Parma University, Math. Deptartment

Algorithms and Data Structures
Searching graphs (breadth first search)

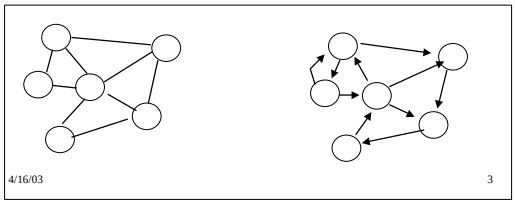
Marco Pellegrini

Summary of this lecture.

- Representation of Graphs:
  - Adjacency-list
  - Adjacency matrix
- Breadth-first search
- Shortest path
- Breadth-first trees
- Material covered in CLR 23.1-2.

Graphs

- G = (V,E). A Graph is formed by a set of vertices V and by a set of Edges E where an edge is a pair of vertices.
- We distinguish undirected graphs where (u,v) = (v,u) and directed graphs where (u,v) and (v,u) are different edges.

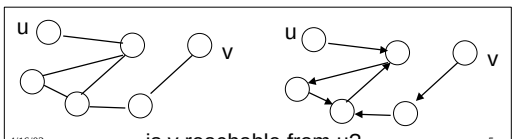


Terminology

- Edge (u,v) is said to be ***incident*** to nodes u and v.
- If (u,v) is oriented, then the edge ***leaves*** u and ***enters*** in v.
- If (u,v) is in E then is called ***adjacent*** to v (and u).
- In an undirected graph the ***degree*** of a node u is the number of nodes adjacent to u.
- In a directed graph the ***in-degree***(u) is the number of edges entering node u.
- In a directed graph the ***out-degree***(u) is the number of edges leaving a node (u).

More terminology.

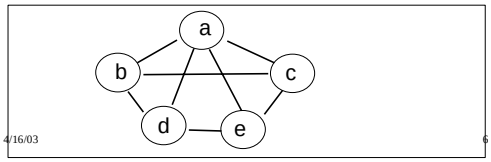
- A ***path*** from u to v of length k is a sequence of nodes: $v_0,v_1,...,v_k$.
  Where v_0=u, v_k=v and, for every i=1,2,..k, (v_i,v_{i+1}) is in E.
- A path is ***simple*** if all the nodes on it are distinct.
- A node v is ***reachable*** from u if there is a path starting at u and ending at v.



is v reachable from u?

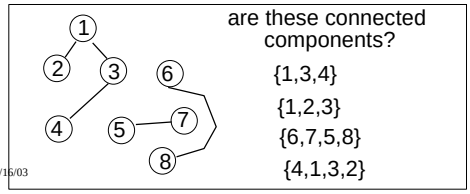
More terminology, cntd.

- A path is called a ***cycle*** if it starts and end at the same node.
- A ***simple cycle*** has all its nodes distinct (except for the first and last node).
- A graph with no cycles is called ***acyclic***.
- How many simple cycles can you find in this picture?



Some more terminology

- An undirected graph G is called ***connected*** if any two vertices are reachable for each other.
- A ***connected component*** of G is a subset of V' of V such that every node in V' can reach every node in V' by using in the path only nodes in V', and V' cannot be made bigger by adding a node in V/V'.

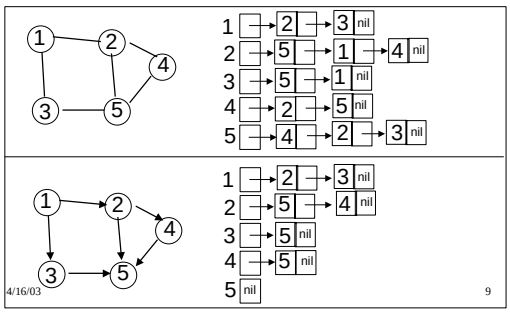


How can we represent graphs in a computer?

- Suppose we use to indicate the nodes the integer numbers from 1 to n: V = [1...n].
- There are two main methods:
  1) Adjacency lists.
  2) Adjacency matrix.
- Adjacency lists are used when the graph is sparse, that is when $|E| << |V|^2$.
- Adjacency matrix is used when the graph is dense, that is |E| close to $|V|^2$.

