CSE 350B

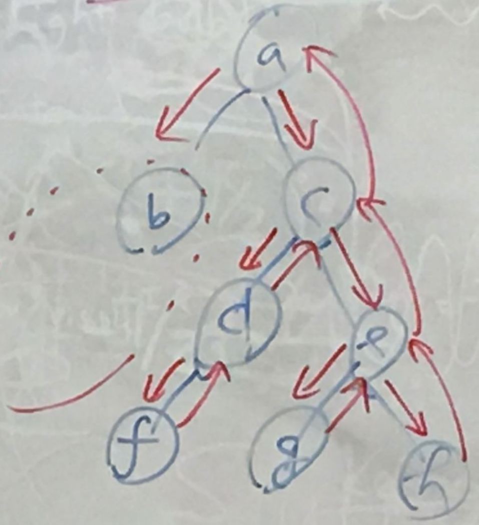
HW2 due on 11-8-16

Exam 2 on 11-15-16 @5PM IN MONT 104

Exam 3 on 12-8-16 @3:30PM

FACT: TREE TRAVERSAL TAKES $O(n)$ TIME, n being the # of nodes in the tree.

IN-ORDER



EACH NODE IS LOOKED AT AT MOST 3 TIMES.

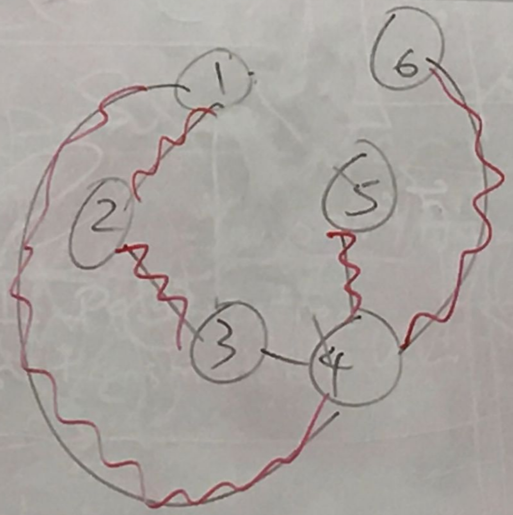
$\Rightarrow$ TOTAL RUN TIME $= O(n)$.

GRAPH SEARCH.

INPUT: AN UNDIRECTED GRAPH G(V,E).

GOAL: SEARCH through the NODES.

BREADTH FIRST SEARCH: (BFS).



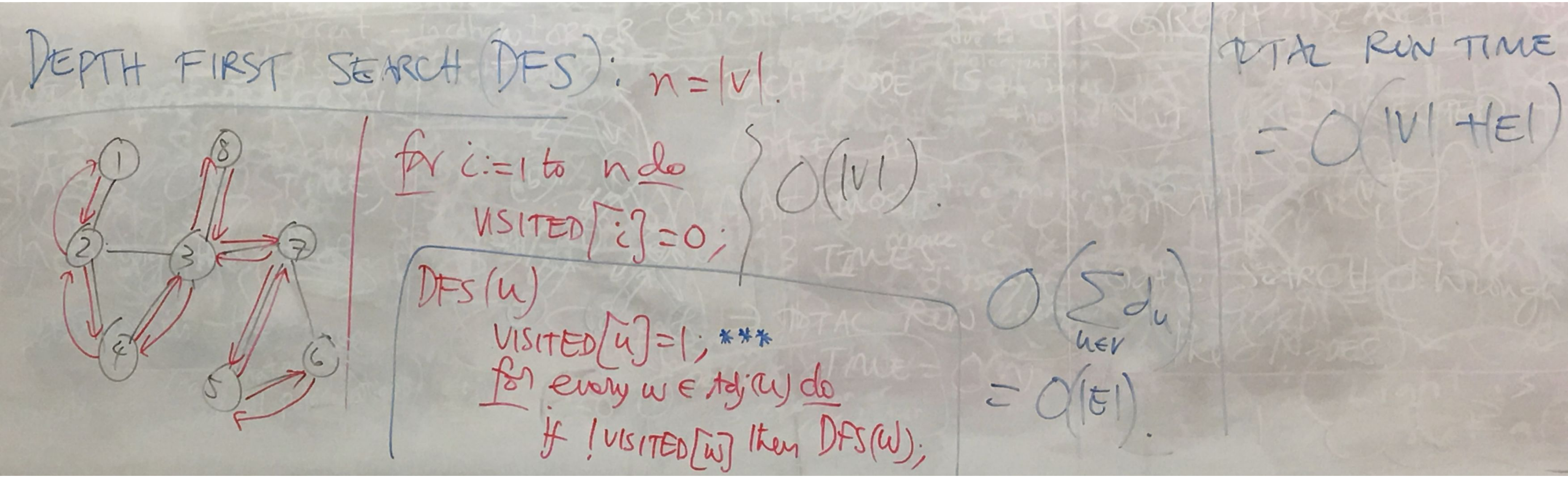
START FROM NODE 1.

