BFS & DFS

Lydia Zakynthinou

Algo I
MPLA

November 6, 2014

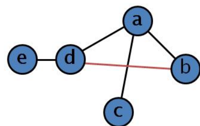
Outline

1. Graph Representation
2. Breadth First Search
3. Depth First Search

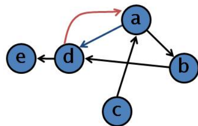
Outline

1. Graph Representation
2. Breadth First Search
3. Depth First Search

Adjacency Matrix



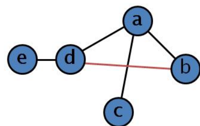
	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



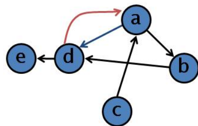
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

- $A(i,j) = \begin{cases} 1 & \text{if } (u_i, v_j) \in E \\ 0 & \text{if } (u_i, v_j) \notin E \end{cases}$
- Undirected $G \Rightarrow A$ symmetric.
- $\Theta(n^2)$ space.
- $O(n)$ time to find all neighbours, $O(1)$ time to check adjacency.

Adjacency Matrix



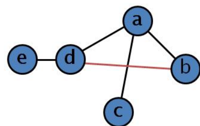
	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



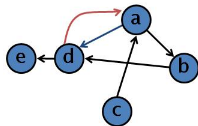
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

- $A(i,j) = \begin{cases} 1 & \text{if } (u_i, v_j) \in E \\ 0 & \text{if } (u_i, v_j) \notin E \end{cases}$
- Undirected $G \Rightarrow A$ symmetric.
- $\Theta(n^2)$ space.
- $O(n)$ time to find all neighbours, $O(1)$ time to check adjacency.

Adjacency Matrix



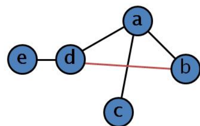
	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



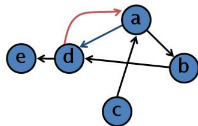
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

- $A(i,j) = \begin{cases} 1 & \text{if } (u_i, v_j) \in E \\ 0 & \text{if } (u_i, v_j) \notin E \end{cases}$
- Undirected $G \Rightarrow A$ symmetric.
- $\Theta(n^2)$ space.
- $O(n)$ time to find all neighbours, $O(1)$ time to check adjacency.

Adjacency Matrix



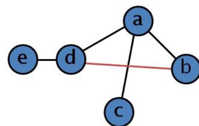
	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



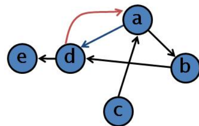
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

- $A(i,j) = \begin{cases} 1 & \text{if } (u_i, v_j) \in E \\ 0 & \text{if } (u_i, v_j) \notin E \end{cases}$
- Undirected $G \Rightarrow A$ symmetric.
- $\Theta(n^2)$ space.
- $O(n)$ time to find all neighbours, $O(1)$ time to check adjacency.

Adjacency Matrix



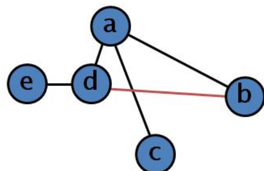
	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0

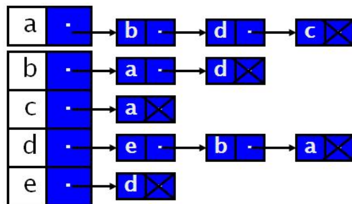


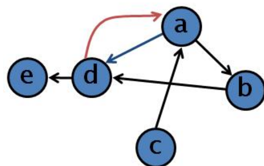
	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

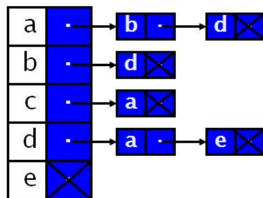
- $A(i,j) = \begin{cases} 1 & \text{if } (u_i, v_j) \in E \\ 0 & \text{if } (u_i, v_j) \notin E \end{cases}$
- Undirected $G \Rightarrow A$ symmetric.
- $\Theta(n^2)$ space.
- $O(n)$ time to find all neighbours, $O(1)$ time to check adjacency.

