lecture27: Graph Traversals

Sean Massung

Largely based on slides by Cinda Heeren
CS 225 UIUC

25th July, 2013
Announcements

- mp7.1 extra credit due tomorrow night (7/26)
- Code challenge tomorrow night (7/26) at 6pm in 0224
- lab_hash due Saturday night (7/27)
- This is our last Thursday class
Traversals

- We know how to do traversals on trees, but now we want to traverse a graph.
- That is, we want to visit every vertex and every edge in the graph, *honoring the connectivity of the graph*.
- Compared to tree traversals...
  - there is no obvious start point
  - there is no obvious order
  - there is no obvious way to denote progress... how can we make sure we aren’t traversing in circles forever??
A level order traversal

```cpp
template <class T>
void BinaryTree<T>::traverse() const
{
    Queue<TreeNode*> q;
    q.enqueue(root);
    while(!q.empty())
    {
        TreeNode* cur = q.dequeue();
        if(cur != NULL)
        {
            yell(cur->data);
            q.enqueue(cur->left);
            q.enqueue(cur->right);
        }
    }
}
```

Prints:
```
+ - 6 + / 4 * 5 + 3 7 3 4
```
A level-order traversal Breadth-First Search

```
/* BFS */

template <class T>
void BinaryTree<T>::traverse() const
{
    Queue<TreeNode*> q;
    q.enqueue(root);
    while(!q.empty())
    {
        TreeNode* cur = q.dequeue();
        if(cur != NULL)
        {
            yell(cur->data);
            q.enqueue(cur->left);
            q.enqueue(cur->right);
        }
    }
}
```

First, explore all nodes at a distance one away, then at a distance two, then, three.....
BFS

- From the start point, expands outwards like a ripple in a pond
- Every time we see an unvisited node, this is the shortest distance from the start to that node (by number of edges)
- BFS is written most intuitively in an iterative fashion using a Queue as an ordering structure
- BFS can use the labels discovery and cross to label edges
BFS pseudocode

1. Enqueue the start node
2. Dequeue a node and enqueue any adjacent nodes that have not yet been visited
3. If the queue is empty, every node in the graph has been examined; stop
4. Otherwise, if the queue is not empty, repeat from step 2
BFS example
BFS implementation

Assume we are given the graph and some start vertex.

```cpp
void BFS(Graph & graph, const Vertex & start) {
    queue<Vertex> q;
    q.push(start);
    graph.setVertexLabel(start, "VISITED");

    while(!q.empty())
    {
        // traverse!
    }
}
```

How could we modify this to search for a specific vertex, or to work on a graph that isn’t connected?
BFS implementation: the traversal

```cpp
while(!q.empty())
{
    Vertex v = q.front();
    q.pop();

    vector<Vertex> adj = graph.getAdjacent(v);
    for(size_t i = 0; i < adj.size(); ++i)
    {
        if(graph.getVertexLabel(adj[i]) == "UNEXPLORED")
        {
        }

        else if(graph.getEdgeLabel(v, adj[i]) == "UNEXPLORED")
        {
        }
    }
}
```
BFS analysis

- What is the running time of BFS in terms of $n$ and $m$?
- ...does it change based on our graph implementation?
- How can we use BFS to make a spanning tree?
- How can we use BFS to detect cycles?
From the start point, DFS explores as far as possible along each branch, then starts *backtracking*.

DFS is written intuitively in *either* an iterative fashion (very similar to BFS) or recursively.

DFS can use the labels *discovery* and *back* to label edges.
DFS pseudocode (using a stack)

1. Push the start node
2. Pop a node and push any adjacent nodes that have not yet been visited
3. If the stack is empty, every node in the graph has been examined; stop
4. Otherwise, if the stack is not empty, repeat from step 2
DFS example
DFS implementation (iterative)

Assume we are given the graph and some start vertex.

```cpp
void DFS(Graph & graph, const Vertex & start) {
    stack<Vertex> s;
    s.push(start);
    graph.setVertexLabel(start, "VISITED");

    while(!s.empty())
    {
        // traverse!
    }
}
```

How could we modify this to search for a specific vertex, or to work on a graph that isn’t connected?
DFS implementation (iterative): the traversal

```cpp
while (!s.empty())
{
    Vertex v = s.top();
    s.pop();

    vector<Vertex> adj = graph.getAdjacent(v);
    for (size_t i = 0; i < adj.size(); ++i)
    {
        if (graph.getVertexLabel(adj[i]) == "UNEXPLORED")
        {
        }
        else if (graph.getEdgeLabel(v, adj[i]) == "UNEXPLORED")
        {
        }
    }
}
```
DFS implementation (recursive)

```cpp
void DFS(Graph & graph, Vertex cur) {
    graph.setVertexLabel(cur, "VISITED");

    vector<Vertex> adj = graph.getAdjacent(cur);
    for(size_t i = 0; i < adj.size(); ++i) {
        if(graph.getVertexLabel(adj[i]) == "UNEXPLORED") {
        }
        else if(graph.getEdgeLabel(cur, adj[i]) == "UNEXPLORED") {
        }
    }
}
```
DFS analysis

- What is the running time of DFS in terms of $n$ and $m$?
- ...does it change based on our graph implementation?
- How can we use DFS to make a spanning tree?
- How can we use DFS to detect cycles?
Bipartite graphs
Bipartite graph test