Linked List

Hsuan-Tien Lin

Dept. of CSIE, NTU

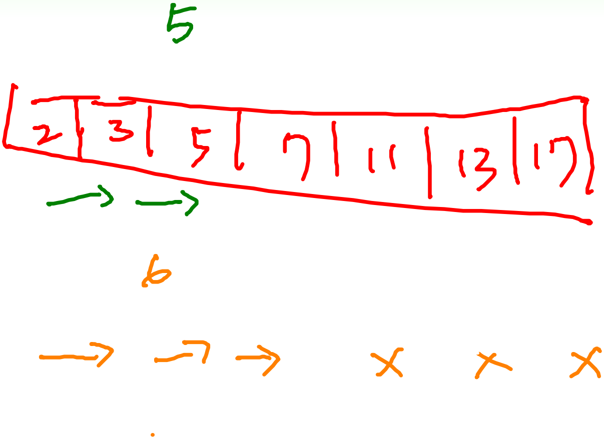
March 9, 2021

What We Have Done

- pseudo code: ‘spoken’ language of programmers
- data structure: scheme of data organization
- why dsa: proper use of resources;
  move from coding to programming
- array: (fast)-index-access data structure
- ordered array: consecutive array with sorted values
  - harder to maintain, e.g. `insert`
  - easier to search by value

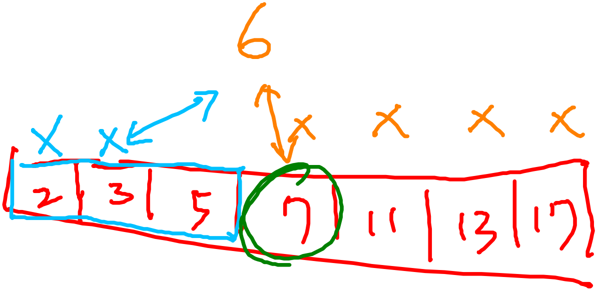
More on Ordered Array

Ordered Array: Sequential Search Algorithm with Cut



ordered: possibly easier to declare not found

Ordered Array: Binary Search Algorithm



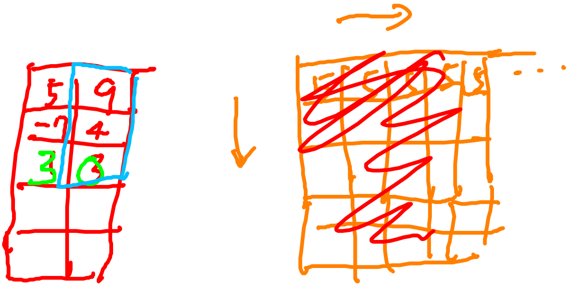
"cut" multiple times by fast random access to the middle

Singly Linked List

Application: Polynomial Computation

$$f(x) = \underline{5}\,x^{\underline{9}} \underline{-7}\,x^{\underline{4}} \underline{+3}$$

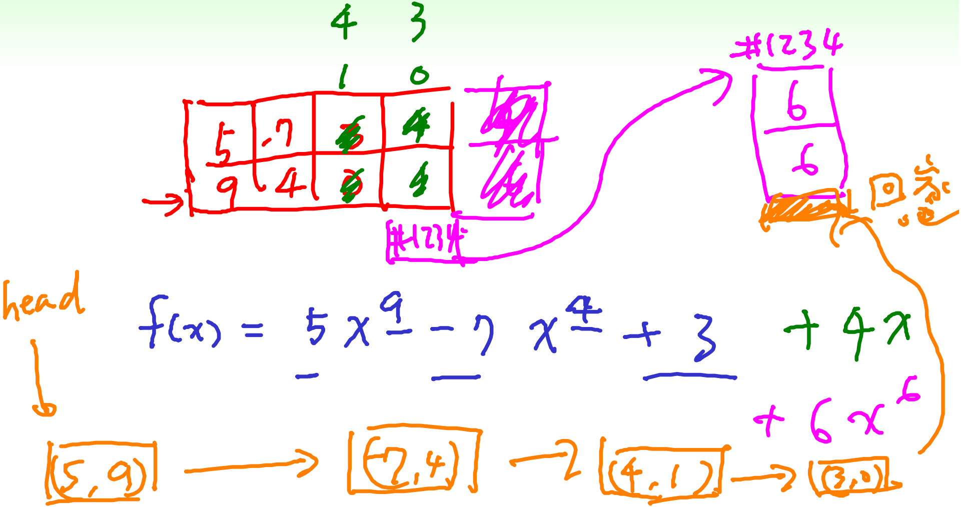
$(5, 9)$ $(-7, 4)$ $(\overset{3,0}{\cancel{1,3}})$



