**Application of dynamic memory allocation in searching in a text document using splay trees, skip lists and treaps**

**Madhullika. R 106121071**

**R. Kishore Kumar 106121105**

**Introduction:**

Dynamic memory allocation refers to the process of allocating memory space at runtime, as opposed to at compile time. This technique enables programs to allocate memory as needed and adjust memory usage during execution, which can be especially useful when dealing with data of varying size or when memory requirements are unknown in advance. Dynamic memory allocation is typically used in programming languages such as C and C++, where it provides a way to allocate memory on the heap, which is a region of memory that is not managed by the compiler. However, the flexibility and power of dynamic memory allocation also come with added complexity and potential pitfalls, such as memory leaks and fragmentation, which must be carefully managed.

Resources are constantly scarce. We have always sought for improved resource utilisation; this is the foundation of our success. Memory allocation is related to this goal. Memory must be allocated to the variables we construct in order for actual variables to exist. There is now a limit between how we imagine it happens and how it actually happens.

What is the process by which a computer creates a variable?

When we think of creating something, we think of starting from scratch, but this isn't what happens when a computer creates a variable 'X'; to the computer, it's more like an allocation; the computer just allocates a memory cell from a pool of pre-existing memory cells to X. It's like assigning someone a room from among a large number of free or unoccupied pre-existing rooms. This example most likely made it extremely clear how the computer allocates memory.

What exactly is Static Memory Allocation? When we declare variables, we are actually preparing all of the variables that will be used, so that the compiler understands that the variable being used is an important element of the programme that the user desires, rather than simply a rogue symbol floating around. So, when we declare variables, the compiler really assigns those variables to rooms (recalling the hotel analogy from earlier). As you can see, this is done before the programme executes; you cannot assign variables using this approach while the programme is running.

Why dynamic memory allocation?

Why should we create another allocation mechanism if this one suffices? Why should we allocate memory when the programme is running? Because, while it may not be obvious, the inability to allocate memory during runtime limits flexibility and reduces space efficiency. In particular, when the input is unknown ahead of time, we suffer from inefficient storage consumption and a lack or excess of slots to enter data (given an array or comparable data structures to store entries). As a result, we define Dynamic Memory Allocation as follows: Dynamic Memory Allocation (not to be confused with DMA) is the technology that allows storage/memory/cells to be allocated to variables during runtime. So, as we've gone through it all, we can see that it allocates memory at runtime, allowing us to use as much storage as we want without worrying about waste.

-

There are several advanced data structures that can be used for dynamic memory allocation, some of which include:

1. Red-black trees: These are self-balancing binary search trees that provide efficient lookup, insertion, and deletion operations. They are often used in memory allocators to maintain free memory blocks.

2. Buddy allocators: These are a type of memory allocator that divides memory into equal-sized blocks and maintains a binary tree to keep track of free blocks. They are efficient for allocating and deallocating memory blocks of varying sizes.

3. Segregated free lists: This data structure maintains separate linked lists of free memory blocks of different sizes. It is commonly used in memory allocators to provide quick allocation of blocks of the appropriate size.

4. Threaded binary trees: These are binary search trees where every leaf node has a null right child, and its left child points to the next node in an in-order traversal. They are used in some memory allocators to provide fast traversal of free memory blocks.

5. Fibonacci heaps: These are a type of heap data structure that provides efficient insertion, deletion, and decrease-key operations. They are used in some memory allocators to maintain the set of free memory blocks.

6. Splay trees: Splay trees are self-adjusting binary search trees that reorganize themselves so that frequently accessed nodes are closer to the root. In dynamic memory allocation, splay trees can be used to maintain a set of free memory blocks. When a block is allocated or deallocated, the splay tree can be reorganized to ensure that frequently accessed blocks are closer to the root, reducing the time needed to search for a free block.

7. Treaps: Treaps are randomized binary search trees where each node has a priority value assigned randomly. In dynamic memory allocation, treaps can be used to maintain a set of free memory blocks. When a block is allocated or deallocated, the priority values of the nodes can be adjusted to ensure that frequently accessed blocks have higher priority values, reducing the time needed to search for a free block.

