DSALG Report

Data structures and algorithms

Computer Science

UP899802

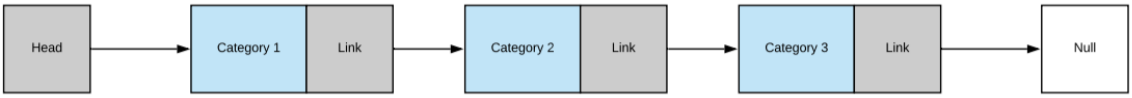
Contents

Pg.:

Critical review of Approach 1

SSL use in approach 1

A SSL is a linear data structure (Each node has a unique predecessor and unique successor). Every node (except the last one) contains a reference (location/Link) to the next node.



The above model shows how this will work. Each data portion of each node holds a category name and another SSL that contains all the species of a category. The species SSL holds a species name and details on that species when required.

SSL's can be created in a forward (Inserted A-Z) or backward(Inserted Z-A) direction, for this use either is suitable as long as the resulting list is ordered from A to Z.

Operations

- List all species of an orchid for a specified category, in alphabetical order.
    - A linear searching algorithm will be used to find the specified category, the species list will then be traversed linearly with each data value being returned. Time proportional n, O(n)
- Search for an orchid, given its category and species, and output details of the orchid
    - A linear searching algorithm will be used to find the specified category and then species, assuming details of the orchid is present in the data portion of the species node found this will be returned. Time proportional 2n, O(n)
- Insert a new species of orchid given its category
    - A linear searching algorithm will be used to find the specified category and the position alphabetically where it will be inserted in the species list. Time proportional 2n, O(n)
- Delete a species of orchid given its category
    - A linear searching algorithm will be used to find the specified category the species SSL can then be deleted. Time proportional n, O(n)
- Create a new category of orchid and insert a new species with details into the new Category.
    - A linear searching algorithm will be used to find the position alphabetically where the new category is to be inserted a new species SSL is added to the data contained in the new node. Time proportional n, O(n)

Efficiency of operations

- All searching must be linear O(n)
- Insertion and Deletion is always constant O(1) (Excluding searching)
    - Including searching which always needs to be done O(n), except for insertion at the start of the list O(1).
- Values closer to A will be found faster than values close to Z

Conclusion

SSL's allow dynamic sizing and ease of insertion in comparison to arrays, for this case an SSL would work well for the category's.

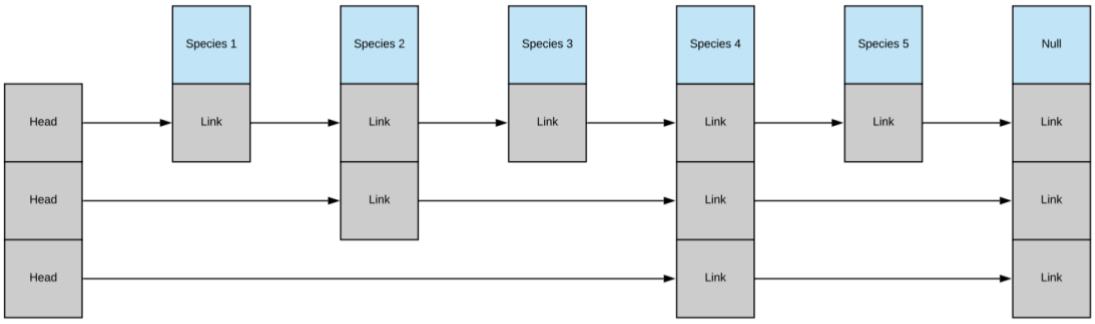
Random access is not allowed. Elements must be accessed sequentially starting from the first node. So, we cannot do binary search or any over searching algorithm that does not visit nodes sequentially with linked lists. Finding an element takes linear time O(n). Insertion and deletion also take linear time as a linear search has to be completed first.

In this case the use of an SSL for orchid categories can be justified as the data set is small. However, as a structure for the species of orchid that may contain much larger data sets its use would not be optimal due to the drawbacks above.