VISIT ALL the NODES AT A DISTANCE OF 1.

NODES 2,4.

VISIT the NEIGHBORS of these at a distance of 1.

3,5,6



DEPTH FIRST SEARCH (DFS): $n = |V|$.

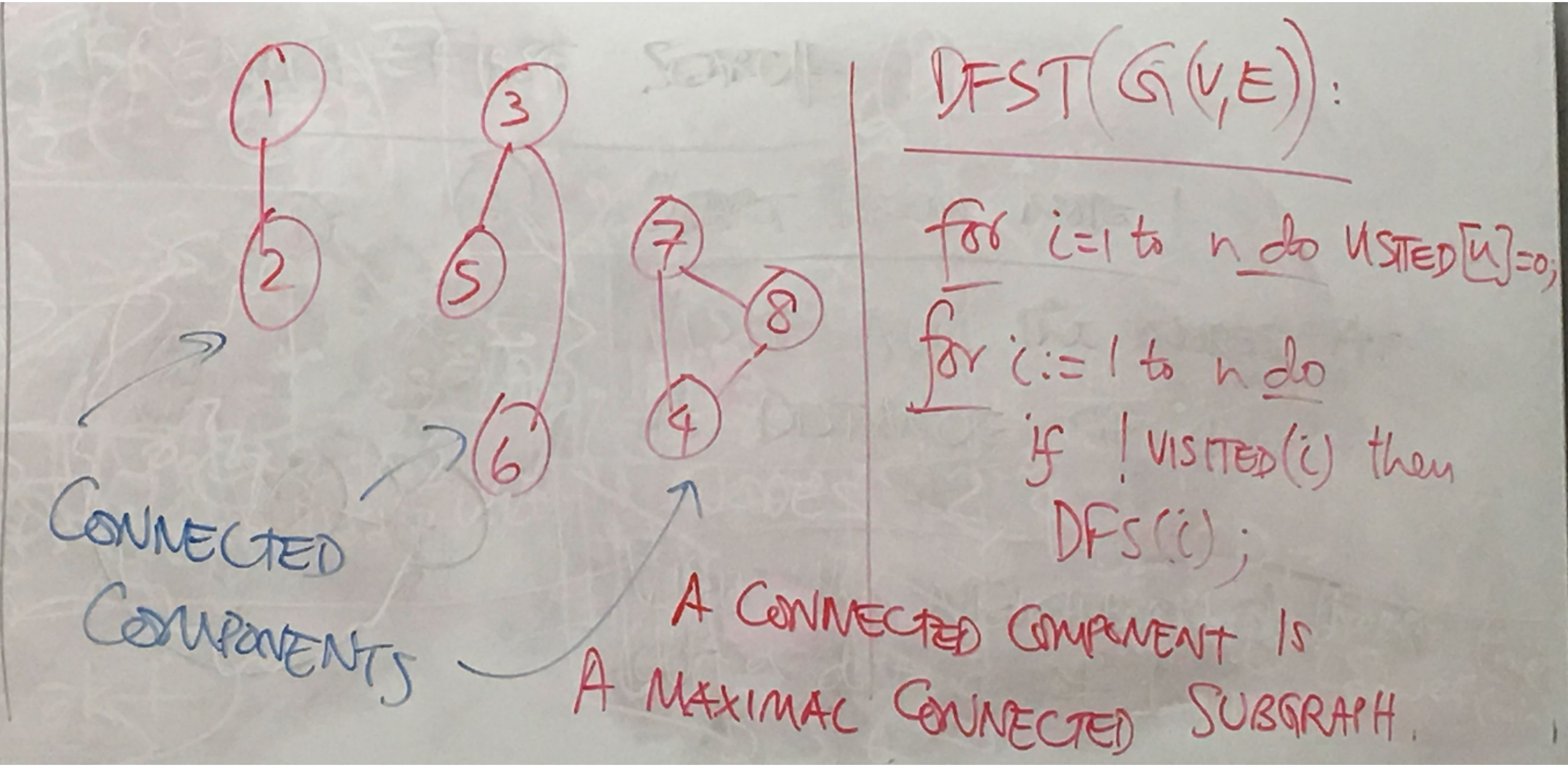
```
for i:=1 to n do
    VISITED[i] = 0;        } O(|V|).

DFS(u)
    VISITED[u]=1; ***
    for every w ∈ Adj(u) do
        if !VISITED[w] then DFS(w);
```

