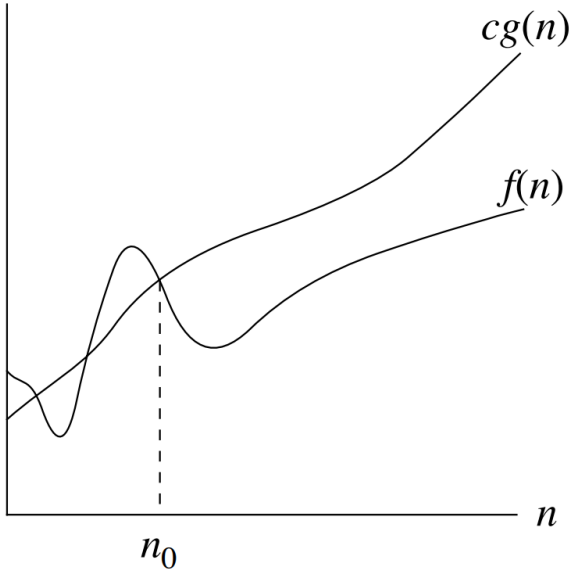
Algorithm Design & Analysis (CSE222)

Lecture-2

Recap

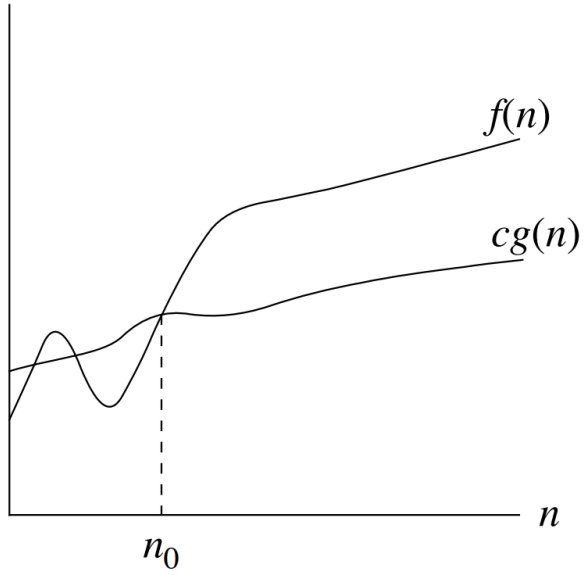
Characteristics of Algorithm: Well defined Input, Output, Finiteness, Definiteness, Effectiveness.

Growth Functions: *O*-notation



O(g(n)) = {f(n): there exists positive constants c and n_0 such that $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0$}.

Ω-notation



Ω(g(n)) = {f(n): there exists positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n)$ for all $n \ge n_0$}.

Outline

- Growth Functions (cont.)
- Graph Notations
- Revisit BFS

Growth of Function

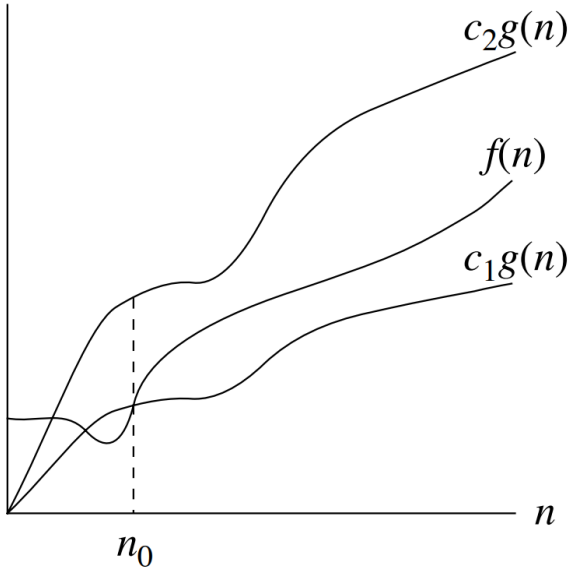
For an input of size n, let the running time of your algorithm is f(n). Then which of the following are true?

- $f(n) = n^2 + 10^{10} \cdot n$
  - $f(n) = O(n)$?
  - $f(n) = O(n^2)$?
- $f(n) = 10^{-10} \cdot n^2 + n$
  - $f(n) = \Omega(n)$?
  - $f(n) = \Omega(n^2)$?