Approach 2

SLL & Skip list use in approach 2

A Skip List is an extension of an ordered singly linked list with additional forward links, added in a randomized way, so that a search in the list can quickly skip parts of the list. A skip list will be used for the species of orchid. A linked list from approach 1 will still be used for the categories.



Each data portion will hold the name of the species and any details if needed.

Operations

- List all species of an orchid for a specified category, in alphabetical order.
    - A linear searching algorithm will be used to find the specified category, the species list will then be traversed linearly on level 0 of the skip list with each data value being returned. Time proportional n, O(n)
- Search for an orchid, given its category and species, and output details of the orchid
    - A linear searching algorithm will be used to find the specified category. Another search for species will be performed this will be a linear search adapted to skip list (see details below), assuming details of the orchid is present in the data portion of the species node found this will be returned. The best case will be O($\log_2 n$) and the worse-case will be O(n)
- Insert a new species of orchid given its category
    - A linear searching algorithm will be used to find the specified category. Another search for species will be performed this will be a linear search adapted to skip list (see details below) inserted alphabetically to its correct position. The best case will be O($\log_2 n$) and the worse-case will be O(n)
- Delete a species of orchid given its category
    - A linear searching algorithm will be used to find the specified category the species SSL can then be deleted. Time proportional n, O(n)
- Create a new category of orchid and insert a new species with details into the new Category.

- A linear searching algorithm will be used to find the position alphabetically where the new category is to be inserted a new species skip list is added to the data contained in the new node. Time proportional n, O(n)

Skip list searching

From A-Z, Starting at the highest level, if the alphabetical order of next node is higher than the species to be found then keep on moving forward on the same level.

If the alphabetical order of next node is lower than the species to be inserted store the pointer to the current node and move one level down and continue the search. Repeat until the species if found.

At the lowest level if the element has not been found it must not exist.

Skip list insertion & deletion

Searching for the alphabetical position to be inserted is completed first. During insertion the number of levels required for a new node is determined by using a probabilistic technique, a random number generator for the level of insertion. During the insertion an update array is used to store pointers at each level prior to insertion. (The best case will be O($\log_2 n$) and the worse-case will be O(n))

Searching for node to be deleted is completed first. Deletion starts once the element is located, rearrangement of pointers is computed to remove the element form list using an update array. After deletion of an element there could be levels that contain no elements, so these levels are to be removed.(The best case will be O($\log_2 n$) and the worse-case will be O(n))

