Graphs II

Lecture 27

Prof. Eric Polizzi
Summary Previous Lecture

- Composed of vertices (nodes) and edges

- Directed vs non-directed, weighted vs unweighted
Summary Previous Lecture

- Matrix representation

- Graph operations: two main Graph Traversal algorithms (search algorithms)
  - 1- Depth-First Search (DFS)- using a stack
  - 2- Breadth-First Search (BFS)- using a queue
DFS: Example

Visit: ABCDEF
Graph Operations- Breadth-First Search (BFS)

- **DFS**: The algorithm 'wants' to get as far away as possible from the starting point (until it reaches a dead end)
- **BFS**: the algorithm 'likes' to stay as close as possible from the starting point
  - Start with a starting point, for example A
  - Visit this current vertex, mark it, and follow these simple rules:
    - **Rule 1**: visit the next adjacent unvisited vertex, mark it, and insert it in the queue, come back to current vertex
    - **Rule 2**: If you can't follow rule 1, then if possible dequeue a vertex from the queue to become your current vertex
    - **Rule 3**: If you can't follow rules 1 and 2 (queue is empty), then you are done
Graph Operations- Breadth-First Search (BFS)

- Visit A
- Visit B: Queue: B
- Visit C: Queue: BC
- Visit D: Queue: BCD
- Visit E: Queue: BCDE
- Dequeue: Queue: CDE
- Visit F: Queue: CDEF
- Dequeue: Queue: DEF
- Dequeue: Queue: EF
- Visit G: Queue: EFG
- Dequeue: Queue: FG
- Dequeue: Queue: G
- Visit H: Queue: GH
- Dequeue: Queue: H
- Visit I: Queue: HI
- Dequeue: Queue: I
- Dequeue: Queue: (empty)
- Done

Visit: ABCDEFGHI

Analogy: Find all vertices that are 1 edge away from start, then 2 edges away and so on.
BFS: Example

Visit: ABCEFD
Todo: Java applet GraphN.html

```java
public void bfs(){
    nodes[0].visited=true;
    nodes[0].display();
    theQueue.enqueue(0);
    int v2;
    while(!theQueue.isEmpty())
    {
        int v1=theQueue.dequeue();
        while((v2=getAdjUnvisitedNode(v1))!=-1)
        {
            nodes[v2].visited=true;
            nodes[v2].display();
            theQueue.enqueue(v2);
        }
    }
    // Queue is empty so we can reset the flags
    for(int i=0; i<N;i++) nodes[i].visited=false;
}

public int getAdjUnvisitedNode(int v){
    for(int i=0; i<N;i++)
        if (mat[v][i]==1 && nodes[i].visited=false)
            return i; // found neighbor
    return -1; // no such node
}
```
Minimum Spanning Trees (MST)

- **Spanning Trees?**
  - Connected subgraph that contains all N vertices of the original graph
  - It contains only E=N-1 edges (there is no cycle)
  - Not unique

- Algorithms to create a MST is almost the same as searching
- One can use either DFS or BFS, but we need to record the edges
Minimum Spanning Trees (MST)

- MST is easy to derived from DFS, since it visits all the nodes once.
- Different starting vertex leads to different MST
- From DFS, we can modify the algorithm as follows

```java
public void dfs(){
    nodes[0].visited=true;
    nodes[0].display();
    theStack.push(0);

    while(!theStack.isEmpty()){
        int currentNode=theStack.peek(); // save current node
        int v=getAdjUnvisitedNode(currentNode);
        if (v ==-1) // no such node
            theStack.pop();
        else {
            nodes[v].visited=true;
            theStack.push(v);
            System.out.println('From to');
            nodes[currentNode].display(); // from current
            nodes[v].display(); // to
        }
    }
    // stack is empty so we can reset the flags
    for(int i=0; i<N;i++) nodes[i].visited=false;
}
```

- Todo: Java applet
  GraphN.html
Minimum Spanning Trees (MST) - Example

**Graph Representation:**
- **Vertices:** ALB, BDL, LGA, JFK, EWR, HPN, PVD, BOS, MHT, BTV, SWF
- **Edges and Weights:**
  - ALB: 0 0 1 1 0 0 0 0 1 1
  - BDL: 0 0 0 0 1 0 0 1 0 1 0
  - LGA: 1 0 0 0 1 0 0 0 1 1
  - JFK: 1 0 0 0 0 1 1 0 0 0 0
  - EWR: 0 1 1 0 0 0 0 0 1 0 0
  - HPN: 0 0 0 1 0 0 0 1 0 0 0
  - PVD: 0 0 0 1 0 0 0 1 1 0
  - BOS: 0 1 0 0 0 1 0 0 1 0 1
  - MHT: 0 0 0 0 1 0 1 1 0 1 0
  - BTV: 1 1 1 0 0 0 1 0 1 0 0
  - SWF: 1 0 1 0 0 0 0 1 0 0 0
Items or events must be arranged (traversed) in a specific order

Example: course prerequisites

Obtaining your degree (last item H on the list) may look like: BAEDGCFH

Arranged this way the graph is said to be 'topologically sorted'
Topological Sorting using Directed Graphs

- Topological sorting algorithm involved those steps:
  - 1- Find a vertex that has no successors (the successors to a vertex are those vertices that are directly downstream from it connected by a directed edge)
  - 2- Delete this vertex from the graph, and insert its label at the beginning of the list (insertFirst in LL)
  - 3- Repeat steps 1 and 2 until all vertices are gone

- Remarks:
  - It will not work for cycles (A → B → C → A)
  - We must consider a directed graph with no cycle

- Complete Code p657 textbook
  - Todo: Java applet GraphD.html