Growth of Function

Θ-notation

$\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2 and n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0\}$.



g(n) is an asymptotic tight bound for f(n).
So, $f(n) = \Theta(g(n))$.

Growth of Function

Θ-notation

Can find a relation between the following functions using Θ-notation.

- $f(n) = n^{1/2}$ and $g(n) = n$
- $f(n) = n^{-1}$ and $g(n) = \log(n)$
- $f(n) = n^{1.000000001}$ and $g(n) = n$
- $f(n) = 3n + 4$ and $g(n) = 2n - 1$
- $f(n) = 10^{-10} \cdot n^2 + n$ and $g(n) = n^2 + 10^{10} \cdot n$

Growth of Function

o-notation

$o(g(n))$ = {f(n): there exists positive constants c and n_0 such that $0 \leq f(n) < c \cdot g(n)$ for all $n \geq n_0$}.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

What is the relationship between f(n) and g(n) for the following functions.

- $f(n) = n$ and $g(n) = n^{1/2}$
- $f(n) = n \cdot \log(n)$ and $g(n) = n^{3/2}$
- $f(n) = n$ and $g(n) = n^2 + n$
- $f(n) = n^2$ and $g(n) = n^2 + n$

Growth of Function

ω-notation

$\omega(g(n))$ = {f(n): there exists positive constants c and n_0 such that $0 \leq c \cdot g(n) < f(n)$ for all $n \geq n_0$}.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

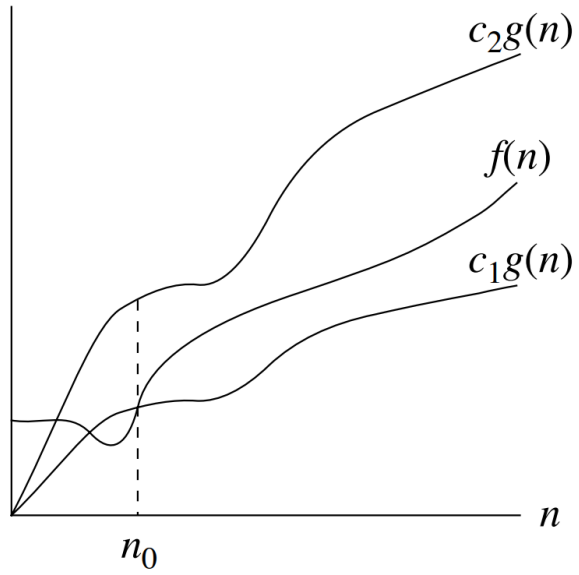
What is the relationship between f(n) and g(n) for the following functions.

- $f(n) = n$ and $g(n) = n^{1/2}$
- $f(n) = n$ and $g(n) = n^2 - n$
- $f(n) = n^2$ and $g(n) = n^2 + n^{-1}$

Growth of Function

Θ-notation

$\Theta(g(n))$ = {f(n): there exists positive constants c_1, c_2 and n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$}.



$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

Relational Properties

Transitivity
If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
This also holds for O, Ω, o and ω

