CS330HW5

ras70

October 2017

Problem3A

On first inspection this problem it appears that the solution is choosing the smaller of the two disjoint sets V_1, V_2 of a bipartite graph. In a bipartite graph each vertex in V_1 is adjacent to each vertex in V_2 and vice versa. **However**, a bipartite graph has the constraint that each vertex in V_1 is not adjacent to any node in V_1 (and the same for V_2). This is **not** a necessary constraint for this algorithm because a store can be adjacent to another store.

The algorithm for picking store locations is similar to a simple breadth first search (BFS) traversal. First, a neighborhood is selected as the root. Then during the traversal, each vertex, when visited, is placed in one of two sets depending on its depth. A vertex v has depth d if the minimum path from v to the root contains d edges. If the vertex is at an even depth it is placed in the set of even depth edges. If the vertex is at an odd depth it is placed in the set of odd depth edges.

Once the BFS traversal is complete and all vertices have been visited stores are placed at the neighborhoods in the set, either the even or odd, that contains the least number of neighborhoods. For an explanation why the traversal is guaranteed to visit all vertices and put stores at most $\frac{n}{2}$ neighborhoods is explained in detail in the proof of correctness in *Problem 3b*.

An more formal list of the algorithm steps are given below. Assume that the neighborhoods are in a graph G.

1. Chose a neighborhood r as the root.
2. Let there be two sets S_e and S_o indicating the even and odd depth sets respectively.
3. Begin traversing G starting at r using BFS. When a vertex is add to the queue add it to S_e if it is at an even depth in the traversal and add it to S_o if it is at an odd depth.
4. Upon graph traversal completion return $\min_{\text{el}}(S_e, S_o)$ where $\min_{\text{el}}(A, B)$ returns the set A or B with less elements.

To be completely clear the pseudo-code for this algorithm is given below. To handle the depth component the queue holds a touple of the vertex and depth of

the vertex. When a vertex v is added to the queue it is added with its parents' depth plus one.

```
drop_stores()

    // Initialize an even and odd set of vertices
    evenSet, oddSet

    // Chose the root as any node (could be random or first)
    // Algorithm runs correctly regardless of chosen root
    root = random vertex

    Mark root as visited
    evenSet.add(root)
    # queue stores a touple of (vertex, depth)
    queue.push((root,0))
    WHILE queue is not empty
        Let (x,d) be the head of the queue
        FOR all edges (x, y)
            IF y has not been visited THEN
                queue.push(y,d+1) to the queue
                if (d+1) is even
                    evenSet.add(y)
                else if (d+1) is odd
                    oddSet.add(y)
                Mark y as visited
        Remove x from the queue

    if (eventSet.size < oddSet.size)
        drop stores at vertices in eventSet
    else
        drop stores at vertices in oddSet
```

Running time

The algorithm uses BFS traversal, thus its running time is $\mathbf{O(n + m)}$, where n is the number of vertices and m is the number of edges. The running time of this algorithm is $O(n+m)$ because each vertex n and edge m is processed once*. Here the running time does not deviate from BFS because the additionally actions, such as choosing a root and adding a vertex to a set take constant time.

It is important to note that the algorithm is storing two sets, one for the even and one for the odd edge depths. This results in storing the graph twice. Another approach would be to just store the even **or** odd depth edges and either return this set (if it has less than or equal to $\frac{n}{2}$ edges) or the graph set G minus the set saved. This will result in storing at most $\frac{3}{2}$ times the size of the set. Either way the space complexity is still some multiple of the graph size.

* Each vertex is processed a constant number of times if visited checks are counted as processing

Problem 3b

To prove the overall correctness of the algorithm the algorithm must: 1. place store locations such that each neighborhood contains or is adjacent to a neighborhood that contains a store 2. place at most $n/2$ stores. For the sake of clarity these two proofs will be done separately.

Statement 1: The algorithm places stores a locations such that each neighborhood contains or is adjacent to a neighborhood that contains a store

Proof

Let vertex v be a vertex in the graph that corresponds to a neighborhood that is not chosen at the root. Since the graph is connected* BFS is guaranteed to visit every vertex. Therefore, there must exist some vertex u and edge (u, v). If no vertex satisfied this condition then there would be no path from v to any of the other vertices and the graph would be unconnected, which cannot be the case. Let u be at depth d, where depth corresponds to the depth given in the algorithm [1].

