CS2351
Data Structures

Lecture 10:
Graph and Tree Traversals I
About this lecture

• We introduce two popular algorithms to traverse a graph
  1. Breadth First Search (BFS)
  2. Depth First Search (DFS)
    • DFS Tree and DFS Forest
    • Parenthesis theorem
Breadth First Search
Lost in a Desert

• After an unfortunate accident, we survived, but are lost in a desert
• To keep surviving, we need to find water
• How to find the closest water source?
Breadth First Search (BFS)

- A simple algorithm to find all vertices reachable from a particular vertex \( s \)
  - \( s \) is called **source** vertex

- **Idea**: Explore vertices in rounds
  - At Round \( k \), visit all vertices whose shortest distance (#edges) from \( s \) is \( k-1 \)
  - Also, discover all vertices whose shortest distance from \( s \) is \( k \)
The BFS Algorithm

1. Mark $s$ as discovered in Round 0

2. For Round $k = 1, 2, 3, \ldots$,
   
   For (each $u$ discovered in Round $k-1$)
   
   { Mark $u$ as visited;
     Visit each neighbor $v$ of $u$;
     If ($v$ not visited and not discovered)
     Mark $v$ as discovered in Round $k$;
   }

Stop if no vertices were discovered in Round $k-1$
Example ($s = \text{source}$)

visited ($?$ = discover time)

discovered ($?$ = discover time)

direction of edge when new node is discovered
Example \((s = \text{source})\)

1. The graph starts with the source node \(s\) marked as visited with a discover time of 0.
2. The first node to be visited is \(v\), followed by \(w\) and \(x\), each with their own discover times.
3. The direction of the edge is indicated when a new node is discovered, showing the order of traversal.
4. The graph is updated as nodes are discovered, maintaining the discover times and visited status.

- \(v\) visited, discover time \(0\)
- \(w\) discovered, discover time \(1\)
- \(x\) discovered, discover time \(2\)

The arrows indicate the direction of the edge when a new node is discovered.
Example ($s = \text{source}$)

- **Visited** ($? = \text{discover time}$)
- **Discovered** ($? = \text{discover time}$)
- **Direction of edge when new node is discovered**
Example \((s = \text{source})\)

The directed edges form a tree that contains all nodes \(\text{reachable from } s\) called \(\text{BFS tree of } s\).

Done when no new node is discovered.
Correctness

• The correctness of BFS follows from the following theorem:

Theorem: A vertex $v$ is discovered in Round $k$ if and only if shortest distance of $v$ from source $s$ is $k$

Proof: By induction
Performance

• BFS algorithm is easily done if we use
  • an $O(|V|)$-size array to store discovered/visited information
  • a separate list for each round to store the vertices discovered in that round
  • Since no vertex is discovered twice, and each edge is visited at most twice (why?)

  ➔ Total time: $O(|V|+|E|)$
  ➔ Total space: $O(|V|+|E|)$
Performance (2)

• Instead of using a separate list for each round, we can use a common queue
  • When a vertex is discovered, we put it at the end of the queue
  • To pick a vertex to visit in Step 2, we pick the one at the front of the queue
  • Done when no vertex is in the queue

→ No improvement in time/space ...
→ But algorithm is simplified

Question: Can you prove the correctness of using queue?
Depth First Search
Depth First Search (DFS)

• An alternative algorithm to find all vertices reachable from a particular source vertex $s$

• Idea:
  Explore a branch as far as possible before exploring another branch

• Easily done by recursion or stack
The DFS Algorithm

DFS(u)
{
    Mark u as discovered;
    while (u has unvisited neighbor v)
        DFS(v);
    Mark u as finished;
}

The while-loop explores a branch as far as possible before the next branch.
Example \((s = \text{source})\)

- The process begins with the source node \(s\) being marked as discovered.
- As new nodes are discovered, their directions of edges are marked.
- Nodes that have been visited before are marked as finished.

Directions:
- Red arrows represent the direction of the edge when a new node is discovered.
Example ($s = \text{source}$)

1. The process starts with node $s$ being marked as discovered.
2. As new nodes are discovered, their edges are directed towards them.
3. Once all nodes are discovered, they are marked as finished.

