Problem set 10

by Maksim Al Dandan

April 21, 2024

1 Task 1

1.1 Statement

Assume that vertex labels in a graph $\mathcal{G}$ are integers. Imagine a variation of the adjacency-map graph representation [Goodrich, §14.2.2], where instead of maps we use Red-Black trees [Cormen, §13] with vertex labels serving as keys in the search tree. What are the worst-case time complexities of the following Graph ADT operations over this modified representation?

(a) insertVertex (v)
(b) insertEdge (from, to, e)
(c) removeEdge (from, to)
(d) removeVertex (v)

Assume that v, from, and to arguments above are labels, not references to vertex objects.

1.2 Solution

(a) $O(1)$
(b) $O(n)$
(c) $O(n)$
(d) $O(n \log n)$

2 Task 2

2.1 Statement

Would you use the adjacency matrix structure [Goodrich, §14.2.3] or the adjacency list structure [Goodrich, §14.2.2] in each of the following cases? Justify your choice in each case.

(a) The graph has 20,000 vertices and 80,000 edges, and it is important to use as little space as possible.

(b) The graph has 20,000 vertices and $80,000,000$ edges, and it is important to use as little space as possible.

(c) The graph has 20,000 vertices and 20 edges, and you need to answer the query getEdge (u, v) as fast as possible, no matter how much space you use.

(d) The graph has 20,000 vertices and 80,000 edges, and you need to answer the query getEdge (u, v) as fast as possible, no matter how much space you use.

2.2 Solution

(a) The adjacency list structure is more suitable for this case. The adjacency matrix structure would require $20,000^2$ bits of space, while the adjacency list structure would require $100,000$ bits of space. The adjacency list structure is more space-efficient.

(b) The adjacency list structure is more suitable for this case. The adjacency matrix structure would require $20,000^2$ bits of space, while the adjacency list structure would require $80,020,000$ bits of space. The adjacency list structure is more space-efficient.

(c) The adjacency matrix structure is more suitable for this case. The adjacency matrix structure would allow for constant time access to the edge between two vertices, while the adjacency list structure would require linear time access to the edge between two vertices.

(d) The adjacency matrix structure is more suitable for this case. The adjacency matrix structure would allow for constant time access to the edge between two vertices, while the adjacency list structure would require linear time access to the edge between two vertices.

3 Task 3

3.1 Statement

In your own words, explain why, for an adjacency-list representation of a graph [Goodrich, §14.2.2], the method removeVertex (v) takes $O(\deg(v))$ time (worst case) and does not depend on the number of vertices in a graph. You must explicitly specify what contributes to the running time of the method, as well as explicitly name the data structures involved that make the efficient implementation possible.

3.2 Solution

The method $removeVertex(v)$ takes $O(\deg(v))$ time (worst case) because the method must remove all edges incident to vertex v. The method must iterate through the adjacency list of vertex v and remove each edge. List allows for efficient access to the edges incident to vertex v. It is a list of linked lists, where each linked list contains the edges incident to a vertex. The adjacency list structure allows for constant time access to the edges incident to a vertex. The adjacency list structure makes the efficient implementation of the $removeVertex(v)$ method possible.

References

[Cormen] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. Introduction to Algorithms, Fourth Edition. The MIT Press 2022

[Goodrich] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser. Data Structures and Algorithms in Java. WILEY 2014.