CSE341T/CSE549T **11/03/2014**

Parallel Breadth First Search

Lecture 18

Today, we will look at a basic graph algorithm, breadth first search (BFS). BFS can be applied to solve a variety of problems including: finding all the vertices reachable from a vertex v, finding if an undirected graph is connected, finding the shortest path from a vertex v to all other vertices, determining if a graph is bipartite, bounding the diameter of an undirected graph, partitioning graphs, and as a subroutine for finding the maximum flow in a flow network (using Ford-Fulkerson's algorithm).

1 BFS

BFS, as with the other graph searches, can be applied to both directed and undirected graph. In the following discussion the *distance* $\delta_G(u, v)$ from a vertex u to a vertex v in a graph G is the shortest path (minimum number of edges) from u to v. The idea of *breadth first search* is to start at a *source* vertex s and explore the graph level by level, first visiting all vertices that are the (out) neighbors of s (i.e. have distance 1 from s), then vertices that have distance two from s, then distance three, etc. It should be clear that a vertex at distance $i + 1$ must have an in-neighbor from a vertex a distance i. Therefore if we know all vertices at distance i, then we can find the vertices at distance $i + 1$ by just considering their out-neighbors.

The BFS approach therefore works as follows. As with all the search approaches, the approach needs to keep track of what vertices have already been visited so that it does not visit them more than once. Let's call the set of all visited vertices at the end of step i, X_i. On each step the search also needs to keep the set of new vertices that are exactly distance i from v. We refer to these as the *frontier* vertices $F_i \subset X_i$. To generate the next set of frontier vertices the search simply takes the neighborhood of F and removes any vertices that have already been visited, *i.e.*, $N_G(F) \setminus X$. Recall that for a vertex u, $N_G(u)$ are the neighbors of u in the graph G (the out-neighbors for a directed graph) and for a set of vertices U, that $N_G(U) = \cup_{u \in U} N_G(u)$. This gives us the following code where F is the set of frontier vertices for iteration i and X is the set of vertices that has already been processed.

BFS(G, X, F, i)

1 **if** $|F| = 0$
2 **then return**
3 $F' \leftarrow N_G(F) \backslash X$
4 label all vertices in F' as $i + 1$
5 $X' \leftarrow X \cup F'$
6 BFS$(G, X', F', i + 1)$

The original call is with $X = \{s\}$, $F = \{s\}$ and $i = 0$. It might be easier to look at the iterative version of this algorithm.

BFS(G, s)

1 $X \leftarrow \{s\}$
2 $F \leftarrow \{s\}$
3 $i \leftarrow 0$
4 **while** $F \neq \Phi$
5 **do** $F' \leftarrow N_G(F) \backslash X$
6 label all vertices in F' as $i + 1$
7 $X \leftarrow X \cup F'$
8 $F \leftarrow F'$
9 $i \leftarrow i + 1$

To prove that the algorithm is correct we need to prove the assumptions that are stated in the algorithm. In particular:

Lemma 1 *In algorithm BFS when calling BFS*(G, X, F, i)*, we have* $X = \{u \in V_G : \delta_G(v, u) \leq i\} \wedge F = \{u \in V_G : \delta_G(v, u) = i\}$

Proof. This can be proved by induction on the step i. For the base case (the initial call) we have $X = F = \{s\}$ and $i = 0$. This is true since only s has distance 0 from s. For the inductive step we note that, if all vertices F at step i have distance i from s, then a neighbor of F must have minimum path of length $d \leq i + 1$ from s—since we are adding just one more edge to the path. However, if a neighbor of F has a path $d < i + 1$ then it must be in X, by the inductive hypothesis so it is not added to F'. Therefore F on step $i + 1$ will contain vertices with distance exactly $d = i + 1$ from s. Furthermore since the neighbors of F are unioned with X, X at step $i + 1$ will contain exactly the vertices with distance $d \leq i + 1$. To argue that the algorithm returns all reachable vertices we note that if a vertex u is reachable from u and has distance $d = \delta(s, u)$ then there must be another x vertex with distance $\delta(s, x) = d - 1$. Therefore BFS will not terminate without finding it. Furthermore for any vertex u $\delta(s, u) < |V|$ so the algorithm will terminate in at most $|V|$ steps.

Exercise 1 *So far we have specified a routine that returns the set of vertices reachable from s and the length of all shortest paths to these vertices. Often we would like to know the shortest path itself, not just its length. How would you extend BFS for these purposes?*

The important operations in the above algorithm is

1. calculating $N_G(F) = \bigcup_{u \in F} N_G(u)$,
2. calculating $N_G(F) \backslash X$
3. calculating $X \leftarrow X \cup F'$.

