1 Breath First Search : BFS

1.1 Algorithm

The BFS algorithm is detailed in Algorithm 1. The output is $L_0, L_1, L_2, \ldots L_n$ where L_i = vertices reachable from s in i steps.

Algorithm 1 Breath First Search

procedure BFS($G(V,E), s$) $\triangleright$ $G(V,E)$ is the input graph, $s \in V$
    initialize array *mark* of size $|V|$ with 0 $\triangleright$ complexity $O(|V|)$
    $L_0 \leftarrow \{s\}$ $\triangleright$ L_0 only contains s. $O(1)$
    $mark[s] \leftarrow 1$ $\triangleright$ $O(1)$
    $i \leftarrow 1$ $\triangleright$ $O(1)$
    repeat
        $L_i \leftarrow$ unmarked neighbors of L_{i-1} $\triangleright$ $O(1)$ for every edge. $\leq O|E|$
        mark L_i $\triangleright$ Mark all the nodes in L_i. $O(1)$ for every edge. $\leq O|E|$
        $i \leftarrow i - 1$ $\triangleright$ Increment i. $O(1)$ for every iteration. $\leq O|V|$
    until $L_{i-1} = \phi$
end procedure $\triangleright$ total complexity $O(|V| + |E|$

1.2 Lemma

Note 1.1. Lemma is a claim

Lemma 1.1. *$\forall\, i$ the vertices in L_i are exactly the vertices at distance i from s.*

Proof. **Proof By Induction:** On i, $L_i = \{$ vertices at distance i from $s\}$

Base Case:
$i = 0$. $L_0 = \{s\}$, s only vertex at distance 0 from s.

Induction Hypotheses:
$\forall\, i \leq k \quad L_i = \{$ vertices at distance i from $s\}$

Induction Step:
For $k+1$ we want to show $L_{k+1} = \{$ vertices at distance $k+1$ from $s\}$
Suppose in $v \in L_{k+1}$ $\exists\, u \in L_k \quad (u, v) \in E$ and v is not marked

$$s \xrightarrow{k} u \xrightarrow{1} v$$

inductive hypothesis: u is at distance k from s.
$\Rightarrow$ there exists a path of length $k+1$ from s to v. There is no path of length $i \leq k+1$ from s to v, because v is unmarked $\Rightarrow v \notin L_i$ for $i \leq k$

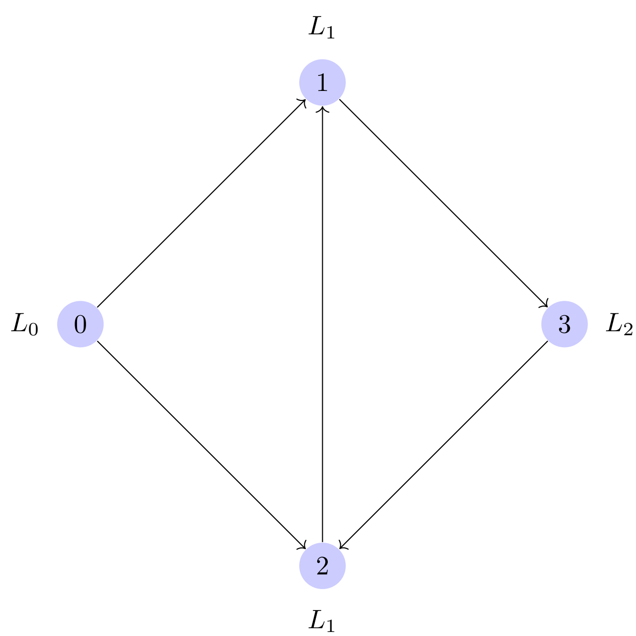
Every vertex $v \in L_{k+1}$ is a neighbor of vertex $u \in L_k$. Hence, there exists a length $k+1$ path from s to v.

