Graph Algorithms

- **Search Algorithms**
  - BFS
    - Spanning Tree Construction
    - Topological Sorting Vertex labeling

$22.1 Intro to two representations for graphs
$22.2 Graph searching algorithm Breadth First search
$22.4 Application of Depth Seach: Topological Sort

Graph Algorithms
$22.1 Intro to two representations for graphs
Terminology – graphs | networks

- G = (V,E) V- set of vertices, nodes
  E- set of edges

  In general, e ∉ E implies that vertices of e are distinct

- **Directed** edge v_1 ⟶ v_2 is denoted by (v_1, v_2)
- **Undirected** edge v_1 ——— v_2 is denoted by $\{v_1, v_2\}$

  Note. It is also rarely used. (v_1, v_2) is used for an edge, if directed it is explicitly stated.

Loop?

Examples—undirected graph and matrix
{(1,2), (1,5),(2,3),(2,4),(2,5),(3,4),(4,5)}

1 2
3
5 4

Examples—directed graph and matrix
{(1,2), (1,4),(2,5),(3,5),(3,6),(4,2),(5,4),(6,6)}
1 2 3
4 5 6

Assumption: graphs are simple unless explicitly specified.

- **Path**
- **Cycle : simple closed**

- **Degree** d(v) of a vertex v
  - **Indegree** = # of edges ending at v
    - Meaningful for directed graphs only

- **Outdegree** = # of edges originating at v

Weighted graph

W: E → R where w(e)=w(u,v) is the weight of the edge, e=(u,v).

Two ways to represent edges: matrix, adjacency lists

Example of Adjacency list

Adjacency list useful for sparse graph, number of edges |E| much smaller than $|V|^2$.

If G is directed graph, the sum of lengths of adjacency lists is |E|, memory needed is O(|V|+|E|)

If G is undirected graph, the sum of lengths of adjacency lists is 2|E| memory needed is O(|V|+2|E|)

Weighted graph

W: E → R where w(e)=w(u,v) is the weight of the edge, e=(u,v).

Adjacency list can be adapted to represent weighted graphs

(u, {v,w}) : (u,v) is an edge with w as the weight.

Advantage of adjacency list: it is useful for sparse graphs
Disadvantage of adjacency list: no quick way to determine if (u,v) is an edge in the graph.

Example of matrix representation

Matrix can be used to represent edge information
M: VxV → R

M(i,j) = 1 if $(v_i,v_j) \in E$
M(i,j) = 0 if $(v_i,v_j) \notin E$
Only one bit needed to record entries.

Adjacency matrix can be adapted to represent weighted graphs

Matrix useful for dense graph, when number of edges |E| close to $|V|^2$.

Adjacency Matrix can be adapted to represent weighted graphs

Matrix can be used to represent edge_wieght information
M: VxV → R

M(i,j) = w if $(v_i,v_j) \in E$
M(i,j) = 0 or • if $(v_i,v_j) \notin E$

Advantage of matrix: it is useful for dense graphs
Disadvantage of matrix: too much memory $O(|V|^2)$.

For undirected graphs, the matrix is **symmetric** and the memory size can be reduced to half.
Note. Asymptotically, both representations behave equivalently. For small graphs matrix representation may be preferable for queries. But overall matrix representation is simpler, easier and preferable.

Exercise 22.1-1

a. outdegree of a vertex =O(length of adjacency list)
   In matrix representation, row (to the right of i) sum is the outdegree(v_i)
b. indegree of a vertex = O(sum of lengths of lists: |E|) because every list is searched for it.
   In matrix representation, column (below i) sum is the indegree(v_i)

GRAPH TRAVERSALS
$22.2

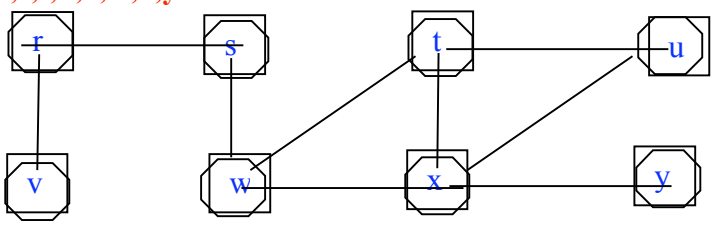
- Scanning / traveling a sequence or a set can be done in a linear order.
- Scanning a graph is not straight forward, it is non-linear
  Two of the algorithms for scanning graphs are:
  - **BFS** - breadth first search
  - **Best First Search - (BFS)**
  - **DFS - depth first search**

Breadth-First search
Applications
Minimum spanning tree
Single Source Shortest paths tree
Tree is a graph with all vertices, no loops and |V|-1 edges.
Breadth-First search creates a Breadth-first tree. It computes the shortest distance (min number of edges) of a vertex-root from all the vertices.
Given a source vertex, traverse all vertices in the graph so that the distance of each vertex is least from the source. Find paths to all vertices in the graph so that the distance of each vertex is least from the source.