Adjacency List



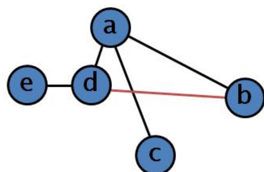


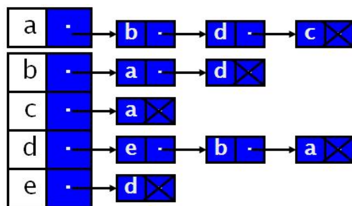


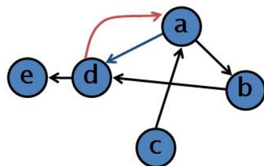


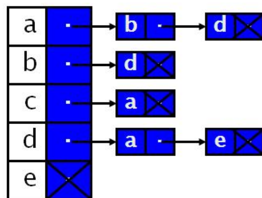
- $\Theta(m)$ space.
- $O(deg(u))$ time to find all neighbours, $O(deg(u))$ time to check adjacency.

Adjacency List



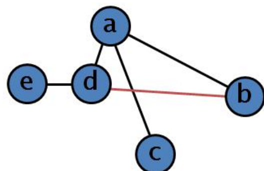


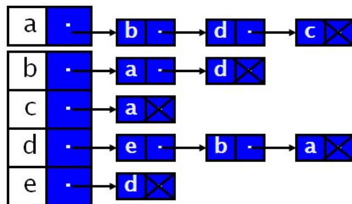


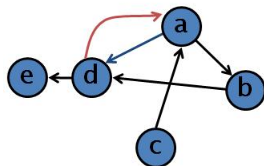


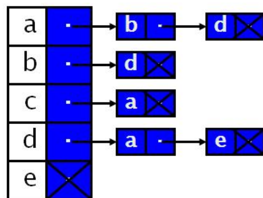
- $\Theta(m)$ space.
- $O(deg(u))$ time to find all neighbours, $O(deg(u))$ time to check adjacency.

Adjacency List









- $\Theta(m)$ space.
- $O(deg(u))$ time to find all neighbours, $O(deg(u))$ time to check adjacency.

Outline

1 Graph Representation

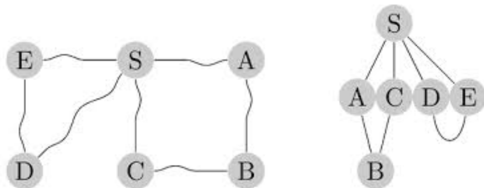
2 Breadth First Search

3 Depth First Search

BFS

- E. F. Moore (1959), The shortest path through a maze. In Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, pp. 285–292.
- C. Y. Lee (1961), An algorithm for path connection and its applications. IRE Transactions on Electronic Computers, EC-10(3), pp. 346–365.

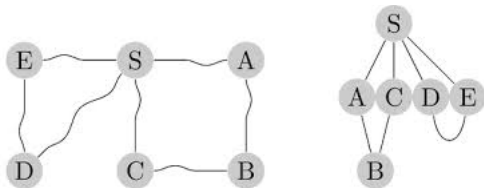
Idea: Start from a vertex *s* and explore all the vertices reachable from *s*. In each round *i* explore vertices within a distance *i* ("breadth").



BFS

- E. F. Moore (1959), The shortest path through a maze. In Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, pp. 285–292.
- C. Y. Lee (1961), An algorithm for path connection and its applications. IRE Transactions on Electronic Computers, EC-10(3), pp. 346–365.

Idea: Start from a vertex *s* and explore all the vertices reachable from *s*. In each round *i* explore vertices within a distance *i* ("breadth").



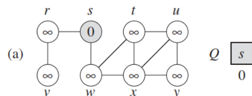
BFS: The Algorithm

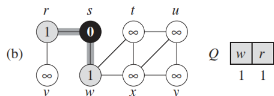
procedure BFS(G, s)
Input: Graph $G = (V, E)$, directed or undirected; vertex $s \in V$
Output: For all vertices u reachable from s, $dist(u)$ is set to the distance from s to u.