$O\left(\sum_{u \in V} d_u\right)$

$= O(|E|)$.

TOTAL RUN TIME

$= O(|V| + |E|)$



```
DFST(G(V,E)):

for i=1 to n do VISITED[u]=0;
for i:=1 to n do
    if !VISITED(i) then
        DFS(i);
```

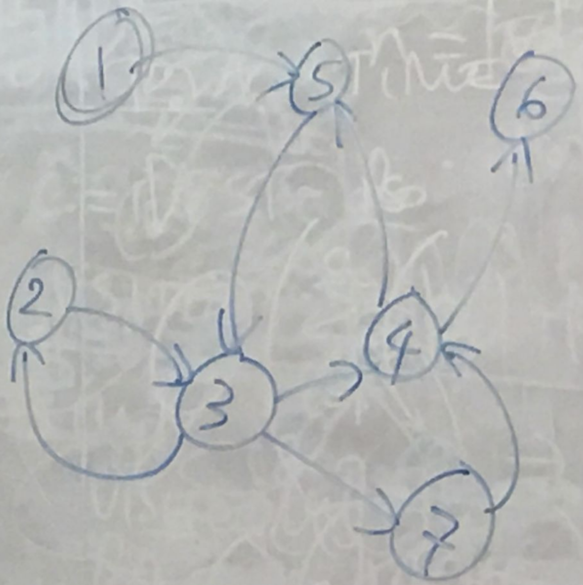
A CONNECTED COMPONENT IS A MAXIMAL CONNECTED SUBGRAPH.

REPRESENTING A GRAPH:

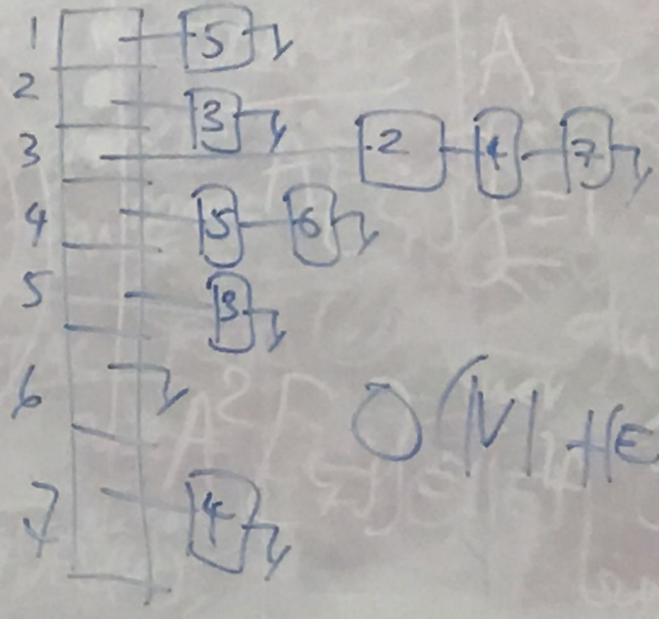
$G(V,E)$

① ADJACENCY LISTS

② ADJACENCY MATRIX.



ADJACENCY LISTS:



$O(|V|+|E|)$

ADJACENCY MATRIX

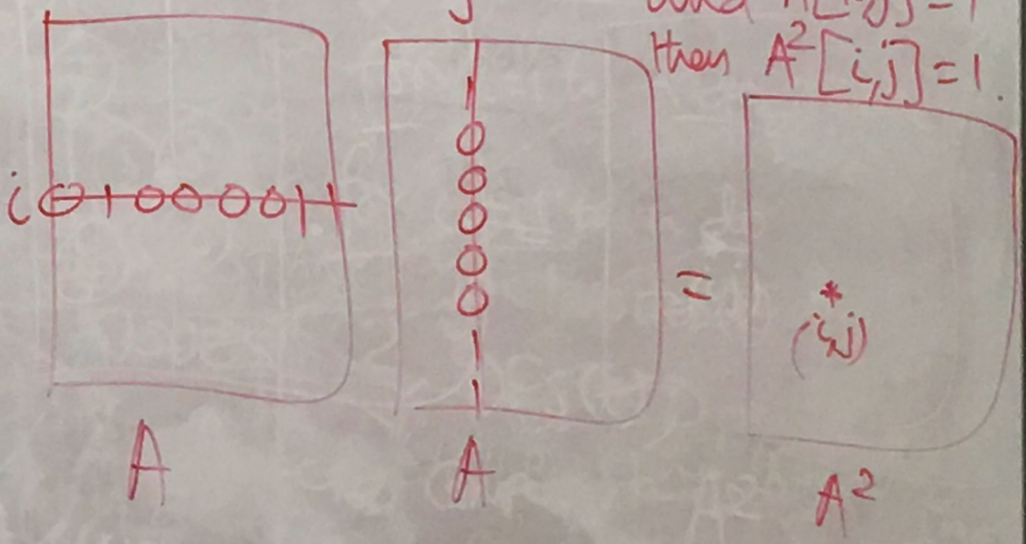
$n = |V|$.

$$A = \begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$

$\xleftrightarrow{\;n\;}$ $\quad$ $n \updownarrow$

$O(|V|^2)$

if $\exists k$ s.t.
$A[i,k] = 1$
and $A[k,j] = 1$
then $A^2[i,j] = 1$.



$A^k[i,j] = 1$ if $\exists$ a path of length k from i to j

$\forall k$.

$$A^* = I + A + A^2 + \cdots + A^{n-1}$$

FACT: $I + A + A^2 + \cdots + A^{n-1}$

$$= \boxed{(I+A)}^{n-1}.$$

COMPUTING a^n.

Consider the case $n = 2^q$ for some integer q.

$a, a^2, a^4, a^8, a^{16}, a^{32}, \ldots$

We only need q Mult. and $q = O(\log n$

$12 = 1100$

Let $n = \sum_{i=0}^{q} b_i 2^i$

$$a^n = a^{\sum_{i=0}^{q} b_i 2^i} = a^{b_0} \cdot a^{b_1 \cdot 2} a^{b_2 2^2} \cdots a^{b_q 2^q}$$

$$= \prod_{i=0}^{q} a^{b_i 2^i}$$

(1) Compute $a, a^2, a^4, \ldots, a^{2^q} \leftarrow O(q)$

(2) Use the above formula $\leftarrow O(q)$

$\Big\}$ TOTAL $= O(\log n)$.

