Depth-first search (DFS)

Explore graph always moving away from last visited vertex, similar to preorder tree traversals

Pseudocode for Depth-first-search of graph *G*=(*V*,*E*)

ALGORITHM *DFS*(G)

mark each vertex in V with 0 // label as unvisited

$count \leftarrow 0$

for each vertex v in V **do**

  if v is marked with 0

    dfs(v)

//

dfs(v)

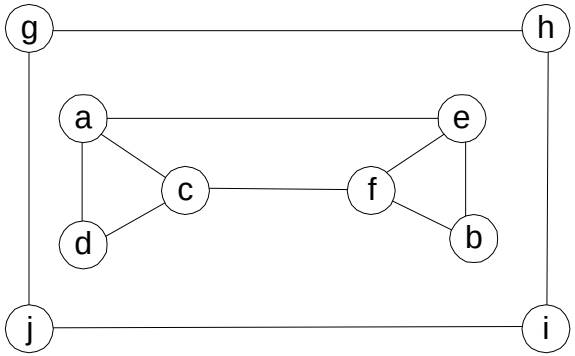
$count \leftarrow count + 1$; mark v with $count$

for each vertex w in V ajacent to v **do**

  if w is marked with 0

    dfs(w)

Example – Undirected Graph



Input Graph

(Adjacency matrix / linked list

$e_{6,2}$

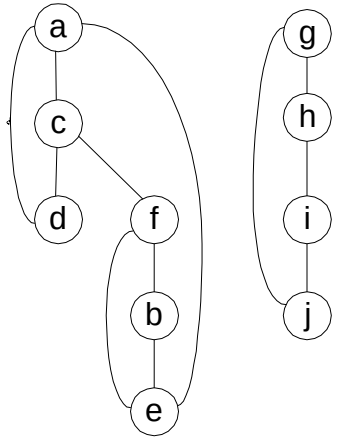
$b_{5,3}$ $j_{10,7}$

$d_{3,1}$ $f_{4,4}$ $i_{9,8}$

$c_{2,5}$ $h_{8,9}$

$a_{1,6}$ $g_{7,10}$

Stack push/pop



DFS forest

(Tree edge / Back edge)

DFS: Notes

DFS can be implemented with graphs represented as:

- Adjacency matrices: $\Theta(V^2)$
- Adjacency linked lists: $\Theta(V+E)$

Yields two distinct ordering of vertices:

- preorder: as vertices are first encountered (pushed onto stack)
- postorder: as vertices become dead-ends (popped off stack)

Applications:

- checking connectivity, finding connected components
- checking acyclicity
- searching state-space of problems for solution (AI)

Breadth-First Search (BFS)

Explore graph moving across to all the neighbors of last visited vertex

Similar to level-by-level tree traversals

Instead of a stack (LIFO), breadth-first uses queue (FIFO)

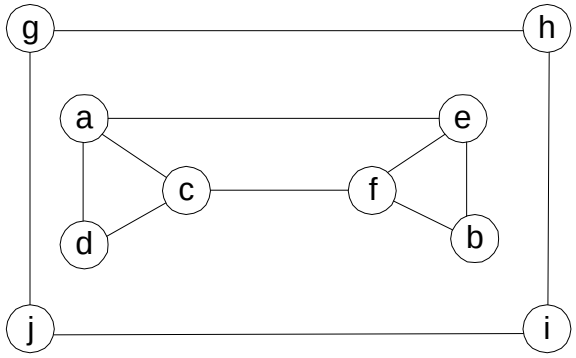
Applications: same as DFS

BFS algorithm

BFS(G)
count ← 0
mark each vertex with 0
for each vertex ***v*** in ***V*** **do**
    bfs(*v*)

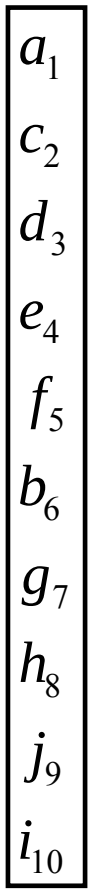
bfs(*v*)
count ← *count* + 1
mark *v* with *count*
initialize *queue* with *v*
while *queue* is not empty **do**
    a ← front of *queue*
    for each vertex *w* adjacent to *a* **do**
        if *w* is marked with 0
            count ←*count* + 1
            mark *w* with *count*
            add *w* to the end of the *queue*
    remove *a* from the front of the *queue*

