Richard Hayes Crowley

07/19/2021

CSC\_1242\_Lab\_011\_QA

**(1)** What benefits do Linked Lists have as a common data structure? Provide an example or two.

**Doubly-linked lists, when implemented with doubly linked nodes, a circular structure, and a list iterator, have the benefit of having O(k) access to the first and last node in the list. It also never needs to resize unlike an Array List, as its memory storage is non-contiguous.   
  
The structure itself is also useful when establishing hierarchical relationships between data items. For example, a node’s “next” pointer could be the data item’s child, and it’s “previous” pointer could be that data item’s parent.**

**The fallbacks of a linked list come when trying to do subscript operations e.g., list[5], as a probe in a linkedList has O(n) complexity while an arrayList, which uses an array’s contiguous memory indexing operation to convert a decimal index to a binary address and access the data item with O(k) complexity.**

**(2)** How can a locomotive train be an illustration of a Doubly – Linked List. Explain your answer.

**A doubly linked list has a dummy “head” pointer that is instantiated like so:**

*self*.head = TwoWayNode()

*self*.head.previous = *self*.head.next = *self*.head

**The dummy head has no data itself, rather, it exists only to provide a circular reference from the first node to the last node. This way, the iterator circles back around to the head when it reaches the end of the list.   
  
This is similar to a train in a closed track system, in that the train (iterator) circles back around to the beginning of the tracks.**

**(3)** Refer to the polynomial belowconsisting of an order or

degree of 3 with 4 terms and individual coefficients and exponents for each term. How can a Linked List help to represent the polynomial?

*P* ( *x* ) = *c* 1 *x* 3 + *c* 2 *x* 2 + *c* 3 *x* 1 + *c* 4 *x* 0

If a term is missing from what would be shown below, would that

be considered a broken link in the list? Explain your answer.

*Q* ( *x* ) = *c* 1 *x* 4 + *c* 2 *x* 2 + *c* 3 *x* 1 + *c* 4 *x* 0

**A linked list could represent a polynomial by having the coefficients and exponents of each term as the data item, with the “next” pointer referencing the next term. An iterator could aggregate and return the sum of these terms based on an input.**

**If there was a missing degree, as in the second example Q(x), then the linked list could still point to the next data item, so I do not think it would result in a “broken” link. A linked list is only broken if its “next” or “previous” pointer is None.**

**(4)** Consider comparing the speed of traversing a regular list with a linked list of the exact size of the regular list. Also consider utilizing built - in timers provided by Python to perform such a speed test. Provide some suggestions to show the speed test result of comparing the two lists.

**Well you could benchmark an ArrayList vs. LinkedList implementation using the computer’s clock. I can tell you that ArrayList would win most of the time, except for when the array list would need to resize (which takes O(n) time complexity). Accessing the first or last Node of a linkedList would take O(k) time, but trying to do a subscript operation on a linkedList could take O(n) time, while in an ArrayList it would take O(k) time.**

**(5)** Blockchains can be established using Linked Lists to chain transactions from customers, corporations, etc. Consider the challenges with the operations or methods to create this linkage. Explain ways that could be possible to efficiently create the list and have the list maintained given transaction possibilities that may add to the chain, be updated or even deleted. What kind of data can be stored in such a list on a per transaction basis?

**I implemented something similar to a block chain in my lab (a “event log”), wherein an event propagates through links in a linked list. The challenge lies in making sure that the event (or in blockchain speak, “transaction”), goes down the correct chain.**