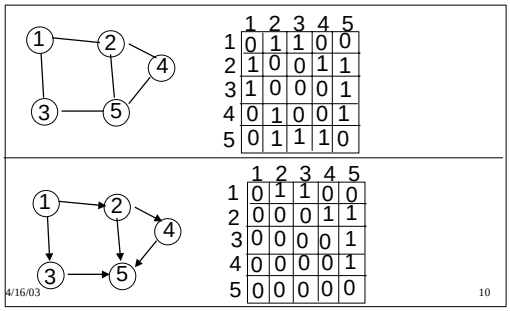
Adjacency list representation.

- We create an array A[1..n] and in location A[i] we put a list of nodes adjacent to node i.



Adjacency matrix representation.

- We create a matrix A[1..n][1..n] and in location A[i][j] we put 1 if (i,j) is in E, 0 otherwise.



Exercise + Homeworks

- Exercise: Given a directed graph represented as adjacency list, describe an algorithm to compute the in-degree and the out-degree of each node.
- HW4.1 (CLR 23.1-3) Given a directed graph G represented as adjacency list describe and analyze an efficient algorithm to compute the transpose of G (that is the graph obtained by reversing every edge in G). Repeat with G as adj. matrix.
- HW4.2 (CLR 23.1.5) Given a directed graph G describe an efficient algorithm to compute the square of G (see book for definition). Analyze your algorithms when G is given as adjacency list and when it is given as adjacency matrix.

Breadth first search.

- In a graph G we select a node s, called the source and we wish to visit all nodes in G starting from the source.
- We assume G to be connected to make things simple.
- The main idea is to visit all nodes at distance k from s before all nodes at distance k+1, for k=1,2...
- We use colors to track what is happening:
  White: the node has not been discovered yet.
  Black: the node and all of its neighbors have been discovered.
  Grey: the node has been discovered but not all of its neighbors (i.e. some neighbor might be white).

Breadth first search (2)

- Input: G=(V,E) , V =[1..n].
- Output: Array d[1..n] where d[u] is the distance between s and u, that is the length of the shortest path from s to u.
- Output: Array p[1..n] representing for each node u a link in the shortest path from s to u. We can reconstruct a shortest path from s to u using p.
- Auxiliary data structures:
  1) An array color[1..n] recording the dolor of a node.
  2) A queue Q for nodes.

Breadth first search (3)

```
BFS_init(G,s)
    % Initialization
      FOR each u in V(G)/{s} DO
         color[u] := white
         d[u] := infinity
         p[u] := NIL
      END FOR
      color[s] := gray
      d[s] := 0
      p[s] := NIL
```

Now the source is grey since we visit it first and we have not visited its neighbors.
The distance of s from itself is 0.
Every other node is not visited yet, so it is white and since we do not know whether G is connected it might by unreachable (that is at infinity).

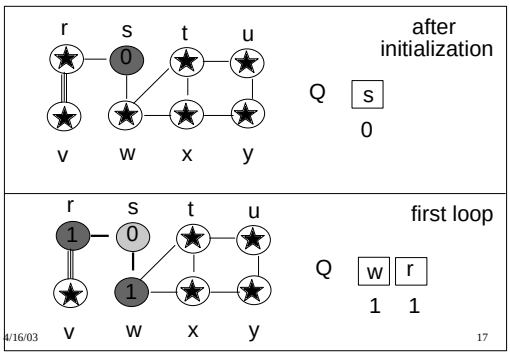
Breadth first search (4)

```
BFS_Main(G,s)
    make-empty-queue(Q)
    enqueue(Q,s)
    WHILE NOT empty(Q) DO
        u := head(Q)
        FOR each v in Adjacency-list[u] DO
            IF color[v] = white
              THEN
                color[v] := grey
                d[v] := d[u] + 1
                p[v] := u
                enqueue(Q,v)
            END IF
        END FOR
        dequeue(Q)
        color[u] := black
    END WHILE
```