If there were a shorter path from $s \rightsquigarrow v$ then by hypothesis v should have been in some layer L_j for $j \leq k$ and not in L_{k+1}

$\square$

1.3 Example

An example of running the BFS search on the graph with start vertex 0.



1.4 Application

Shortest path for unweighted graph.
Web crawling (eg. Google indexing).
Social networking (eg. people you might known)
Network broadcast.
Garbage collection in modern programming languages.
Model checking

2 Depth First Search - DFS

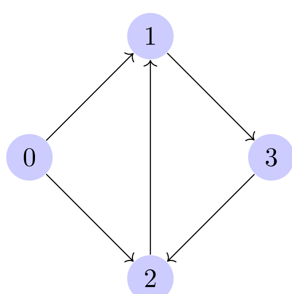
DFS is a back-tracking based search algorithm. It is described as part of Algorithm 2.

Algorithm 2 Depth First Search

procedure DFS$(G(V,E), s)$ ▷ $G(V,E)$ is the input graph, $s \in V$
  $mark[s] \leftarrow 1$ ▷ Total for all vertices $\leq O|V|$
  for each neighbor v of s **do** ▷ Total for all edges $\leq O|E|$
    if $mark[v] \neq 1$ **then**
      DFS(G, v)
    end if
  end for
end procedure ▷ total complexity $O(|V| + |E|)$

Note 2.1. Complexity
marking vertices $\leq O(|V|)$
going over neighborhoods $\leq O(|E|)$
Total: $O(|V| + |E|)$

2.1 Example



One possible DFS starting 0:

$0 \to 1 \to 3 \to 2$

Another possible DFS starting 0:

$0 \to 2 \to 1 \to 3$

2.2 Applications

-cycle detection
-topological sort
-navigating mazes

3 Greedy Algorithms

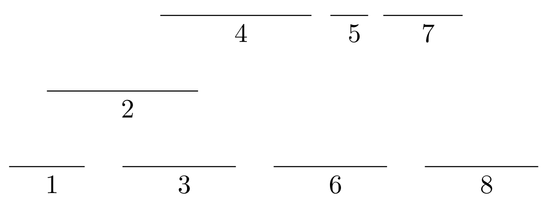
3.1 Interval Scheduling Problem

Input: Activities
Output: Subset of non-overlapping activities that is as large as possible.

Algorithm 3 IntervalSchedule

---

procedure IntervalSchedule(A) $\triangleright$ $A = activities$
    repeat
        Find activities with earliest finish time.
        Remove all overlapping activities.
    until No Activities Remain
    return Subset of non-overlapping activities that is as large as possible.
end procedure



The order, using algorithm 3 is: 1 , 3 , 5 , 7 (4 activities)

3.2 Proof that the algorithm is optimal

Lemma 3.1. *At every step of the algorithm, there exists an optimal solution that picks all the activities that the algorithm picked.*

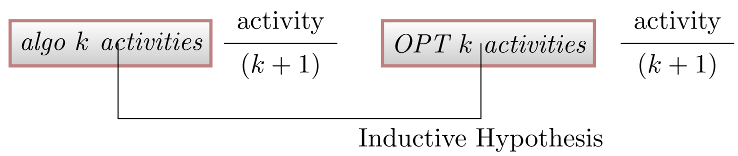
Note 3.1. The lemma implies that the algorithm is optimal.

Proof. **Proof By Induction:** On number of activities

Base Case:
0 activities. Obviously contained in optimal solution

Induction Hypotheses:
After algo picked k activities, there is an optimized solution (OPT) that picks all k activities.

Induction Step:
After the algo picked $k + 1$ activities.



By algo's design its $k+1$ activity has the earliest finish time.

$\square$

3.3 Exchange Argument

OPT' is just like OPT except instead of $(k+1)$th activity that OPT picks, OPT' picks the $(k+1)$th activity of algo
OPT' feasible {no overlapping between activities}
OPT' has as many activities as OPT and is therefore optimal.