Reflexivity
$f(n) = \Theta(f(n))$
Also holds for O and Ω

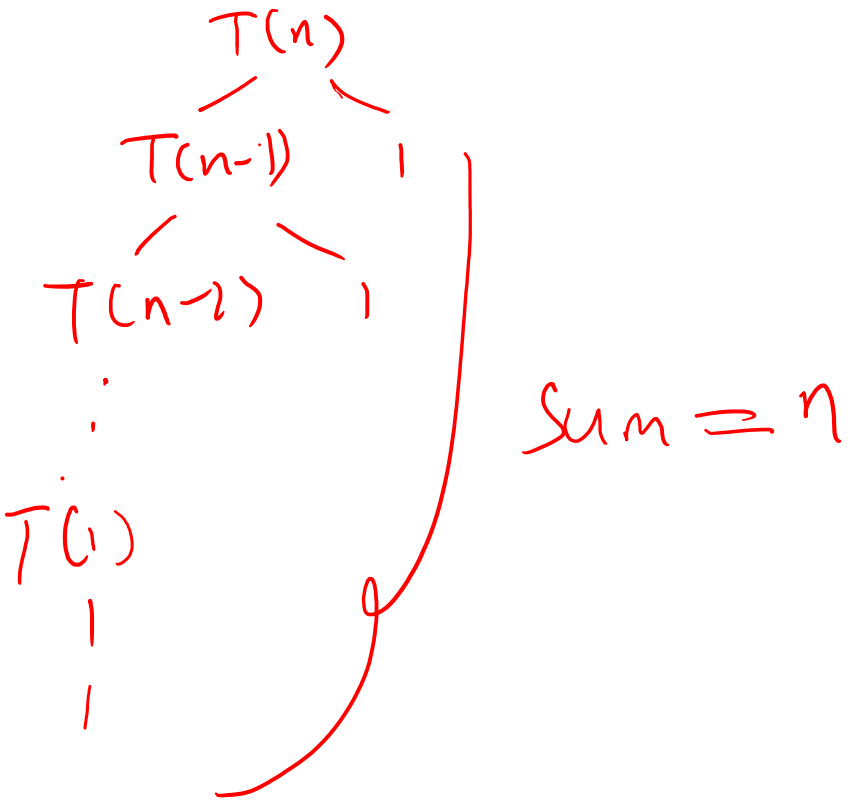
Symmetry
$f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

Transpose
$f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.
$f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$.

Recurrences

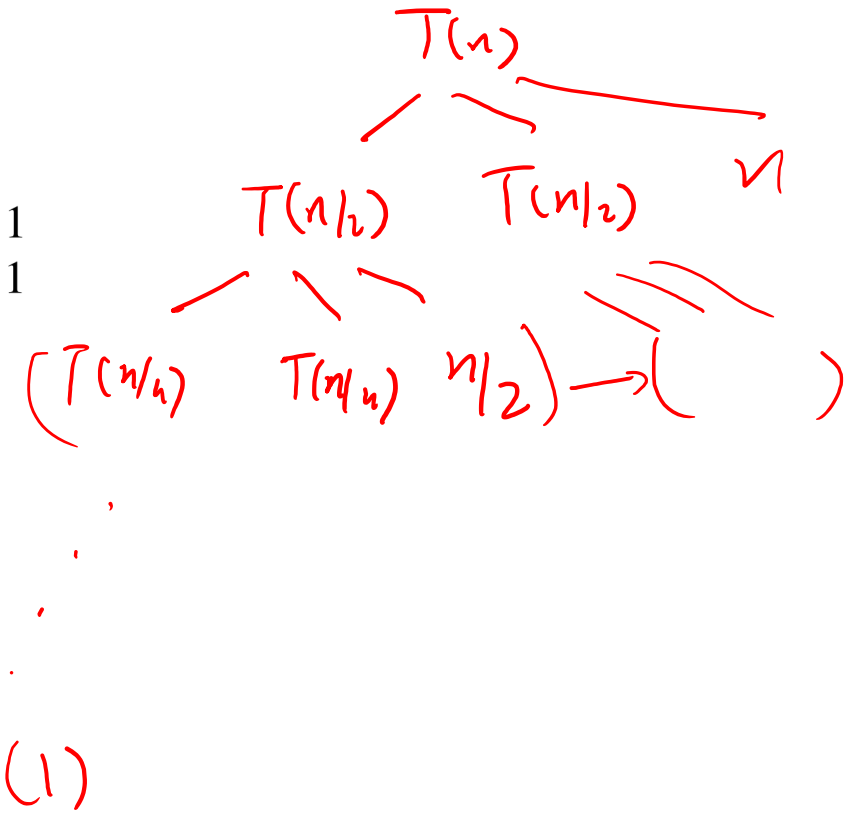
Recurrences are functions with one or more base cases and itself with smaller arguments.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$$



Recurrences

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geq 1 \end{cases}$$



$T(n) = n \log(n)$

Master's Theorem

$T(n) = aT(n/b) + f(n)$ be defined on a non-negative integer, where,
a ≥ 1 is a constant (# subproblems),
b > 1 is a constant (such that n/b is input size to each subproblem) and
f(n) is a polynomial function (cost on each recursive call).