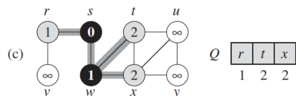
```
for all u ∈ V do
    dist(u) = ∞

dist(s) = 0
Q = [s]                                  ▷ queue containing just s
while Q is not empty do
    u = EJECT(Q)
    for all edges (u, v) ∈ E do
        if dist(v) = ∞ then
            INJECT(Q, v)
            dist(v) = dist(u) + 1
```

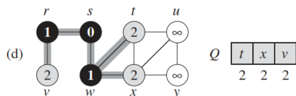
Running time: $\Theta(n + m)$

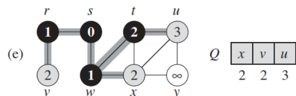
BFS: Example

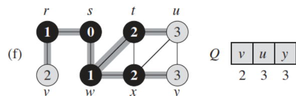


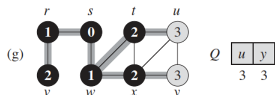


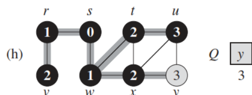


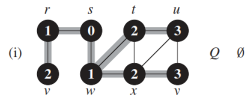












BFS: Applications

- Explore all vertices in the same connected component with s.
- Shortest paths from s to every other vertex in the connected component (for unweighted graphs).
- Testing Bipartiteness
- Useful subroutine for Ford-Fulkerson method (maximum flow).

Outline

1. Graph Representation
2. Breadth First Search
3. Depth First Search

DFS

Invented by Charles Pierre Trémaux (1859-1882) as a maze solving algorithm.

Idea: Start from a vertex s and explore a path as far as it goes. For every vertex u, we explore recursively all reachable vertices from u before we are finished with it (stack).

DFS

Invented by Charles Pierre Trémaux (1859-1882) as a maze solving algorithm.

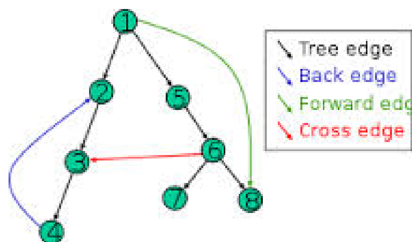
Idea: Start from a vertex s and explore a path as far as it goes. For every vertex u, we explore recursively all reachable vertices from u before we are finished with it (stack).

DFS: The Algorithm

```
procedure dfs(G)

for all v ∈ V:
   visited(v) = false

for all v ∈ V:
   if not visited(v):  explore(v)

procedure explore(G, v)
Input:    G = (V, E) is a graph; v ∈ V
Output:   visited(u) is set to true for all nodes u reachable from v

visited(v) = true
previsit(v)
for each edge (v, u) ∈ E:
   if not visited(u):  explore(u)
postvisit(v)
```

Running time: $\Theta(n + m)$

DFS: Edges

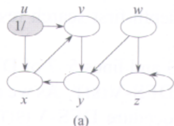


Graph edges belong to four categories:

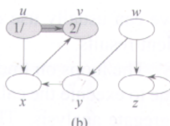
- Tree edges: part of the DFS forest
- Back edges: lead to an ancestor in the DFS tree
- Forward edges: lead from a vertex to a *nonchild* descendant
- Cross edges: lead to a node that has already been explored (neither ancestor nor descendant)

In undirected graphs there are only tree and back edges.

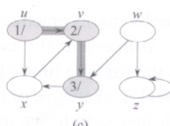
DFS: Example



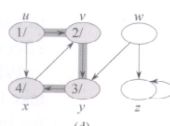
(a)



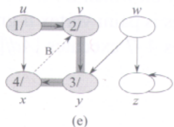
(b)



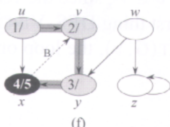
(c)



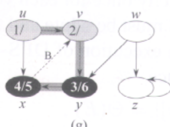
(d)



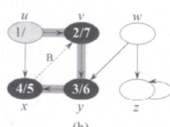
(e)



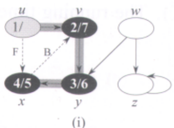
(f)



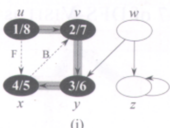
(g)