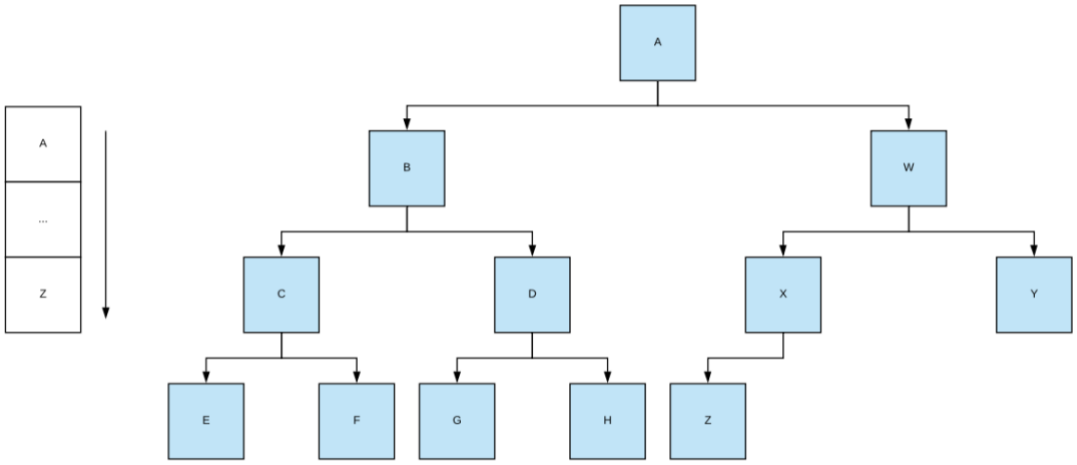
Efficiency of operations

- All searching is best case O($\log_2 n$) and worse-case O(n)
- Insertion and Deletion is always constant O(1) (Excluding searching)
    - Including searching which always needs to be done best case O($\log_2 n$) and worse-case O(n), except for insertion at the start of the list O(1).
- Values can be skipped and found much faster
- Worst case of O(n) if it has no input sequence – in this case it will always be in order A-Z so it is unlikely to reach the worst case.

Approach 3

SSL & Heap use in approach 2

A heap is a binary tree that is complete and has a root node larger than either child nodes. In this case a max heap will be used, the highest priority item will be the first species alphabetically represented in the diagram below by A, child's of A are species below A in the alphabet. A SSL from approach 1 will be used for the category's. Each node of the tree will hold a primary value (Species name) and extra data such as details.



Operations

- List all species of an orchid for a specified category, in alphabetical order.
    - A linear searching algorithm will be used to find the specified category, the species heap will then be traversed linearly with each data value being returned. Time proportional n, O(n)
- Search for an orchid, given its category and species, and output details of the orchid
    - A linear searching algorithm will be used to find the specified category. Another search for species will be performed this will be a binary search, assuming details of the orchid is present in the data portion of the species node found this will be returned. Time complexity O($\log_2 n$)
- Insert a new species of orchid given its category
    - A linear searching algorithm will be used to find the specified category. A binary search for species will be performed and the new species is inserted alphabetically to its correct position. O($\log_2 n$)
- Delete a species of orchid given its category
    - A linear searching algorithm will be used to find the specified category the species SSL can then be deleted. Time proportional n, O(n)
- Create a new category of orchid and insert a new species with details into the new Category.
    - A linear searching algorithm will be used to find the position alphabetically where the new category is to be inserted a new species heap is added to the data contained in the new node. Time proportional n, O(n)

Heap searching

Heap searching is implemented using a binary search. Iterative implementation O(1). Recursive implementation O($\log_2 n$)

Heap insertion & deletion

The new item is inserted on to the bottom of the heap, this will destroy the heap property of the tree you then need to heapify the tree this involves moving the inserted item up the tree until it either becomes the root or finds a parent that satisfies the max heap property. O($\log_2 n$)

Deletion of the root node of the heap can be completed easily leaving 2 sub trees that must be merged into a single tree that satisfies the heap. Deletion an element at an intermediary position in

the heap can be difficult and costly, the element to be deleted is swapped with the last element of the heap so that it can be more easily deleted we then heapify the tree as in insertion. O($\log_2 n$)

Comparison

All approaches have their respective advantages and disadvantages the table below shows the efficiency of the 3 common operations each operation needed for the system is comprised of one or more of these.

	Approach 1	Approach 2	Approach 3
Searching	O(n)	Best - O($\log_2 n$) Worst - O(n)	Iterative - O(1) Recursive - O($\log_2 n$)
Insertion	O(n)	Best- O($\log_2 n$) Worst - O(n)	O($\log_2 n$)
Deletion	O(n)	Best - O($\log_2 n$) Worst - O(n)	O($\log_2 n$)

Approach 3 yields the greatest efficiency however is the most complex to implement. Approach 2 requires more memory due to its use of a skip list but is easier to implement than approach 3. Approach 1 offers the most simplicity of all approaches but is the slowest with constant linear efficiency.

In both approach 2 & 3 I chose to keep a SSL for the categories of orchid this is do to the ease of a SSL, with a small data set the benefit of more advanced structures is lessened, the efficiency gained would only be minimal and not worth the time and effort implementing a more advanced data structure.

Final recommendation

Approach 3 would be my final recommendation it is more difficult to implement and requires more development time. Although the efficiency gained will improve the speed of the system drastically.