Linked List

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 17, 2020

Singly Linked List

Application: Polynomial Computation

solution 0: use ordered array on (exponent, coefficient)

Issues of (Ordered) Array for Polynomial Computation

ordered (consecutive) array:
not flexible for resizing/insertion/removal

Solution 1: Singly Linked List for Flexible Insertion

overhead of `next` $\Leftrightarrow$ flexible `insertAfter`

Singly Linked List as Abstract Data Structure: Access

access

- `data getAt(node)`
- `node getHead()`
- `node getNext(node)`
- `insertAfter(node, data)`
- `insertHead(data)`

linked list: sequential access; array: random access

Singly Linked List as ADT: Maintenance

maintenance

- `construct(length)`: trivial
- `updateHere(node, data)`: trivial
- `removeAfter(node)`: simple
- `removeHead`: simple

think: dummy head node or not?

Doubly Linked List

`removeHere` for Singly Linked List

`removeHere` (and `insertHere`): hard for singly linked list

Doubly Linked List: More Flexible `removeHere`

overhead of `prev` $\Leftrightarrow$ flexible `removeHere`

Iterator for Sequential Access

iterator: abstraction of array index, linked list node and more!

C++ STL List: a Doubly Linked List

access

- `node getHead(): list.begin()`
- `node getNext(node): iterator++`
- `data getAt(node): (*iterator)`
- `insertHere(node, data): list.insert(iterator, data))`

and more!

maintenance

- `updateHere(node, data): (*iterator = data)`
- `removeHere(node): list.erase(iterator)`

and more!

STL list and its iterator:
a more "structured" way of using doubly linked list

Linked List for Sparse Vectors

Application: Sparse Vector in Scientific Computing

polynomial: can be viewed as special case of sparse vector

Sparse Vector: (Dense) Array versus Linked List

storing only non-zeros can be time/space efficient

Merging Sparse Vectors

"running cursors" algorithm:
similar for other uses, like dot product

Real-World Usage of Sparse Vector: LIBSVM

```
1  double Kernel::dot(const svm_node *px, const svm_node *py){
2          double sum = 0;
3          while(px->index != -1 && py->index != -1){
4                  if(px->index == py->index){
5                          sum += px->value * py->value;
6                          ++px;
7                          ++py;
8                  }
9                  else{
10                         if(px->index > py->index)
11                                 ++py;
12                         else
13                                 ++px;
14                 }
15         }
16         return sum;
17 }
```

good data structure needed everywhere

Linked List in Job Interviews

Linked List Reversal

nothing special, but important to “code on board”

“Cycle” in Linked List?

tortoise-hare (turtle-rabbit) algorithm

Middle of Linked List

two pass, or tortoise-hare algorithm