Note. In breadth first tree, path from root to any vertex is shortest of all the paths in the graph (directed or undirected).

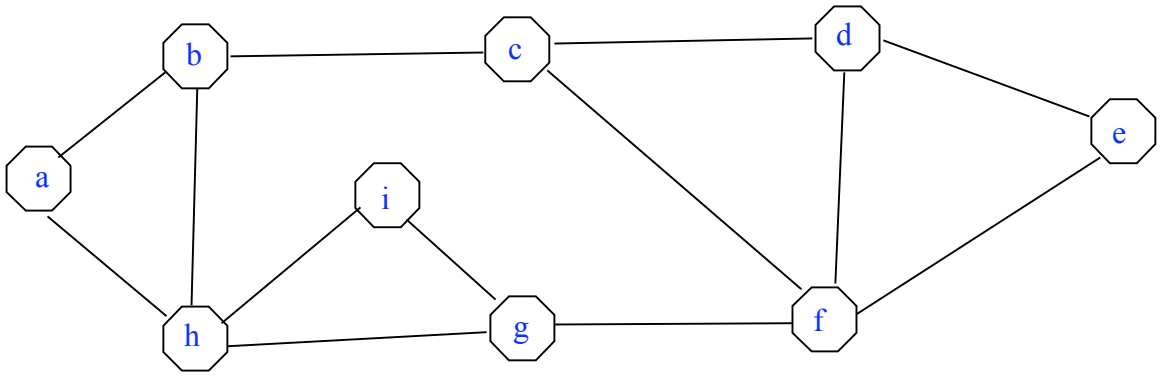
- BFS search
  - Best First Search
- Topological search

The same algorithm works for spanning tree for a graph (undirected).
First example and algorithm {(s,r),(s,w),(r,v),(w,t),(w,x),(t,x),(t,u),(x,u),(x,y),(u,y)}
For consistency, consider edges from left to right or from right to left.
Construct a BFS tree
Depends on the order in which the siblings are visited.
First find all the vertices at distance k before finding a vertex at a distance k+1.
Giving depths.
r,s,t,u,v,w,x,y



Use parents to determine the shortest path from v to s.
Build a graph tree using BFS implementation
BFS uses queue datastructure
Initialization
depth[v]= • , distance from the root

```
label(v) = 0;   parent[v]=null
start
root = s;
depth[s]=0;
count =1;
label(s);   label(s)= count;
parent[s]=null;
insert s in a queue Q
Create v as a node to start the tree
while   Q<>[]
{       Remove v
            for all (v,w) in E   {   if w is not labeled then     label(w)=0;
            {       depth[w]= depth[v]+1;
                    parent[w]=v;
                    count++;
                    label(w)=count;
                    insert w in the queue Q;
                    put (v,w) in the tree to build tree
            }
        }
}
```



Allows you to trace the shortest paths --SSSP.

Labeled vertices correspond to discovered edges and unlabeled vertices correspond to undiscovered edges.

If u is on a path from s to v, then s is the ancestor of u and v is descendent of u.

Complexity

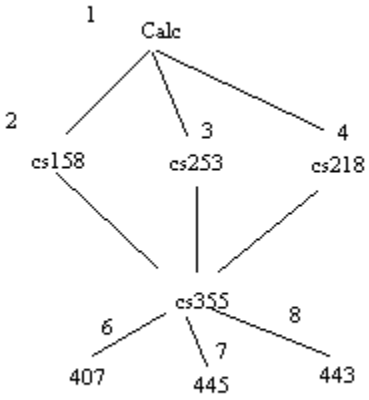
Initialization O(|V|) -- each vertex is marked with: label and depth

Queueing O(|V|)

Edge scanning O(|E|)