8. Skip lists: Skip lists are a probabilistic data structure that use multiple layers of linked lists to provide efficient search and insertion operations. In dynamic memory allocation, skip lists can be used to maintain a set of free memory blocks. When a block is allocated or deallocated, the skip list can be updated to ensure that frequently accessed blocks are closer to the top of the list, reducing the time needed to search for a free block.

Overall, these data structures can provide efficient solutions for dynamic memory allocation by maintaining a set of free memory blocks and quickly finding an appropriate block when one is needed. These advanced data structures provide efficient and scalable solutions for dynamic memory allocation in computer systems.

In this project we will discuss in detail the topic of dynamic memory allocation using 3 from the above list of data structures: splay tree, treap, and skip lists. In this project, you will implement each of these data structures and compare their performance in terms of time and space complexity.

Splay Tree

Splay trees are a type of self-adjusting binary search tree that can be used for efficient dynamic memory allocation. In a splay tree, nodes that are frequently accessed are moved closer to the root, making them easier to access in future operations. This property allows splay trees to provide fast access to recently allocated memory and can reduce the average cost of memory allocation and deallocation operations.

One common use case for splay trees in dynamic memory allocation is the implementation of memory pools. A memory pool is a pre-allocated region of memory that can be divided into smaller chunks and assigned to various objects and data structures at runtime. Splay trees can be used to keep track of the free blocks of memory within a memory pool, allowing for efficient allocation and deallocation of memory without the need for expensive memory management operations.

Dynamic memory allocation is a common task in computer programming, where memory needs to be allocated and deallocated dynamically during the execution of a program. Splay trees are self-balancing binary search trees that can be used to efficiently perform operations such as insertion, deletion, and searching in a set of items. In this answer, I will provide an outline of how to implement dynamic memory allocation using splay trees.

Implementation of DMA using splay tree:

Treaps

A treap is a binary search tree where each node is assigned a random numeric priority, and the nodes are in heap order with respect to those priorities. Two properties are guaranteed: The binary search tree property on the keys. The max heap property on the priorities. They are useful in situations where you need to dynamically allocate memory and maintain a certain ordering or priority.

One example of an application of treaps with dynamic memory allocation is in a priority queue. A priority queue is a data structure where each element has a priority associated with it, and elements are removed from the queue in order of priority. A treap can be used to implement a priority queue where the priority is based on a key associated with each element, and the elements are ordered in a binary search tree based on the keys.

When a new element is added to the priority queue, a new node is dynamically allocated for it and inserted into the treap based on its key. The priority of the element is determined by the priority of the node in the treap. The treap is then rebalanced to maintain the heap property.

When an element is removed from the priority queue, the node corresponding to the element is removed from the treap and the memory for the node is deallocated. The treap is then rebalanced to maintain the binary search tree property.

Overall, treaps with dynamic memory allocation can be useful in situations where you need to maintain a priority queue or other ordered data structure that requires dynamic memory allocation.

Skip lists

Skip lists are a probabilistic data structure that provide an efficient way to search, insert, and delete elements in a sorted list or map.Skip lists consist of a series of linked lists, where each list represents a level in the skip list. The lowest level contains all the elements in sorted order, while higher levels contain "skip" pointers that allow for faster traversal of the list. Each element in a skip list has a key and a value, and elements are sorted by their keys. The skip pointers form a linked structure that "skips" over certain elements in the lower levels, reducing the number of comparisons needed during search, insertion, and deletion operations.

Skip lists can be used for dynamic memory allocation due to their efficient search, insertion, and deletion operations. Skip lists provide an average time complexity of O(log n) for these operations, making them suitable for managing a large number of memory blocks.In a dynamic memory allocation system, memory blocks may need to be allocated, reallocated, and deallocated frequently as the program executes. Skip lists can be used to maintain a sorted list of allocated memory blocks, where the key represents the starting address of each memory block. This allows for efficient searching of available memory blocks that match the requested size, making it easy to find a suitable memory block for allocation. Insertion and deletion of memory blocks in the skip list can also be done efficiently, allowing for fast reallocation and deallocation operations.Furthermore, skip lists are randomized data structures, which means that the structure of the skip list is determined probabilistically during insertion and deletion operations. This randomization helps to balance the distribution of memory blocks across the skip list, reducing the chances of encountering worst-case scenarios that could result in poor performance.