PARALLEL ALGORITHMS.

Let π be any problem.

Let P be the # of PROCESSORS.

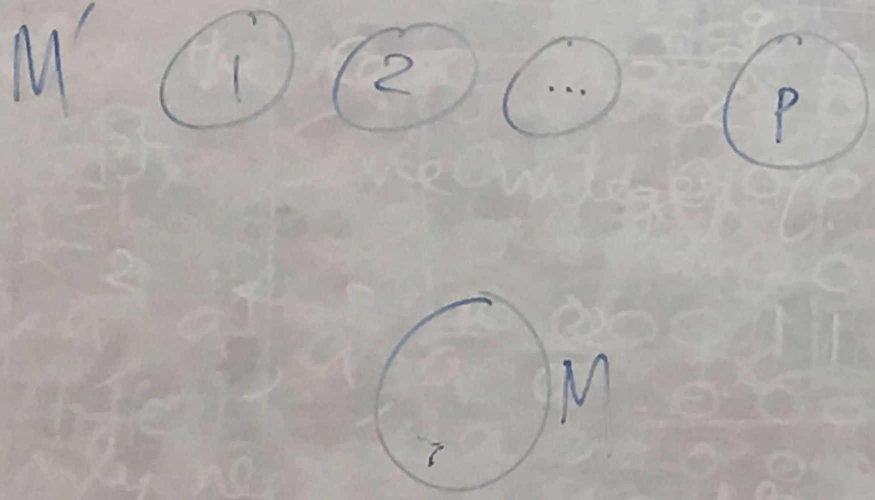
Let T be the PARALLEL RUN TIME and S be the BEST KNOWN SEQUENTIAL RUN TIME.

Example. $S = 10h$.

$P = 10$.

FACT: $T \geqslant \frac{S}{P}$.

PROOF BY CONTRADICTION:



ASSUME that $T < \frac{S}{p}$.

ONE $\|^{l}$ step can be sequentially SIMULATED IN $\leq p$ steps.

$\Rightarrow$ The entire $\|^{l}$ alg. can be sequentially simulated in $\leq pT$ steps.

i.e. in LESS than S steps.

$\Rightarrow$ A CONTRADICTION. □

CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 20: November 3, 2016

Tree Traversal and Graph Search

- Traversal and search refer to the systematic visiting of the nodes of a tree or a graph, performing certain operations at each node.

- In the last lecture we showed that we can perform tree traversal in $O(n)$ time, n being the number of nodes in the tree.

- In this lecture we will focus on searching through a general graph.

- Let $G(V, E)$ be a given undirected graph that we are interested in searching. There are several ways of searching. Two popular methods are Breadth-First Search (BFS) and Depth-First Search (DFS).

- Recall that G can be represented as adjacency lists or an adjacency matrix. Let $V = \{1, 2, \ldots, n\}$.

  The adjacency lists representation of G is an array $A[1:n]$ of lists. $A[i]$ is a list of all the neighbors of the node i, $1 \leq i \leq n$.

  The adjacency matrix representation of G is a $n \times n$ matrix A such that $A[i, j] = 1$ if there is an edge from the node i to node j in G; and $A[i, j] = 0$ otherwise.

- BFS starts from a node, say, u. The node u is visited first. Nodes that are at a distance of 1 from u are visited next; Nodes that are at a distance of 2 from u are visited next; and so on.

Depth First Search (DFS)

- In DFS we start from a node, say, u and visit a neibhor v of u that has not been visited before; From v we visit a neighbor w of v that has not been visited before, and so on, until we reach a node x such that all the neighbors of x have already been visited. When this happens we backtrack to the node y that was visited before x and start the search from y, etc. The search terminates when we backtrack to the start node u.

