V'| \neq 0$ **do**
4. choose $u \in V'$
5. T = all nodes reachable from u (use the DFS_Tree function)
6. $V' = V' - T$
7. $c = c+1$
8. $G_c = T$
9. $T = 0$;

Connected Components



- Suppose the DFS tree starts at A, we traverse from A → B → C → D and do not explore the vertices F, G, and H at all! The DFS_tree algorithm does not work with graphs having two or more connected parts.

- We have to modify the DFS_Tree algorithm to find a DFS forest of the given graph.

Curtin University of Technology

DFS Forest

DFSForest _G(V,E)

Input: G = (V,E); *S* is a stack - initially empty;

x refers to the top of stack; initially mark all vertices *new*;

L[*x*] refers to the adjacency list of *x*.

DFS Forest *F* = { };

Output: DFS tree *F*;

```
for each vertex v ∈ V do  //generate all components
    if v is new then   // generate one component
        v =old
        push (S,v)
        while S is nonempty do
            while there exists a vertex w in L[x] marked new
                F = F ∪ (x,w)
                w = old
                push (S,w)
            pop S
```

The End

Curtin
University of Technology