The direction of the edge changes when a new node is discovered, reflecting the traversal order in a graph search algorithm.
Example \( (s = \text{source}) \)

\[
\begin{array}{c}
\text{r} & \text{s} & \hat{t} & \text{u} \\
\text{v} & \text{w} & \text{x} & \text{y} \\
\text{r} & \text{s} & \hat{t} & \text{u} \\
\text{v} & \text{w} & \text{x} & \text{y}
\end{array}
\]

\[
\begin{array}{c}
\text{r} & \text{s} & \hat{t} & \text{u} \\
\text{v} & \text{w} & \text{x} & \text{y} \\
\text{r} & \text{s} & \hat{t} & \text{u} \\
\text{v} & \text{w} & \text{x} & \text{y}
\end{array}
\]

- finished
- discovered
- direction of edge when new node is discovered
Example \((s = \text{source})\)

![Diagram of a graph with nodes and edges, illustrating the concept of discovered and finished nodes.](image)

- **Finished**: Nodes marked with a red circle.
- **Discovered**: Nodes marked with a pink circle.
- **Direction of edge when new node is discovered**: Arrows indicating the direction.

The diagram shows the process of traversing a graph, discovering new nodes, and marking them as finished.
Example \((s = \text{source})\)

When a new node is discovered, its direction of edge is reversed.
Example (s = source)

Done when s is discovered

The directed edges form a tree that contains all nodes reachable from s

Called DFS tree of s
Generalization

• Just like BFS, DFS may not visit all the vertices of the input graph $G$, because:
  • $G$ may be disconnected
  • $G$ may be directed, and there is no directed path from $s$ to some vertex

• In most application of DFS (as a subroutine), once DFS tree of $s$ is obtained, we will continue to apply DFS algorithm on any unvisited vertices ...
Suppose the input graph is directed.

### Generalization (Example)
Generalization (Example)

1. After applying DFS on s
2. Then, after applying DFS on $t$
3. Then, after applying DFS on y
Generalization (Example)

4. Then, after applying DFS on r
Generalization (Example)

5. Then, after applying DFS on v
Generalization (Example)

Result: a collection of rooted trees called DFS forest

![Diagram of rooted trees]

- **r**
- **s**
- **t**
- **u**
- **v**
- **w**
- **x**
- **y**
Performance

• Since no vertex is discovered twice, and each edge is visited at most twice (why?)
  ➔ Total time: $O(|V| + |E|)$

• As mentioned, apart from recursion, we can also perform DFS using a LIFO stack (Do you know how?)
Discovery and Finishing Times

- When the DFS algorithm is run, let us consider a global time such that the time increases one unit:
  - when a node is discovered, or
  - when a node is finished
    (i.e., finished exploring all unvisited neighbors)

- Each node $u$ records:
  - $d(u) = \text{the time when } u \text{ is discovered}$, and
  - $f(u) = \text{the time when } u \text{ is finished}$
Discovery and Finishing Times

In our first example (undirected graph)
Discovery and Finishing Times

In our second example (directed graph)
Nice Properties

Lemma: For any node $u$, $d(u) < f(u)$

Lemma: For nodes $u$ and $v$,

$d(u), d(v), f(u), f(v)$ are all distinct

Theorem (Parenthesis Theorem):
Let $u$ and $v$ be two nodes with $d(u) < d(v)$.
Then, either

1. $d(u) < d(v) < f(v) < f(u)$ [contain], or
2. $d(u) < f(u) < d(v) < f(v)$ [disjoint]
Proof of Parenthesis Theorem

- Consider the time when $v$ is discovered.
- Since $u$ is discovered before $v$, there are two cases concerning the status of $u$:
  - **Case 1**: ($u$ is not finished)
    This implies $v$ is a descendant of $u$
    $\Rightarrow f(v) < f(u)$ (why?)
  - **Case 2**: ($u$ is finished)
    $\Rightarrow f(u) < d(v)$
Corollary:

\[ v \text{ is a (proper) descendant of } u \]

if and only if

\[ d(u) < d(v) < f(v) < f(u) \]

Proof:

\[ v \text{ is a (proper) descendant of } u \]

\[ \Leftrightarrow d(u) < d(v) \text{ and } f(v) < f(u) \]

\[ \Leftrightarrow d(u) < d(v) < f(v) < f(u) \]