2> cost of a walk: the sum of the costs of the edges that form that walk

2> closed walk : walk that starts and ends in the same vertex

2> cycle: a closed walk of length at least 1, with no repeating vertices or edges.

Graph traversal

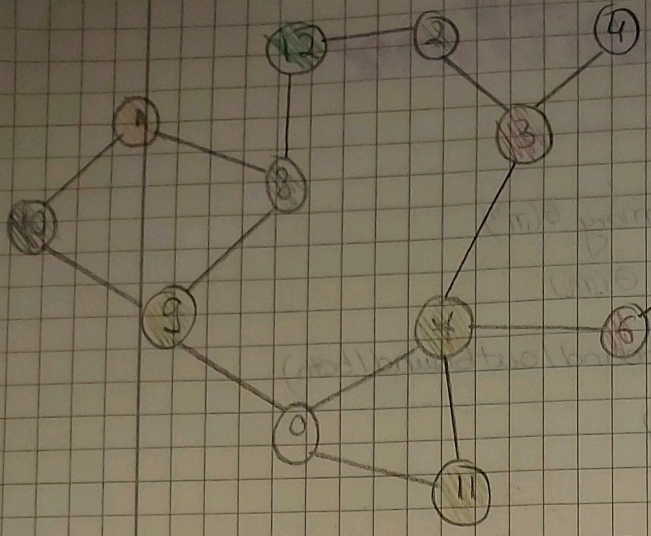
Useful for: finding the shortest path on unweighted graphs

BREADTH-FIRST TRAVERSAL

Time Complexity: $O(V+E)$

↳ fundamental search algorithm

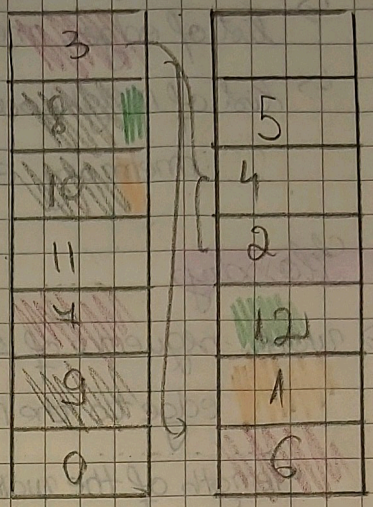
Starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.



Beginning at 0

0's unvisited neighbours

9, 7, 11



Breadth-first traversal

Input:
G: graph
S: a vertex

Output:
accessible: set of vertices accessible from S
prev: a map that maps each accessible vertex to its predecessor on a path from s to it

Algorithm:

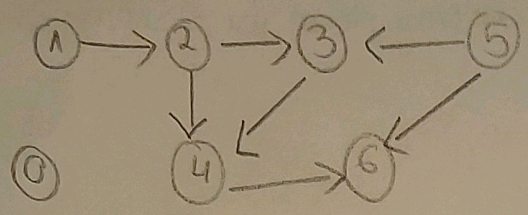
```
Queue q
Dictionary prev
Dictionary dist
Set visited
q.enqueue(s)  //add the starting vertex to the queue
visited.add(s)
dist[s]=0
while not q.isEmpty() do
    x = q.dequeue()
    for y in Nout(x) do
        if y not in visited then
            q.enqueue(y)
            visited.add(y)
            dist[y] = dist[x]+1
            prev[y]=x
```

accessible = visited

BFS
SHORTESTPATH

Lowest length path using BFS

Start vertex: 1



	x	y	Q: queue	visited: set	dist: dictionary	prev: dictionary
init			← 1 ←	{1}	0 1 2 3 4 5 6 0	0 1 2 3 4 5 6 0
iter. 1	x=1	 y=2	← ← ← 2 ←	{1} {1,2}	0 1 2 3 4 5 6 ∞ 0 1	0 1 2 3 4 5 6 1
iter. 2	x=2	 y=3 y=4	← ← ← 3 ← ← 3 4 ←	{1,2} {1,2,3} {1,2,3,4}	0 1 2 3 4 5 6 0 1 2 0 1 2 2	0 1 2 3 4 5 6 1 2 1 2 2
iter. 3	x=3	 y=4	← 4 ←			
iter. 4	x=4	 y=6	← ← ← 6 ←	{1,2,3,4} {1,2,3,4,6}	0 1 2 3 4 5 6 0 1 2 2 3	0 1 2 3 4 5 6 1 2 2 4
iter. 5.	x=6	y=-	← ←			