- A pseudocode for DFS follows. To begin with, each entry in the array $visited[1:n]$ is zero.

```
DFS($u$)
    $visited[u] = 1$;
    **for** each $w \in Adj(u)$ **do**
```

```
3)        if !$visited[w]$ then DFS($w$);
```

Run Time Analysis: Note that, for any node $u \in V$, line 3 is executed d_u times where d_u is the degree of u. Lines 1 and 2 are executed for every node u in V. Thus the run time of this algorithm is $O(|V| + \sum_{u \in V} d_u) = O(|V| + |E|)$. This is a linear time algorithm.

The case of multiple components

- The input graph may not be connected. Recall that an undirected graph is said to be connected if there is a path from every node to every other node in the graph.
- If the graph is not connected, then it has more than one *connected components*. A connected component of a graph is a maximal subgraph of the graph that is connected.
- When the input graph has more than one connected components we can modify the algorithm DFS to get the following algorithm DFST:

```
for $i = 1$ to $n$ do
    $visited[i] = 0$;
DFST($G(V, E)$)
    for $i = 1$ to $n$ do
        if !$visited[i]$ then DFS($i$);
```

- **Run Time:** When DFS is called on any node i, all the nodes in the connected component that i belongs to will be visited. Let the number of connected components in G be c. Let the number of nodes and edges in connected component q be $|V_q|$ and $|E_q|$, respectively, for $1 \leq q \leq c$. If the node i belongs to connected component q, then, the time spent by DFS(i) will be $O(|V_q| + |E_q|)$.

  Thus the total run time of the algorithm will be $O\left(\sum_{q=1}^{c}(|V_q| + |E_q|)\right) = O(|V| + |E|)$.

Hints on Problem 7 in Homework 2

- Note that the adjacency matrix A has information about paths of length 1 in the graph. Specifically, $A[i, j] = 1$ iff there is an edge from the node i to node j.
- Now consider the matrix A^2. $A^2[i, j]$ will be 1 only if there is a k such that $A[i, k] = 1$ and $A[k, j] = 1$, i.e., if there is a path from i to j of length 2.
- Similarly, we can prove by induction that $A^k[i, j] = 1$ only if there is a path from node i to node j of length k.

- Therefore, it follows that $A^* = I + A + A^2 + \cdots + A^{n-1}$.

- Using the binomial theorem, we can show that $I + A + A^2 + \cdots + A^{n-1} = (I + A)^{n-1}$.

- Let a be a real number and n be an integer. We can compute a^n using $n-1$ multiplications.

- In fact we can compute a^n using only $O(\log n)$ multiplications. Consider the case when $n = 2^q$ for some integer q. Then we can repeatedly square elements starting from a to get the following sequence: $a, a^2, a^4, \ldots, a^{2^q}$. Clearly, the computation of a^{2^q} takes only $O(q) = O(\log n)$ multiplications.

- Even when n is not an integral power of two we can compute a^n using $O(\log n)$ multiplications. Express n in binary form as: $n = \sum_{i=0}^{q} b_i 2^i$ where each b_i is a bit. Note that $q = O(\log n)$.
$$a^n = a^{\sum_{i=0}^{q} b_i 2^i} = \Pi_{(0 \le i \le q) \text{ and } b_i = 1} a^{2^i}.$$

- The above equation suggests the following algorithm: 1) Compute the sequence: $a, a^2, \ldots, a^{2^q}$ in $O(q)$ time; and 2) Multiply the appropriate powers of a from the above sequence. This takes $O(q)$ time as well.

  The total run time is $O(q) = O \log n)$.

Parallel Algorithms

- The idea of parallel computing is to employ multiple processors to solve a problem.

- Let π be any problem for which the best known sequential algorithm takes S time. Let P be the number of processors used and let T be the parallel run time.

Fact: $T \geq \frac{S}{P}$.

Proof: by contradiction. Assume to the contrary that there is a parallel algorithm that takes $< \frac{S}{P}$ time.

We can sequentially simulate each step of the parallel algorithm in $\leq P$ steps. This means that we can sequentially simulate the entire parallel algorithm in a total of $\leq PT < S$ time! This is a contradiction to the fact that S is the best known sequential run time for solving π. □