CSCI 335 Notes

Anton Goretsky Ralph Vente

Contents

02.07.19

First section was answering questions about the homework, talking about the meaning of friend in…

```
friend Points2 operator+(const Points2 &c1, const Points2 &c2)
```

Back to Algorithm 4 slide in Lec 2 slides. Going through it step by step.

Is an example of an *on-line algorithm.* * Data can be read sequentially * Always provides current best solution while running

Binary Search

Best case scenario: its right in the middle.

Worst case scenario: if we constantly have to go to the middle until only 1 element. And you find that element does not exist.

1) Go to middle # 1
2) Go left or right # 1
3) Do same thing half size. # T(n/2)

$$T(n) = 1 + 1 + T(n/2)$$

$$T(n/2) = 1 + T(n/4)$$

If n is a power of 2, then this turns into a basic logarithmic algorithm.

$$T(N) = O(log(n))$$

Exponentiation

Compute X^N for positive N

$$\left.\begin{aligned} X^0 &= 1 \\ X^1 &= X \end{aligned}\right\}\text{Base Case}$$

$$X^N = (X^{(N/2)^2}), \quad \left.\vphantom{X^{(N/2)^2}}\right\}\text{EVEN}$$

$$X^N = (X^{(N-1/2)^2}) \quad \left.\vphantom{X^{(N/2)^2}}\right\}\text{ODD}$$

Polynomial Evaluation

Horner's Method

```
poly = 0;
for (i = n; i >= 0; i--)
    poly = x * poly + a[i];
```

Lecture 3 - Iterators, Stacks, Other STL things.

Vector vs. List in the sequentially

Vector * constant time indexing * slow to add data to middle, fast to end.

List * doubly linked List * no indexing * fast insertion/removal everywhere

Commonality * `push/pop_back` * `&back(), &front() const`

Vector only * `[], at, capacity, reserve`

List only * `push/pop front`

Iterators

Position represented by iterator. Book will shorten longhand to shorthand iterator. Ex:

```
list<string>::iterator itr1

//Basic Operators:
itr1.begin();
itr1.end(); //points just past last element
```

In an empty vector, `itr1.begin() == itr1.end()`

Iterator Methods

```
itr++
++itr
*itr //returns ref to object stored at location
itr1 == itr2
itr1 !+ itr2
```

For and while loop print array examples given in slides.

****Do not run `*itr` on the end!!****

Watch out for `++itr` vs `itr++`, don't accidentally access the end.

As iterators return reference, speed

```
string value1 = *itr; //copy
const string &value2 = *itr //not copy
//*itr returns reference, not const reference
```

Operations that require Iterators

```
//will insert prior to pos.
iterator insert(iterator pos, const Object &x)

//delete at, return next iter
iterator erase(iterator pos);
iterator erase(iterator start, iterator end);
//when erasing last item watch out for return of end iterator.
```

Using auto when declaring iterators will be much easier.

Erase example in slides will still work on an empty list. It'll just do nothing.

Is linear for a list. O(1) to erase.

For a vector, unless you erase at the end, you have to constantly shift. Is costly.

const_iterators

`*itr` is a reference to the object at the iterators position.

In the code example, the list is not supposed to be changed. `*itr = 0` will be an error.

Use a const_iterator to access data without wanting to change it. `*itr` will now return a const reference.

```
ostream & out = cout;
out << *itr;
//will either send to file or cout depending on what you set out to.
```