T(n) has following asymptotic bound for a constant ε > 0.

- If $f(n) = O(n^{\log_b a - \varepsilon})$, then $T(n) = \theta(n^{\log_b a})$.
- If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} \log_e n)$.
- If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and $af(n/b) < cf(n)$ for some constant $c < 1$ and for all sufficiently large n then $T(n) = \theta(f(n))$.

Example

$T(n) = 9T(n/3) + n$

$T(n) = T(3n/5) + 1$

$T(n) = 2T(n/4) + n$

Outline

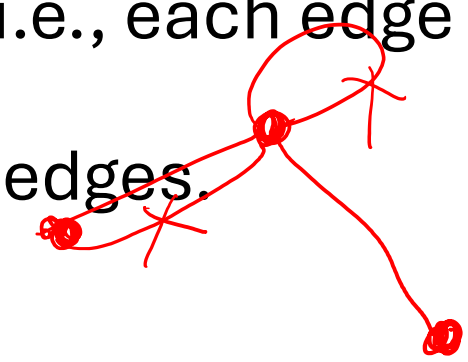
- Growth Functions (cont.)
- Graph Notations
- Revisit BFS

Graph Notations

Graphs: Captures relationship (a.k.a edges) between objects (a.k.a nodes).
Consists of collection of nodes (a.k.a vertices) V and a collection of edges E (sets of nodes).
$G = (V, E)$

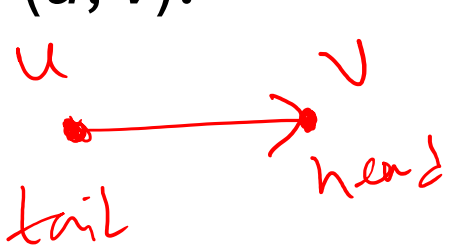
<u>Simple Graphs</u>

- At most one edge between pair of vertices (i.e., each edge is a pair of nodes).
- Undirected (or symmetric) and unweighted edges.
- No self loops.



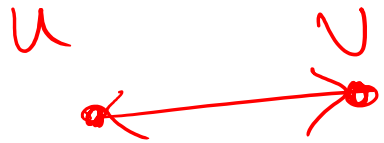
Graph Notations

Directed graph for asymmetric relationship, $G = (V, E)$. Every $e \in E$ is an ordered pair (u, v).



(u, v) ✓

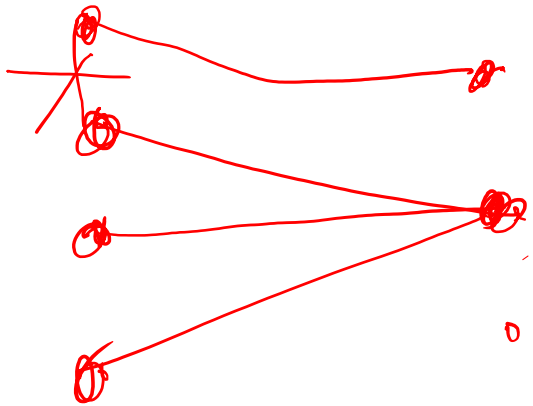
(v, u) ✗



(u, v) or (v, u)

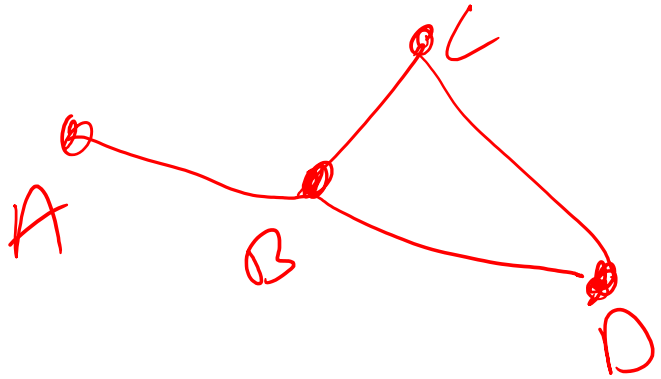
Graph Notations

Bipartite Graph: Vertex set consists of two disjoint sets of vertices, such that every edge is a pair of one vertex from each disjoint sets.