BFS Example – undirected graph

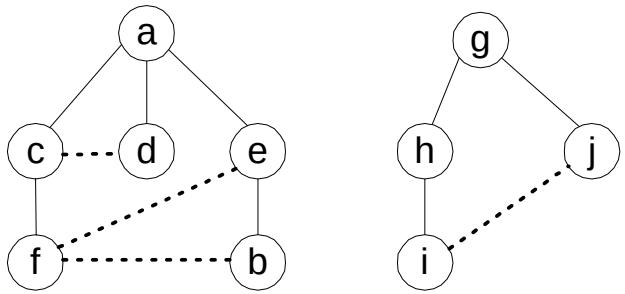


Input Graph

(Adjacency matrix / linked list



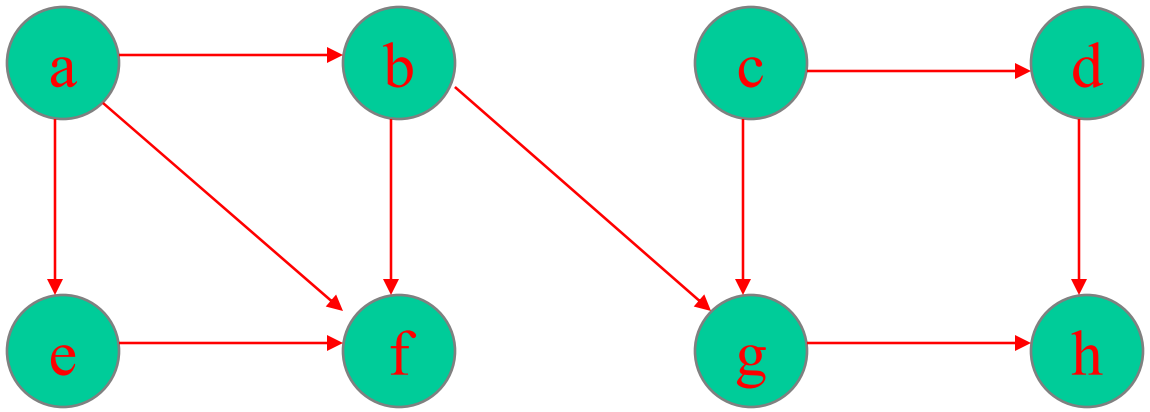
Queue



BFS forest

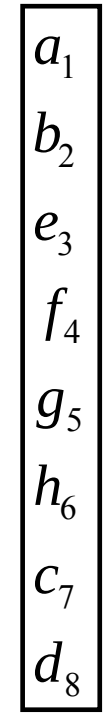
(Tree edge / Cross edge)

Example – Directed Graph

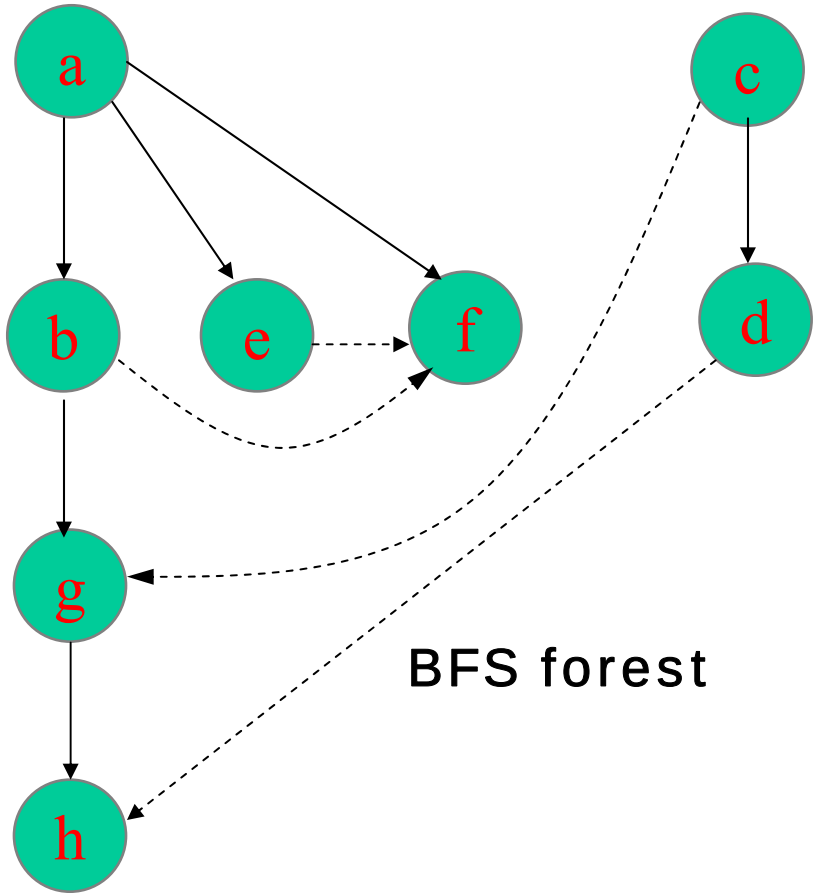


BFS traversal:

BFS Forest and Queue



Queue



BFS forest

Breadth-first search: Notes

BFS has same efficiency as DFS and can be implemented with graphs represented as:

- Adjacency matrices: $\Theta(V^2)$
- Adjacency linked lists: $\Theta(V+E)$

Yields single ordering of vertices (order added/deleted from queue is the same)

Directed Acyclic Graph (DAG)

A directed graph with no cycles

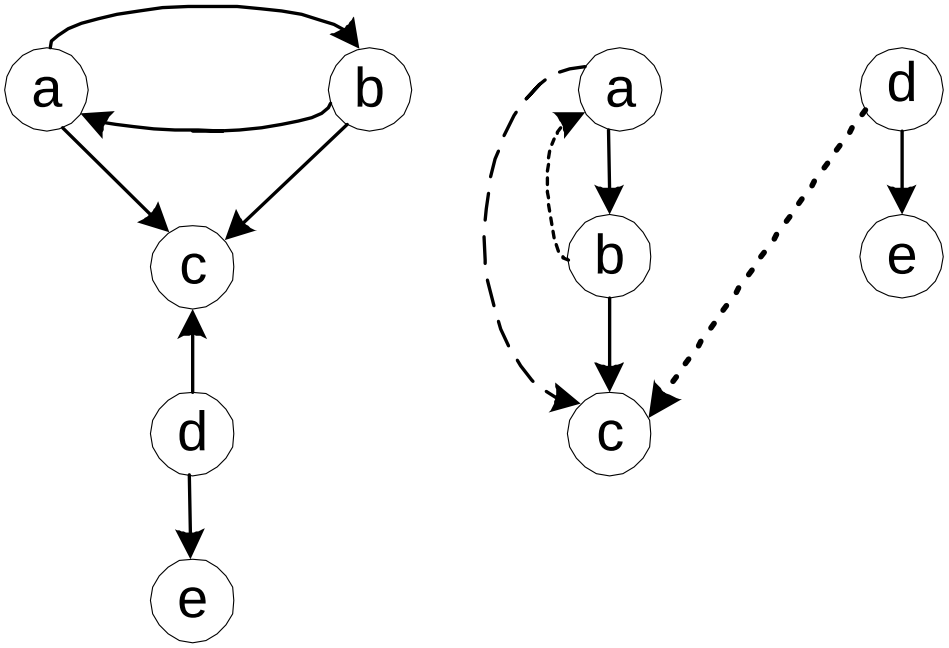
Arise in modeling many problems, eg:

- prerequisite structure
- food chains

A digraph is a dag if its DFS forest has no back edge.

Imply partial ordering on the domain

Example:

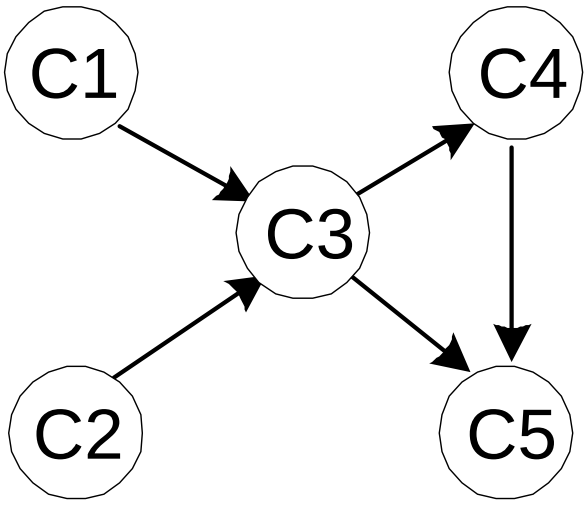


(a) (b)

Topological Sorting

Problem: find a total order consistent with a partial order

Example:



Five courses has the prerequisite relation shown in the left. Find the right order to take all of them sequentially

Note: problem is solvable iff graph is dag

Topological Sorting Algorithms

DFS-based algorithm:

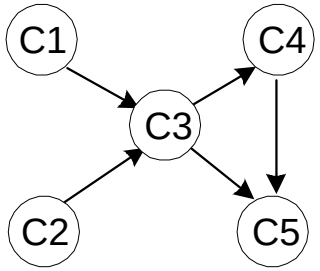
- DFS traversal: note the order with which the vertices are popped off stack (dead end)
- Reverse order solves topological sorting
- Back edges encountered?→ NOT a DAG!