By algorithm design when u is being evaluated and edge (u, v) is selected v is added to the queue with depth $d + 1$. Therefore, each vertex v in the graph, where v is not the root, has depth $d + 1$ and is adjacent to a vertex with depth d.The algorithm places a store at *either* even or odds depths. These two cases will be considered distinctly and it will be shown that the algorithm is correct in either case.

Even

If the algorithm places stores at neighborhoods corresponding to vertices at an even depth and $d+1$ is even then the neighborhood corresponding to vertex v must have a store. If $d + 1$ is odd then d must be even. Therefore, u must correspond to a neighborhood that has a store. In this case, since (u, v) exists, v is adjacent to u and the neighborhood corresponding to v is adjacent to a neighborhood with a store.

Odd

If the algorithm places stores at neighborhoods corresponding to vertices at odd even depth and $d+1$ is odd then the neighborhood corresponding to vertex v must have a store. If $d + 1$ is even then d must be odd. Therefore, u must correspond to a neighborhood that has a store. In this case, since (u, v) exists, v is adjacent to u and the neighborhood corresponding to v is adjacent to a

[1]For greater discussion on what this depth corresponds to and what it means please reference Problem 3A

neighborhood with a store.

Handling the root

Above it has been proven that if a neighborhood is not chosen as the root it will either have a store or be adjacent to a neighborhood that has a store. The argument for the root is similar, but not the same because the root does not have a parent. Rather, the root has children, and a similar argument can be made using this fact. If the root does not have any children then there must only be one neighborhood, because the graph is connected. In this case the algorithm will correctly drop a store in the only neighborhood [2].

If the root does have children then its children must have a depth of 1 by algorithm design. Therefore, either the algorithm places stores at an even depth, in which case the root neighborhood has a store, or the algorithm places stores at an odd depth, in which case the root neighborhoods children have a store. Since the root neighborhood is adjacent to its children the algorithm is also correct in this case.

It has been shown that for any vertex v in the set of neighborhoods it corresponds to a neighborhood that either has a store or is adjacent to a neighborhood that has a store.

Statement 2: At most $n/2$ stores are placed

Proof

Let the notation $|A|$ signify the number of elements or cardinality of set A. Let S_e be the set of even depth vertices and S_o be the set of odd depth vertices respectively. Let the set N be the set of all vertices in the graph (which corresponds to all of the neighborhoods. Since N is the set of all edges in the graph it must have n elements.

Assume towards contradiction that number of elements in the set with less elements is greater than $n/2$. For this to be the case S_e and S_o must have at least $n/2 + 1$ elements. In other words, $|S_e| \geq n/2 + 1$ and $|S_o| \geq n/2 + 1$.

By algorithm design each vertex is add to **either** set S_e or set S_o. In other words, any given vertex can be at an even or odd depth, not both. This stems from the fact that a vertex is not visited more than once, and thus can be only added to the queue once and only given one depth. Also since BFS guarantees each vertex is visited in a connected graph* the union of S_e and S_o must be N, or $S_e \cup S_o = N$. Since S_e and S_o are disjoint sets, as explained above and $S_e \cup S_o = N$ it must follow that,

$$|S_e| + |S_o| = |N|$$

Combining the statement above with the inequalities established above produces the following.

[2] this assumes that $n/2$ actually corresponds to $ceil(n/2)$ and that placing one store when there is one neighborhood does not break the $n/2$ constraint

$$\begin{aligned} |N| &= |S_e| + |S_o| \\ |N| &> n/2 + 1 + n/2 + 1 \\ |N| &> n + 2 \end{aligned}$$

The above work shows that $|N| > n + 2$. However, N, the set of all vertices, must be of size n. Thus, there is a contradiction and the minimum set size cannot be more than $n/2$. Therefore, the minimum set size between S_e and S_o is at most $n/2$.

Since the algorithm places stores at neighborhoods in the set that has the minimum set size between S_e and S_o, at most $n/2$ stores will be placed.

* The graph is connected given the problem statement: "For any pair of neighborhoods, there is always a path connecting them where each step only crosses two neighboring neighborhoods."