How we calculate these depends on the data structure we use to keep the adjacency information of the graph.

Using Sets: If we keep the neighbors of each vertex using a set data structure, then we can simply use the difference and union operators to do the last two operations. For the first operation, we can do a reduction on the union operation since union of sets is an associative operation.

Using Arrays: We can also simply use sorted adjacency arrays and use merge operations to do the union and get sorted merged arrays. You then have to run a remove duplicates algorithm that uses scan. For the difference, again, one can use a modified merge where you mark each node as coming from either $N_G(F)$ or X and then run a version of remove duplicates.

2 Analysis

The number of iterations of the while loop in the iterative code is the diameter of G, namely d. Now we just need to compute the cost of each iteration. Again, recall, that the important operations are:

1. calculating $N_G(F) = \bigcup_{u \in F} N_G(u)$,
2. calculating $N_G(F) \backslash X$
3. calculating $X \leftarrow X \cup F'$.

In order to calculate the cost of computing $N_G(F)$ we can use the following lemma.

2.1 Reduction lemma

So far we have only analyzed reduce when each operation is a constant time operation. For breadth first search, we need a more general cost for reduction, since computation of $N_G(F)$ is basically a reduction on the union operation. Consider a combine operator f where the cost of combining x and y depends on some measure, generally size, of x and y. Say work of the combine is $W(f(x,y))$ and the span is $S(f(x,y))$.

Lemma 2 *For any combine function f and a monotone size measure s, if for any x, y,*

1. $s(f(x,y)) \leq s(x) + s(y)$ *and*
2. $W(f(x,y)) \leq c_f\,(s(x) + s(y))$ *for some universal constant c_f depending on the function f,*

and say $(\texttt{reduce}\ f\ S)$ is reduction based on operator f of the set S. Then, we have

$$W(\texttt{reduce}\ f\ S) = O\left(\log|S| \sum_{x \in S} s(x)\right).$$

Using the reduction lemma, we can calculate the cost computing $N_G(F)$. The cost of union of two nodes v_1 and v_2 is $|N(v_2)|\lg(|N(v_1)|+|N(v_2)|)$. Note that this function is always $O(|N(v_1)|+|N(v_2)|)$. Therefore, the conditions of the above lemma are satisfied. Therefore, the cost of merging the entire frontier is

$$W = O(\lg|F| \sum_{v \in F} |N(v)| = |N_G(F)|\lg|F|$$

2.2 Without using the reduction lemma

Just to give you some intuition about the reduction lemma, let us analyze the reduction operation from scratch using arrays. We can simply merge the adjacency arrays for all elements in F pairwise in a tree (just as we do a reduction). Cost at every level of the tree is $O(|N_G(F)|)$ and the number of levels is $\lg|F|$, giving us the total cost of $|N_G(F)|\lg|F|$.

2.3 Analysis of Breadth First Search

As we analyzed above, the work of the reduction in each iteration is $O(|N_G(F)|\lg|F|)$. Similarly, the cost of calculating $N_G(F)\backslash X$ is also $O(\min\{|X|, |N_G(F)|\}\log(\frac{|X|+|N_G(F)|}{\max\{|X|,|N_G(F)|\}})$ and the cost of calculating $X \leftarrow X \cup F'$ is $O(\min\{|X|, |F'|\}\log(\frac{|X|+|F'|}{\max\{|X|,|F'|\}})$. The second two quantities is are also $O(|N_G(F)|\lg n)$, since $F' \subseteq N_G(F)$.

The span of each iteration is $O(\lg|F|\lg\max_{u\in F}\{|N_G(u)|\})$ for the reduction. Therefore, we can upper bound it by $\lg^2 n$.

Naive Analysis We can upper bound $N_G(F)$ as m in each iteration giving us a per-iteration work of $m \lg n$ and the total work $O(dm \lg n)$. The total span is $O(d \lg^2 n)$.

Tighter Analysis of Work We can do a tighter analysis of work by noting that each node is in exactly one frontier. Therefore, we can add up the work over the diameter.

$$\begin{aligned} W &= O\left(\sum_{i=[0,\ldots,d]} \log n \sum_{v \in F_i} |N(v)|\right) \\ &= O\left(\log n \sum_{i=[0,\ldots,d]} \sum_{v \in F_i} |N(v)|\right) \\ &= O\left((\log n)|E|\right) \\ &= O(m \log n) \end{aligned}$$

Now we increased the work a little bit — sequential BFS has the work of $O(m)$. One can do better, but you need concurrent writes.