Breadth-first search

Breadth-first search (BFS) is a search algorithm used to find all vertices in a graph reachable from a source vertex s. The algorithm proceeds by starting at s and visiting all of its neighbors. That is, vertices connected to s by a single edge. Iteratively, it then visits all the unseen vertices 2 edges away from s, then 3 edges, and so on until all vertices reachable from s have been visited. Thus, the search ends once there are no new vertices to be visited.

BFS has many applications utilized by the modern layman. One of the most powerful uses of BFS is to find the shortest path between two nodes in an unweighted graph. GPS navigation systems represent a city as a graph, where roads are edges and intersections are vertices. BFS is then used to find the shortest path between the traveler's starting location and the destination. Another application lies in the domain of social networks. Facebook can use BFS to suggest people to 'friend' by conducting a breadth-first search on your friend network, using you as the source node. Then, friends are an edge length away, and friends of friends are two edges away. The closer the nodes are to the source, the likelier the person will want to connect with them. A similar application is used in web crawlers, which use BFS to analyze all the websites reachable from a site by following its hyperlinks.

The Breadth-first search algorithm is simple, yet efficient:

```
procedure bfs(G, s)
Input:    Graph G = (V, E), directed or undirected; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞

dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
   u = eject(Q)
   for all edges (u, v) ∈ E:
      if dist(v) = ∞:
         inject(Q, v)
         dist(v) = dist(u) + 1
```

Source: *Algorithms. S. Dasgupta, C.H. Papadimitriou, U.V. Vazirani. July 18, 2006.*

Following this algorithm, we see that each vertex is placed in the queue once, leading to a total of $2 \mid V \mid$ queue operations (one injection and ejection per vertex), each taking $O(1)$ time. The for loop that iterates through edges checks to see if the vertex v has

already been visited, and injects it into the queue if it has not. Therefore, it performs $2|E|$ operations for undirected graphs and $|E|$ operations for directed graphs. As a result, the asymptotic performance is $O(|V|+|E|)$. Since the goal is to traverse the graph and vertices are reached by exploring edges, this runtime is the best lower bound asymptotic runtime.

My implementation of breadth-first search utilizes the above algorithm with some modifications. The asymptotic performance is $O(|V|*maxID^2)$ per BFS traversal. Since the problem asks to run BFS for each key in keys, then the total runtime will be $O(k*|V|*maxID^2)$, where k is the number of keys provided. The first loop iterates through a queue a total of $|V|$ times. For each vertex u in the queue, the corresponding row in the adjacency matrix is iterated through to find edges. This requires *maxID* checks, where *maxID* is the largest ID label found for a vertex in the edges list. Once an edge (u,v) is found, we then check if v has already been visited by iterating N times through the visited array. This could be reduced to an $O(1)$ operation if a hash table was implemented. Since this process is done for each key, then we can expect the total number of operations T to be $T=k*N*maxID^2$, where $k=num\_keys$ and $N=num\_vertices$.

In terms of space, the breadth_first_search function requires a graph in the form of an adjacency matrix of size *maxID x maxID* of type *int64_T*, which has a size of 8 bytes. We also keep a queue of size N of type *int64_t*, and a visited array of size N of type *int* (4 bytes). There is also a two-dimensional array *parents* of size $k \times 2$ of type *int64_t* and an array with k keys of type *int64_t*. Therefore we can expect the space requirement S, in bytes, to be $S = 8maxID^2 + 8N + 4N + 2*8k + 8k$, or $S = 8maxID^2 + 12N + 24k$.