Edsger Dijkstra

Edsger Dijkstra was one of Computer Science's founding father. Despite starting out as a Physicist and programming being an unpopular field at his time, he switched his profession to Programming. He had major contributions in various areas of Computer Science including distributed systems, operating systems, complier construction, graph algorithms, compiler construction, and programming languages.

He is noted for his contribution in multi-process synchronization that is one of the bases for concurrency in distributed systems. Concurrency refers to when different parts of an algorithm are executed in a parallel manner and attain the same result expected when carried out in order. Concurrency increases the speed at which a process is completed and therefore reduces runtimes of processes.

To formulate the multi-process synchronization problem, Dijkstra used the producer-consumer problem, the sleeping barber problem, and deleting items from a singly linked list simultaneously.

In the producer-consumer problem, the producer generates data and places it on a buffer one at time, the consumer then removes data from the buffer one at a time. The problem can be solved by having the producer generate one item then wake the customer up, the consumer uses up one item and wakes the producer up. Edge cases handled include, having the producer sleep if the buffer is full and having the consumer sleep if there are no items to consume.

A deadlock, when both the producer and consumer are waiting for the other to wake them up, occurs if the consumer finds no items on the buffer and is about to sleep but an interruption causes the producer to wake up and create an item, the producer then attempts to wake the consumer but the consumer has still not gone to sleep, therefore when the consumer's process resumes, it goes to sleep and is not woken up again, the producer keeps generating items until the buffer is full and the producer goes to sleep. This results in both the producer and consumer remaining asleep and none can wake the other up.

When it comes to deleting items next to each other in a singly linked list, say we have four items in a linked list labelled one to four. If we intend to delete element two and three and leave the linked list with only one and four, if we carry out the deletions simultaneously, element one's next pointer changes from element two to the element three, while element two's next pointer changes from element three to element four. Since only the head of the linked list is returned, we end up with a linked list of elements one, three, and four. If the deletions had not been simultaneous, element two's next pointer would have switched to pointing to element four, then element one's next pointer would switch to pointing to element four; in the end, the linked list would only have elements one and four. The solution involves preventing simultaneous running of both processes if they access the same data.

Dijkstra came up with mutual exclusion as the solution to thus multi-synchronization problem. Mutual exclusion involves never having multiple process enter their critical stages at the same time for example the producer cannot resume control if the consumer's process is about to make the decision on whether to sleep based on having no items on the buffer.

Mutual exclusion is applied by use of locks, monitors, and semaphores. Using locks involves preventing a process from executing until it acquires the permission to access a resource it needs. A few factors come into consideration when choosing the number of locks to use in a system. Each lock comes with overhead costs like memory space allocation costs and therefore the more the number of locks, the higher the overhead costs. With fewer locks, there are fewer overhead

costs but since each lock has to protect larger amounts of processes and this leads to more lock contentions, which refers to when one process tries to gain access a lock that is currently being used by a different process. Having more locks creates more overhead costs but reduces the number of lock contentions.

Mutual exclusion has served as one of the fundamentals in distributed computing that has had a lot of applications including cloud computing that enables applications to be accessed by multiple cloud service users.

Another contribution, Dijkstra had was in graph algorithms. He formulated a greedy algorithm, Dijkstra's algorithm, that is used to solve the shortest path between two points on a graph. A different algorithm, Bellman-Ford algorithm is also intended to solve the single-source-shortest-path (SSSP) problem. The two algorithms approach the problem differently and as such have different runtimes and space complexities.

Bellman-Ford algorithm works by starting from a given vertex and relaxing every edge on the graph V-1 times, where V is the number of vertices and E is the number of edges. At each edge relaxation, the algorithm checks for if the distance to get to the predecessor vertex plus the edge weight is less than the successor vertex weight, if this test is passed, the successor vertex's weight and predecessor are updated. The runtime of this algorithm is O(VE).

Dijkstra's algorithm is an improvement on this runtime since it solves the problem using O (V log(n) +E log(n)). This is because Dijkstra's algorithm relaxes the edges in a clever manner by picking the vertices with the least distance so far. This is done using a priority queue that is implemented using a minimum binary heap. Since the binary heap is balanced, removing the minimum element takes up O (log (n)) runtime, insertions also take O (log (n)) runtime. This is an improvement from linear time as in the case where we might have decided to use a sorted or unsorted array.

Applications of single-source-shortest-problem include network routing protocols and travel scheduling in flights, Uber, Lyft, and google maps. In a telephone network, bandwidth refers to the highest frequency that a line can support. Higher bandwidths transfer data faster, therefore it is important to use lines with higher bandwidths. Dijkstra's algorithm can be used to find lines with the highest bandwidths using a priority queue implemented suing maximum binary heaps. In travel, minimum priority queues are used to decide on the most favorable travel options, connecting flights and arrival times.

Dijkstra had influence in construction of structured programming languages and this includes object-oriented programming, which is the backbone of many modern programming languages like Java, C++, and Python. It is clear that contributions Edsger made have been invaluable in applications in Computer Science and research continues on other viable applications.

References

Attiya, H., & Welch, J. (2004). *Distributed computing: Fundamentals, simulations, and advanced topics*. New Jersey: John Wiley & Sons.

Broy, M., & Denert, E. (2002). *Software pioneers: Contributions to software engineering*. Berlin: Springer.

Tov, J. (n.d.). Dijkstra's Algorithm. Retrieved October 26, 2017, from http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/lec/12-sssp.pdf

Tov, J. (n.d.). Priority Queue ADT; binary heaps. Retrieved October 26, 2017, from http://users.eecs.northwestern.edu/~jesse/course/eecs214-fa17/lec/13-binheap.pdf

Dijkstra’s algorithm. (2018, October 10). Retrieved from https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

(n.d.). Retrieved from http://www.csl.mtu.edu/cs2321/www/newLectures/30_More_Dijkstra.htm

(n.d.). Retrieved from http://users.cs.cf.ac.uk/Dave.Marshall/C/node31.html#SECTION00311000000000000000