Recitation 7 – *BFS, Graphs and Exam I Debrief*

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2014)

February 25th, 2014

1 Announcements

- How did the exam go?
- *ThesaurusLab* has been released!
- Questions from lecture?

2 Graphs

A graph G is a tuple of sets $G = (V, E)$, where

V is a set of vertices (also known as nodes).

$E \subseteq \binom{V}{2}$ is a set of edges.

3 Graph Representations

1. **Edge Set**

   A graph is represented by a set of edges. An edge $e = (x, y)$ is denoted by its endpoints. Notice that in an undirected graph, (x, y) represents the same edge as (y, x). This representation doesn't provide an efficient means to access the neighbors of a vertex.

2. **Adjacency Table**

   A graph is represented by a table with keys for all $v \in V$. $v \mapsto S$, where S is a set and any vertex $u \in S \iff (v, u) \in E$.

3. **Array Sequences**

   This approach is slightly less general because it requires that the vertices be numbered from 0 to $n - 1$. For a given vertex v with number i, the ith index of the sequence will contain the S that v would map to in the adjacency table representation.

We covered in class the costs of doing various operations with an adjacency table. What would be the cost of doing them with array sequences?

	work	*span*
`isEdge`$(G,(u,v))$		
`map over all edges`		
`map over neighbors of` v		
$d_G^+(v)$		

Solution 3.0

	adj table		**adj seq**	
	work	*span*	*work*	*span*
`isEdge`$(G,(u,v))$	$O(\log n)$	$O(\log n)$	$O(d_G^+(u))$	$O(\log d_G^+(u))$
`map over all edges`	$O(m)$	$O(\log n)$	$O(m)$	$O(1)$
`map over neighbors of` v	$O(\log n + d_G^+(v))$	$O(\log n)$	$O(d_G^+(v))$	$O(1)$
$d_G^+(v)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$

4 Friends of Friends

Suppose we have just founded a new social networking site and naturally we have represented the network of friends as a graph using the adjacency table representation.

Suppose we want to find friends of our friends. These would be the neighbors of our neighbors, or all vertices that have a distance of 2 from a source vertex v. We will need to use the `TABLE` function:

```
val tabulate : (key -> 'a) -> set -> table
```

Write the function `FoF` to solve this problem.

```
1  type graph = Set.set Table.table
2  fun FoF (G : graph) v =
3  let
4
5
6
7
8
9  in
10
11
12 end
```

Solution 4.0

```
1  type graph = Set.set Table.table

2  fun FoF (G : graph) v =
3  let
4   fun neighbors v = Table.find G v
5   val Ngh = neighbors v
6   val NghNgh = Table.tabulate neighbors Ngh
7  in
8   Set.filter (fn x => x <> v) (Table.reduce Table.Set.union {} NghNgh)
9  end
```

We find the neighbors of v and the neighbors' neighbors, which now is a table mapping each neighbor to its neighbors. Then we flatten these sets of neighbors using `Set.union` so that duplicates are removed. The `filter` ensures that the original vertex is not included in the output.

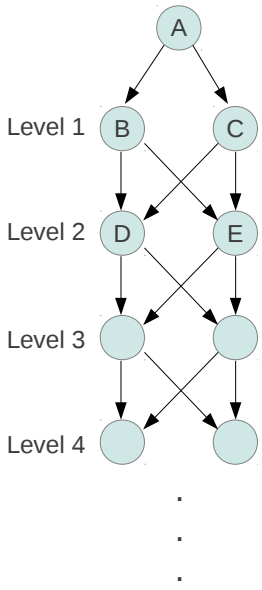
5 Parallelism in BFS?

Q: Can BFS recursive calls be made in parallel?

A: When calls are made in parallel, visited sets are all independent among each parallel call now, and we can end up visiting the same node multiple times.

Q: What are some examples of graphs where parallel BFS would visit nodes multiple times?

A: Consider this graph:



What's wrong here? Also consider the even worse case of a fully-connected graph.

Q: How many times will a node be visited in each graph?

A: Let $V(i)$ be the number of times a node in level i is visited. Then $V(i) = 2V(i-1)$, with $V(1) = 1$. So, we get $V(i) = 2^i$.

After starting at any node, we will end up generating every possible permutation of the rest of the vertices as a path that we search in parallel, giving $O(n^{n-1})$ visited nodes total.

6 Coding BFS

Write a function to find the *diameter* of a graph using *breadth-first* traversal. Recall that the diameter of a graph is the length of the longest distance between two vertices in the graph.

To get started, first consider: how does this relate to BFS? What would be a good graph representation to use?

You may also want to write a helper function to get the neighbors of a set of vertices.

Solution 6.0

```
fun N (G : graph) (F : Set.set) =
    Set.difference (Table.reduce Set.union Set.empty (Table.extract (G, F)), F)

fun longestDist G s =
  let
    fun iDist (visited, F, i) =
      if Set.size F = 0 then i-1
      else
        let
          visited' = Set.union (visited, F)
          F' = Set.difference (N (G) (F), X)
        in
          iDist(visited', F', i+1)
        end
  in
    iDist (Set.empty(), Set.singleton s, 0)
  end

fun diameter (G : graph) =
    let
      val dists = Seq.map (longestDist G) (domain G)
    in
      Seq.reduce Int.max 0 dists
    end
```