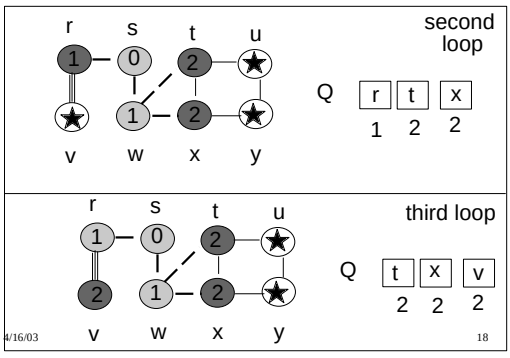

Time analysis of BFS

- A node is put in the queue Q only when it turns from white to grey. This happens only once for a node..
- After a node is deleted from th queue it is marked black so it cannot go back in the queue.
- So all queue operations are executed |V| times and each operation takes constant time.
- We scan the adjacency list of a node when this is taken from the queue. Since a node cannot return in the queue, all FOR loops together take time O(|E|).
- The running time is O(|V| +|E|) using the adjacency list representation.

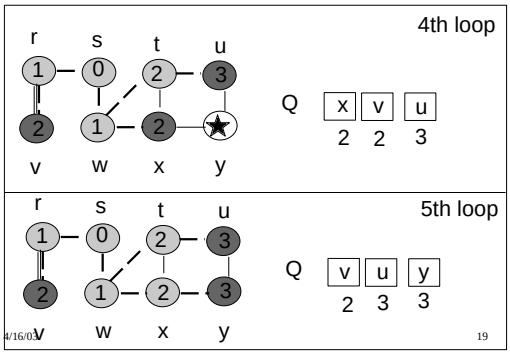
Example (1)



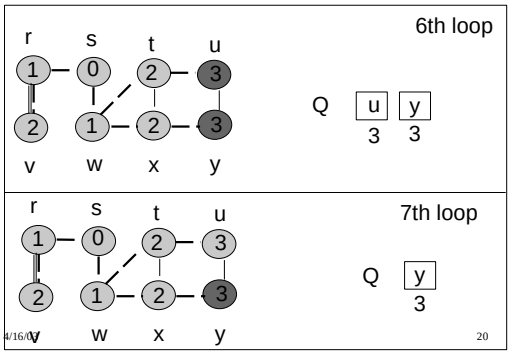
Example (2)



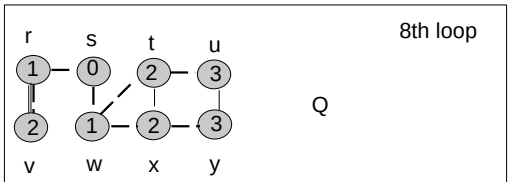
Example (3)



Example (4)



Example (5)



Exercise

- (CLR 23.2-2) Show the result of running BFS on the graph of the example but starting from node u. In particular show the content of the queue and the value of d[] and p[] on the nodes.

Proof of correctness (sketch)

- Theorem 23.4 (CLR)
  (1) During the execution of BFS on G=(V,E) from s, the algorithm discovers every vertex v that is reachable for s.
- (2) Upon termination d[u] is the shortest distance between s and u.
- (3) A shortest path from s to u is given by the shortest path from s to p[u], followed by the edge (p[u],u).
- Idea of proof. The proof's idea is to use induction on the length of the shortest paths from s to the nodes, exploiting the fact that in the queue Q we have closest nodes to s before farthest nodes.
- HW4.3 Read the proof at pages 472-475.

Breadth-first tree.

The array p produced by BFS can be used to produce the shortest path from any node to s.

Implicitly p stores a sub-graph of G that is a tree and visits every node reachable from s.

```
Print-path(G,s,v)
    IF v=s
        THEN Print(s)
        ELSE IF p[v] = NIL
                THEN print "no path from v to s"
                ELSE Print-path(G,s,p[v])
                     Print(v)
             END IF
    END IF
```

Conclusions + Homework

- We have seen two ways to represent graphs adjacency lists and adjacency matrix
- We have seen an algorithm BFS to visit the nodes of a graph so that no node is visited twice.
- BFS is particularly suited for shortest-path computations.
- In next lecture CLR 23.3 we will see a different strategy called Depth First Search which is also used to solve many problems.
- HW4.4:You are given a connected undirected graph G as adjacency list. Modify BFS to detect whether G has a cycle.