Source removal algorithm

- Repeatedly identify and remove a *source* vertex, ie, a vertex that has no incoming edges

Both $\Theta(V+E)$ using adjacency linked lists

An Example



(a)

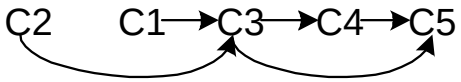
$C5_1$
$C4_2$
$C3_3$
$C1_4$ $C2_5$

(b)

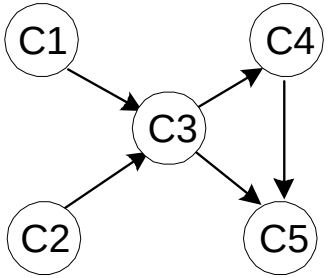
The popping-off order:

C5, C4, C3, C1, C2

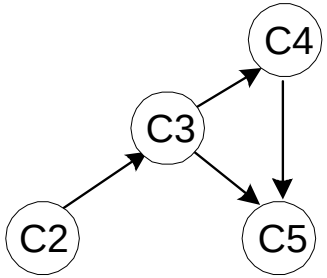
The topologically sorted list:



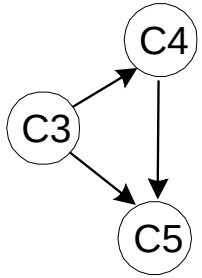
(c)



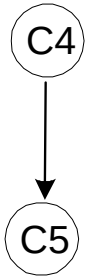
delete C1



delete C2



delete C3



delete C4

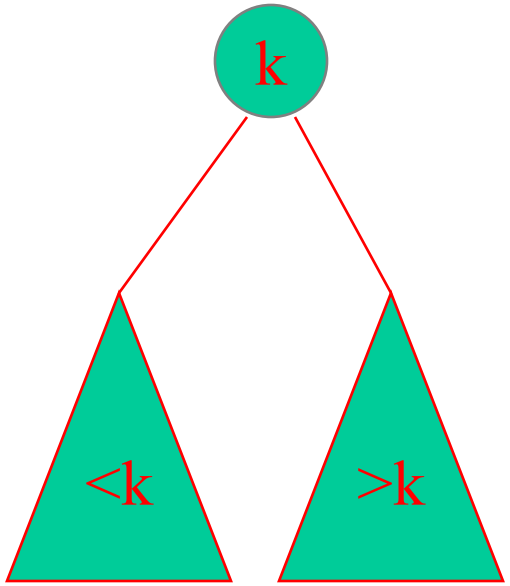
delete C5

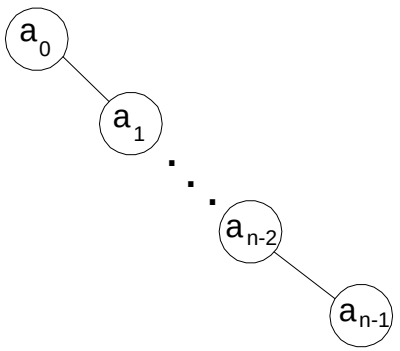
Variable-Size-Decrease: Binary Search Trees

Arrange keys in a binary tree with the *binary search tree property*:

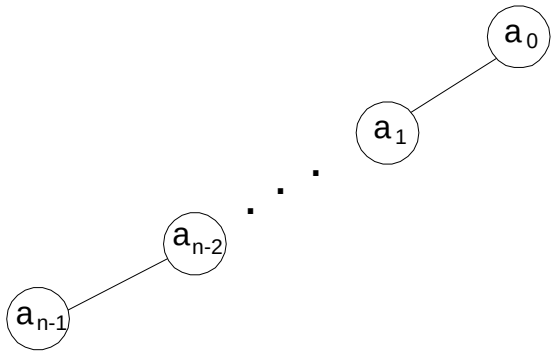
Example 1: 5, 10, 3, 1, 7, 12, 9

Example 2: 4, 5, 7, 2, 1, 3, 6





(a)



(b)

Searching and insertion in binary search trees

Searching – straightforward

Insertion – search for key, insert at leaf where search terminated

All operations: worst case # key comparisons = $h + 1$

$\lfloor \lg n \rfloor \leq h \leq n - 1$ with average (random files) 1.41 lg n

Thus all operations have:

- worst case: $\Theta(n)$
- average case: $\Theta(\lg n)$

<u>Bonus:</u> inorder traversal produces sorted list (*treesort*)