Graph Notations

In an undirected graph $G = (V, E)$, a **path** p is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each pair of consecutive vertices (v_i, v_{i+1}) is an element in E. That is they are connected by an edge.



A, B, D ✓

(A, B), (B, D)

A, B, C ✓

A C B X

Graph Notations

In an undirected graph $G = (V, E)$, a **path** p is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each pair of consecutive vertices (v_i, v_{i+1}) is an element in E. That is they are connected by an edge.

A **simple path** is a sequence of vertices, where none of them are repeated.

A **cycle** is a path if all the vertices are unique except for the first and the last.

A graph is called **connected** if there is a path between every pair of vertices.

Graph Notations

In an undirected graph $G = (V, E)$, a **path** p is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each pair of consecutive vertices (v_i, v_{i+1}) is an element in E. That is they are connected by an edge.

A **simple path** is a sequence of vertices, where none of them are repeated.

A **cycle** is a path if all the vertices are unique except for the first and the last.

A graph is called **connected** if there is a path between every pair of vertices.

A **tree** is a connected graph without any cycle.

Outline

- Growth Functions (cont.)
- Graph Notations
- Revisit BFS

Revisit BFS

Breadth First Search (BFS): Starting from a vertex it traverses the graph layer by layer.

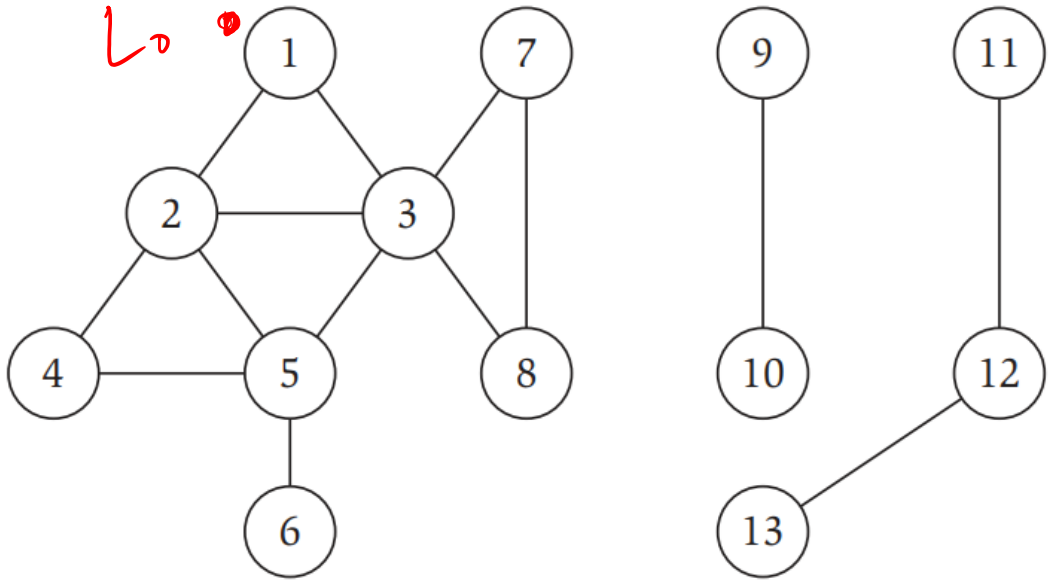
$L_0 = 1,$

$L_1 = 2, 3$

$L_2 = 4, 5, 8, 7$

$L_3 = 6$

L_4



Revisit BFS

Applications

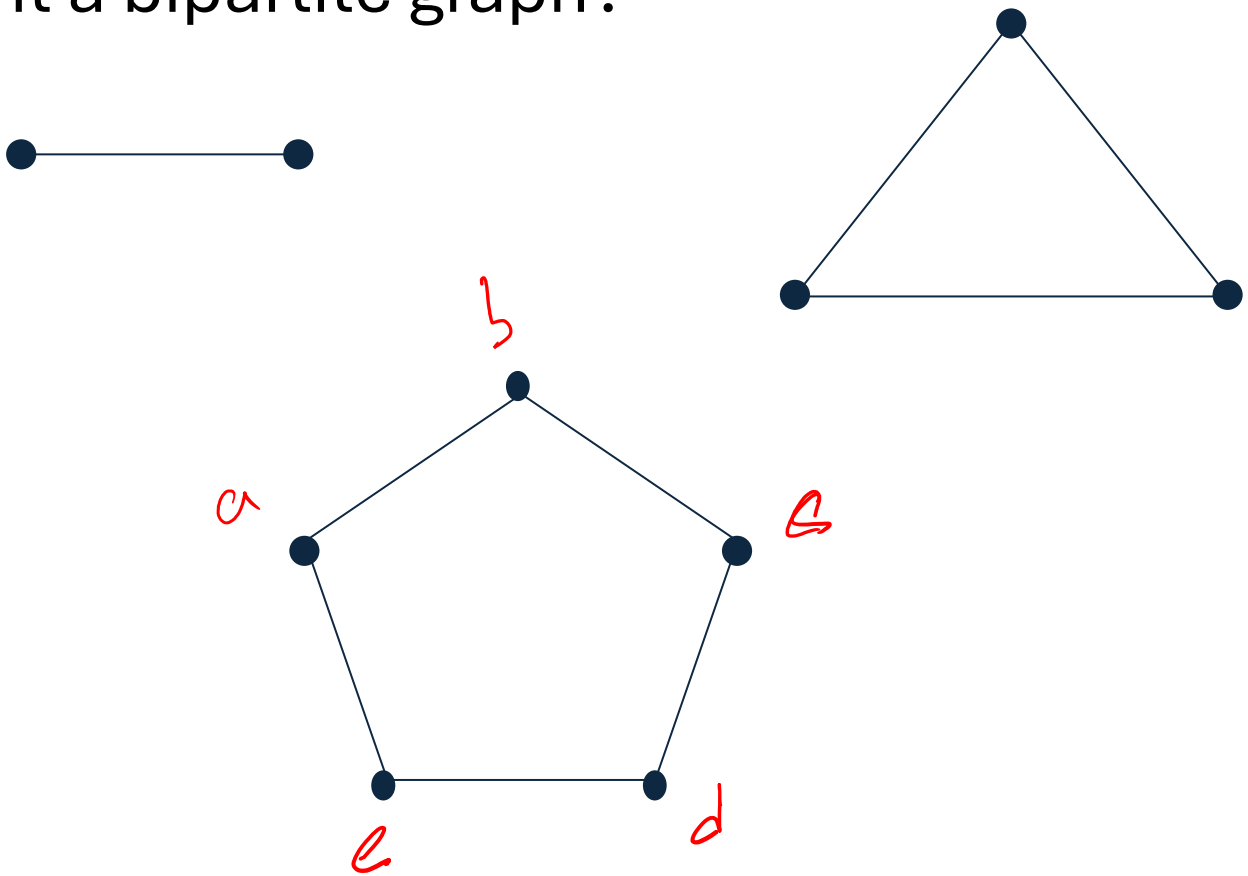
- Finds if two vertices are connected.
- Finds the shortest path between two connected vertices.

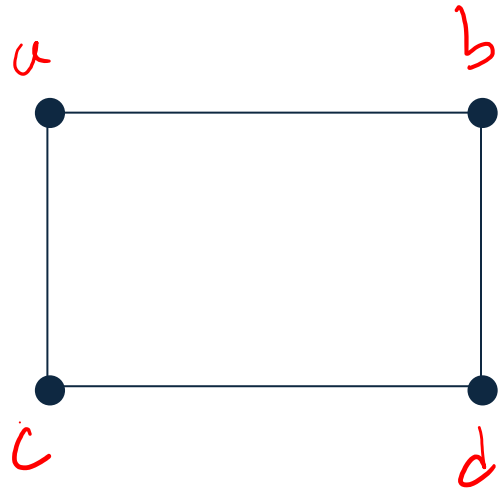
For $j \geq 1$, the layer L_j produced by BFS consists of all the nodes that are exactly at distance j from s (start node).

Running time of BFS?