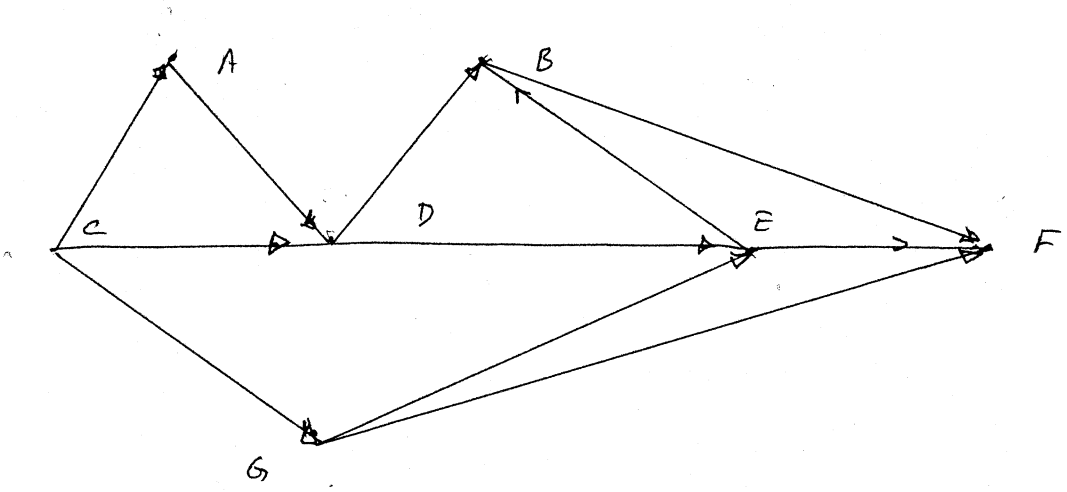
Total complexity is O(|V|+|E|)

$22.4 Topological Search for **directed acyclic** graphs (DAG):
BFS uses queue structure for BFS tree construction.
An example ordering activities so that pre-requisites are satisfied before the activity. That is, label vertice of directed edges so that the label of start vertex is always smaller than the label of end vertex. Queue data structure is useful here too. This is not a tree, but a **directed acyclic** graphs (DAG).

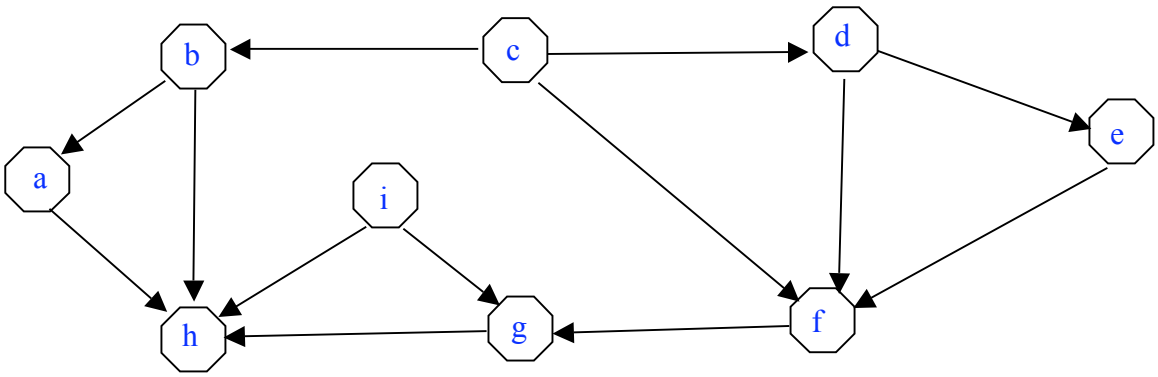


An example of BFS for directed graphs

topological sorting



	indegree
A	1
B	2
C	0
D	2
E	2
F	3
G	1



Directed Acyclic Graph
Implementation of Tree constructions
Topological sort (G)

Initialization

```
For k=1 to n
        label(k) =0
        indegree(k) =0
```

Calculate indegrees

Linked lists are useful for out degrees, Matrix is useful for both in degrees and out degrees.

```
for j=1,n
          for k=1 to n
                if M(k,j) =1
```

indegree(j) =indegree(j)+1;

Build queue

For k=1 to n

parent(k)=null;

If indegree(k) =0, put k in the Queue – this is guaranteed in DAG that there is at least one vertex of indegree zero.

Label vertices

while Queue<>

Remove vertex v from Queue

label(v)= ++count

for each edge (v,j)

indegree(j) =indegree(j)-1;

if indegree(j) =0

insert(j) in the Queue

parent(j)=v

end

Complexity

O(|V|+|E|) repeat loop is executed for each vertex, each every edge is considered.