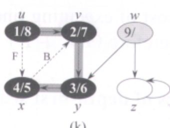
(h)



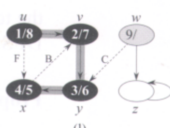
(i)



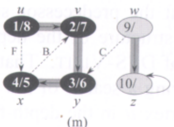
(j)



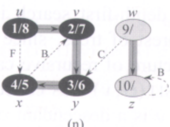
(k)



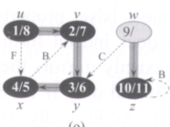
(l)



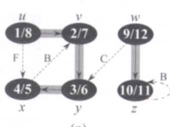
(m)



(n)



(o)



(p)

DFS:Pre/Post-visit

Connected Components:

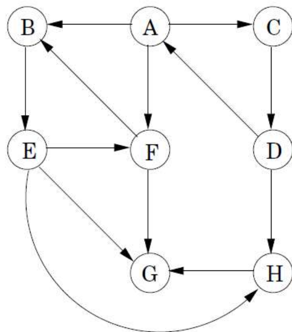
```
procedure previsit(v)
ccnum[v] = cc
```

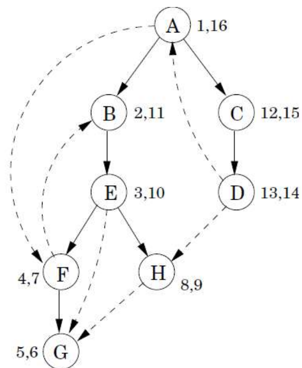
Orderings:

```
procedure previsit(v)
pre[v] = clock
clock = clock + 1

procedure postvisit(v)
post[v] = clock
clock = clock + 1
```

pre/post *ordering for* (u, v)	*Edge type*
$[_u \; [_v \;]_v \;]_u$	Tree/forward
$[_v \; [_u \;]_u \;]_v$	Back
$[_v \;]_v \; [_u \;]_u$	Cross

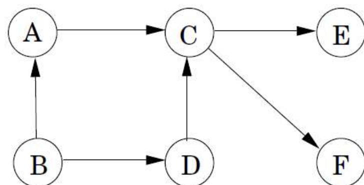
DFS: Example with Timestamps





DFS: DAGs

Directed Acyclic Graphs: Directed graphs without cycles (no DFS back edges). Is it possible to *linearize* a DAG?



DFS: Linearization

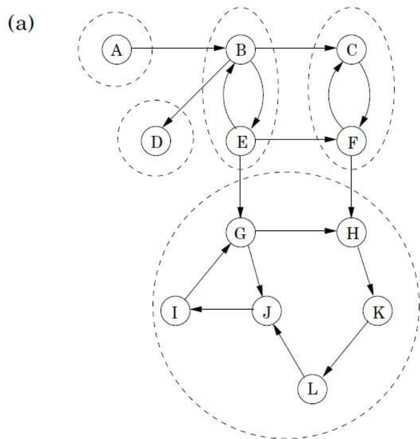
- Perform tasks in decreasing order of their post-numbers.
- Find a source, put it in first in the order, delete it from the graph, repeat.

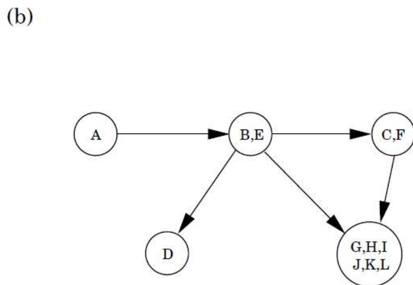
DFS: Linearization

- Perform tasks in decreasing order of their post-numbers.
- Find a source, put it in first in the order, delete it from the graph, repeat.

DFS: Strongly Connected Components

Any directed graph is a DAG of its strongly connected components.





How can we find a decomposition of a graph to its strongly connected components?

DFS: Applications

- Strongly Connected Components
- Planarity Testing (de Fraysseix–Rosenstiehl planarity criterion)

DFS: Applications

- Strongly Connected Components
- Planarity Testing (de Fraysseix–Rosenstiehl planarity criterion)

Thank you! ☺