Testing Bipartiteness using BFS

Is it a bipartite graph?

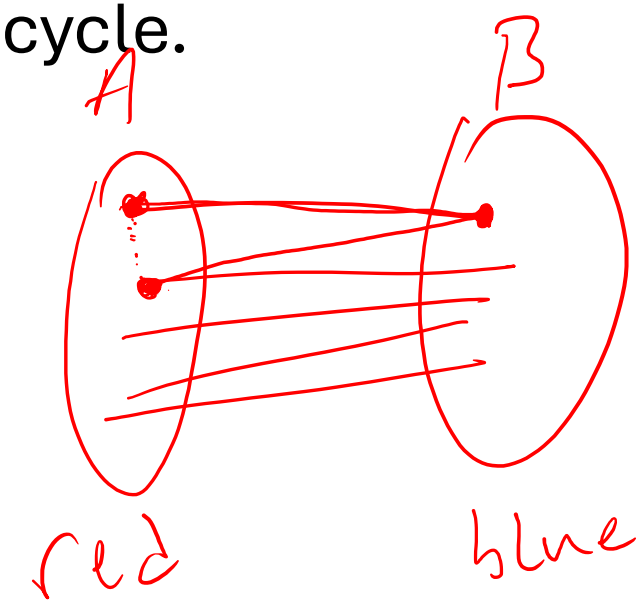




Testing Bipartiteness using BFS

If a graph G having odd length cycle cannot be a bipartite graph.

A bipartite graph cannot contain an odd length cycle.



Testing Bipartiteness using BFS

Start from any vertex and colour it red.

Go to its neighbours and colour them blue.

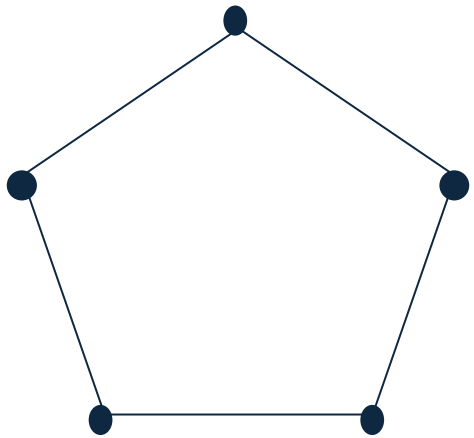
Go to their neighbours and colour them red.

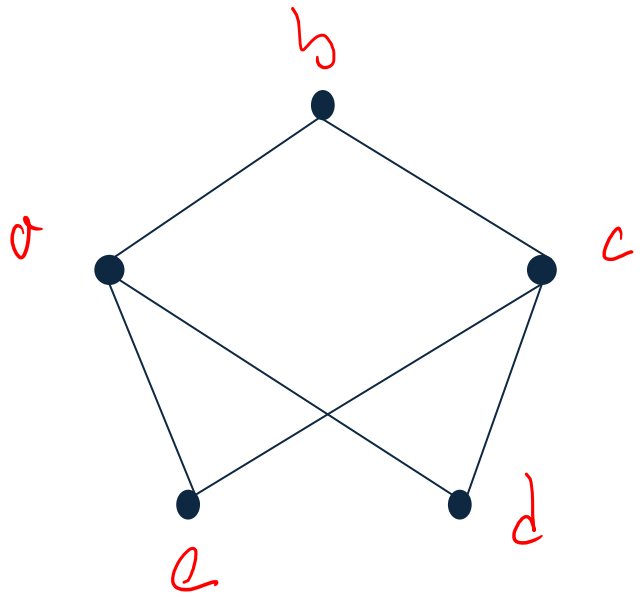
Continue until all the vertices are coloured.

If there is an edge whose vertex pairs are of same colour then the graph is not bipartite. Why?

Think: Let T be a BFS tree, let u and v be nodes in T belonging to layers L_i and L_j respectively, and let (u, v) be an edge of G. Then i and j differ by at most 1.

Testing Bipartiteness using BFS





Testing Bipartiteness using BFS

The bipartite testing algorithm is identical to BFS.

Colour all the vertices in the even numbered layers by red.

Colour all the vertices in the odd numbered layers by blue.