5	9
-7	4
3	$\overset{0}{\cancel{3}}$

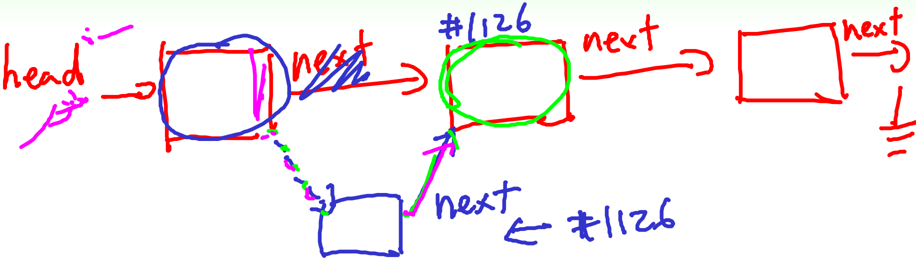
solution 0: use ordered array on (exponent, coefficient)

Issues of (Ordered) Array for Polynomial Computation



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

Solution: Singly Linked List for Flexible Insertion



overhead of `next` ⇔ 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

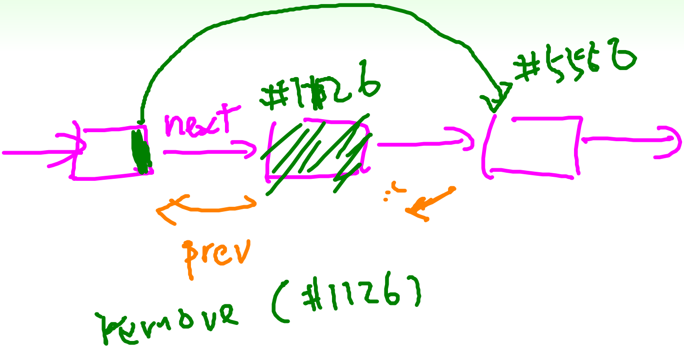
maintenance

- `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



✓ !

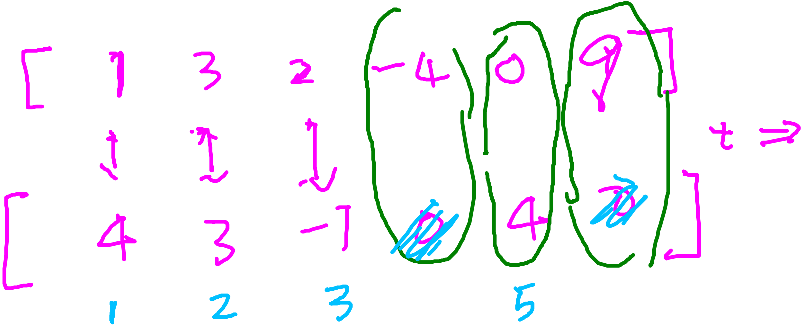
`removeHere` (and `insertHere`): hard for singly linked list

Doubly Linked List: More Flexible `removeHere`

overhead of `prev` ⇔ flexible `removeHere` (and flexible traverse backward)

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

```
double Kernel::dot(const svm_node *px, const svm_node *py){
        double sum = 0;
        while(px->index != -1 && py->index != -1){
                if(px->index == py->index){
                        sum += px->value * py->value;
                        ++px;
                        ++py;
                }
                else{
                        if(px->index > py->index)
                                ++py;
                        else
                                ++px;
                }
        }
        return sum;
}
```

good data structure needed everywhere