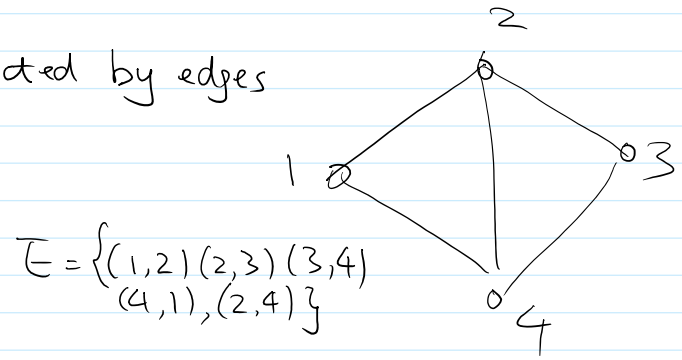
COMPSCI 330 Lecture 8 Graphs

Tuesday, September 20, 2016 5:14 PM

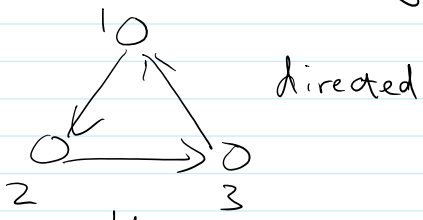
- Graphs and Representations
- Depth First Search
- Breadth First Search

- Graph
  - a set of nodes (vertices) connected by edges
  - (V, E) $V = \{1, 2, \ldots, n\}$
    edge (i, j)

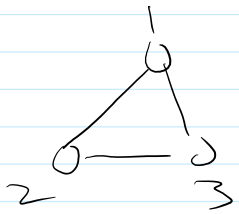


$E = \{(1,2)(2,3)(3,4)(4,1),(2,4)\}$

- abstraction
  - Examples
    1. road network
    2. supplies network
    3. social network (edges = friends)
    4. dependency network (COMPSCI201 $\longrightarrow$ 330)
    5. internet (edges = hyperlinks)
  - directed and undirected graph



directed



undirected

  - graph problems
    1. shortest path
    2. minimum spaning tree
    3. community detection
    4. scheduling
    5. page rank
  - represent (store) a graph
    - adjacency array
      $A[1..n, 1..n]$
      $A[i,j] = 1$ if (i,j) is an edge
      $= 0$ if (i,j) is not an edge.

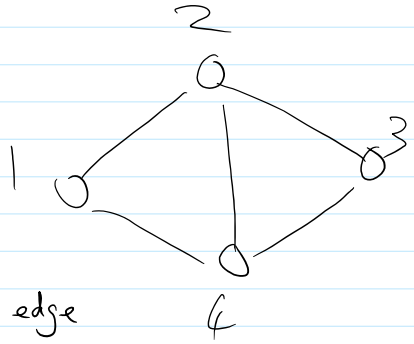
benefit: simple, quickly know whether $(i,j) \in E$

but: take $\Theta(n^2)$ space

- adjacency list

store a list for every vertex
list i contains set of edges from vertex i

1: [2,4] 2: [1,3,4]
3 [2,4] 4: [1,2,3]



benefit: take $\Theta(m)$ space

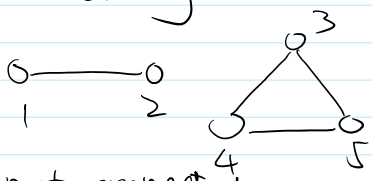
but: hard to know whether (i,j) is an edge

- Basic Graph algorithms
  - Graph traversal: want to visit all nodes of graph G by following edges.
  - DFS (Depth First Search)

```
DFS
for i = 1 to n
    if i is not visited
        DFS_visit(i)
```

```
DFS_visit(i)
    mark i as visited
    for each edge (i,j)
        if j is not visited
            DFS_visit(j)
```

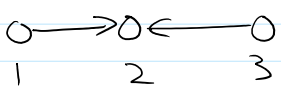
- why do we need ↑ this loop?
  - Connectivity



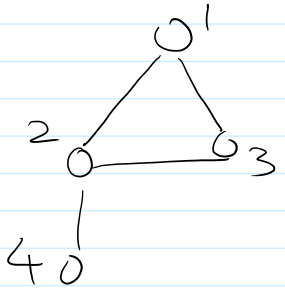
not connected

(Undirected Graph is connected if $\forall i,j$, there is a path from i to j)

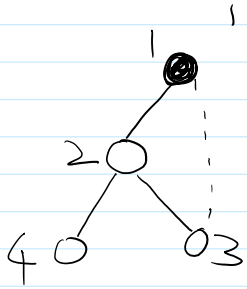
  - directed



- Depth First Search Tree
  - j is a child of i, if DFS_visit(i) called DFS_visit(j)



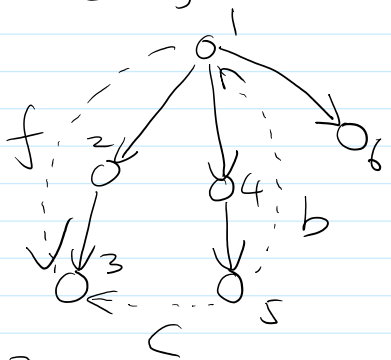
DFS: 0 → 0 → 0 → 0 → 0
1 2 3 2 4



- Pre-order and post-order
  Pre-order: order of visits (1, 2, 3, 4)
  Post-order: order of DFS_visit(i) returns (3, 4, 2, 1)
- DFS and stack

Preorder: order of entering the stack
Post-order: order of exiting the stack

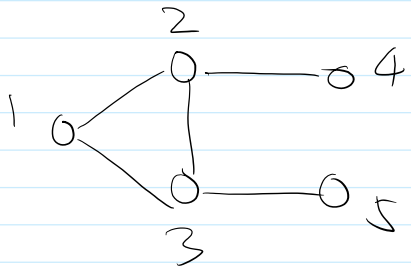
- edge types



tree edge
f: forward edge
b: backward edge
c: cross edge

- BFS (Breadth First Search)

```
BFS_visit(i)
    put i into a queue (mark i as visited)
    while queue is not empty
        u ← dequeue
        for each edge (u,v)
            if v is not visited
                put v into the queue
                mark v as visited.
```



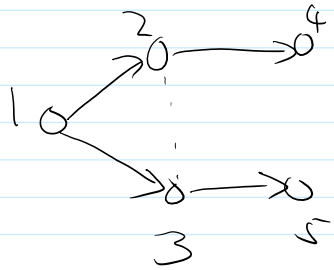
queue 1 2 3 4 5

BFS order : order of entering the queue

Can be used to compute shortest path.

BFS tree

v is a child of u, if v is added to the queue when processing u



(shortest path tree)