CSCI 136: Data Structures and Advanced Programming
Lecture 11
Linked lists
Instructor: Dan Barowy Williams

Topics
• Mathematical induction
  Vectors—why $\text{add}$ is “always” $O(1)$
• Linked lists

Your to-dos
1. Read before Fri: Bailey, Ch 3.4–3.5.
2. Lab 4 (partner lab), due Tuesday 3/9 by 10pm.

Mathematical Induction
Principle of Mathematical Induction

Let \( P(n) \) be a predicate that is defined for integers \( n \), and let \( a \) be a fixed integer.

If the following two statements are true:

1. \( P(a) \) is true.
2. For all integers \( k \geq a \), if \( P(k) \) is true then \( P(k + 1) \) is true.

then the statement

for all integers \( n \geq a \), \( P(n) \) is true

is also true.

To be clear:

If you want to prove that \( P(n) \) is true for all integers \( n \geq a \),

1. You must first prove that \( P(a) \) is true.
2. Then suppose \( P(k) \) is true and prove that \( P(k+1) \) is true.

Names for things and “form”

Hypothesis: \( P(n) \) is true for all integers \( n \geq a \),

1. Base case: \( P(a) \) is true.
2. Inductive step:

For all integers \( k \geq a \), if \( P(k) \) is true then \( P(k+1) \) is true.

Like recursion, there is an analogy
Example

Prove that the sum of the first $n$ integers is:

$$\frac{n(n+1)}{2}$$

$P(n) : 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$

Example: step 1

Step 1: Prove $P(a)$

$P(a) : 1 = \frac{1(1+1)}{2}$

Is this statement true? Yes.

Proof:

$$\frac{1(1+1)}{2} = \frac{2}{2} = 1$$

Example: step 2

Step 2: Prove $P(k) \Rightarrow P(k+1)$

Assume the following is true:

$$P(k) : 1 + 2 + 3 + \ldots + k = \frac{k(k+1)}{2}$$

Prove:

$$P(k+1) : 1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$$

According to $P(k)$, which is true, it must be equal to:

$$(1 + 2 + 3 + \ldots + k) + (k + 1)$$

$$(1 + 2 + 3 + \ldots + k) + (k + 1) = \frac{k(k+1)}{2} + (k + 1)$$

Example: step 2, left side

Step 2: Prove $P(k) \Rightarrow P(k+1)$

$$(1 + 2 + 3 + \ldots + k) + (k + 1)$$

According to $P(k)$, which is true, it must be equal to:

$$\frac{k(k+1)}{2} + (k + 1)$$
Example: step 2, left side

Step 2: Prove $P(k) \Rightarrow P(k+1)$

Simplify

$$= \frac{k(k+1)}{2} + (k + 1)$$

$$= \frac{k(k+1)}{2} + \frac{2(k+1)}{2}$$

$$= \frac{k(k+1) + 2(k+1)}{2}$$

Let’s stop here.
The left side is $\frac{(k+1)(k+2)}{2}$

Example: step 2, right side

Step 2: Prove $P(k) \Rightarrow P(k+1)$

$P(k+1)$ : $1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$

Let’s handle the right side now.

$$= \frac{(k+1)((k+1)+1)}{2}$$

Simplify

$$= \frac{(k+1)(k+2)}{2}$$

Let’s stop here.

Example: step 2, conclusion

Step 2: Prove $P(k) \Rightarrow P(k+1)$

$P(k+1)$ : $1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$

We just showed that the left side

$$= \frac{(k+1)(k+2)}{2}$$

equals the right side

$$= \frac{(k+1)(k+2)}{2}$$

Example: done

Step 1: Prove $P(a)$

Step 2: Prove $P(k) \Rightarrow P(k+1)$

Therefore,

$$P(n) : 1 + 2 + 3 + \ldots + \frac{n(n+1)}{2}$$

For all $n \geq 1$.

Is true.
Expanding vectors: why double?

Why is the array doubling strategy for Vector better than expanding the array one element at a time?

One-at-a-time expansion

Initial array.

Insert element.

New array; copy previous; insert element.

New array; copy previous; insert element.

New array; copy previous; insert element.

... ... ...

Insertion into an array

How much does array insertion cost?

It costs $O(1)$.

In fact, lookup and insertion both cost $O(1)$.

Tradeoff: arrays are fixed size.

Copying an array

How much does an array copy cost?

It costs $O(1) \times m$, where $m$ is the size of the original array.

$\approx O(m)$
How many copies?

# of copies for one-at-a-time expansion:

\[
\begin{align*}
1 & + 2 + 3 + \ldots + (n-1) \\
\text{add()} & \quad \text{2nd} \quad \text{3rd} \quad \text{4th} \quad \ldots \quad \text{nth} \\
\text{elem.} & \quad \text{elem.} \quad \text{elem.} \quad \ldots \quad \text{elem.}
\end{align*}
\]

Recall theorem: \(1 + 2 + 3 + \ldots + k = k(k+1)/2\)

Sub \(n-1\) for \(k\):

\[
(n-1)((n-1)+1)/2 = n(n-1)/2
\]

\[
= (n^2-n)/2
\]

One-at-a-time expansion costs \(\approx O(n^2)\)

How many copies?

# of copies for doubling expansion:

\[
\begin{align*}
1 & + 2 + 4 + \ldots + (n/2) \\
\text{add()} & \quad \text{up to} \quad \text{up to} \quad \text{up to} \quad \text{up to} \\
\text{2nd} & \quad \text{4th} \quad \text{8th} \quad \text{nth} \\
\text{elem.} & \quad \text{elem.} \quad \text{elem.} \quad \text{elem.}
\end{align*}
\]

Neat theorem: \(1 + 2 + 4 + \ldots + 2^{k-1} = 2^k-1\)

Suppose \(n = 2^k\).

Then \(1 + \ldots + n/2 = 1 + \ldots + 2^{k-1} = 2^{k-1} = n-1\)

Doubling expansion costs \(\approx O(n)\)

One-at-a-time expansion costs \(\approx O(n^2)\)

Doubling expansion costs \(\approx O(n)\)

A good practice induction problem

Prove: \(n\) cents can be obtained by using only 3-cent and 8-cent coins, for all \(n \geq 15\).

Which is faster?

💩 One-at-a-time expansion costs \(\approx O(n^2)\)

😎 Doubling expansion costs \(\approx O(n)\)

Doubling is Vin Diesel-approved.
A linked list is a recursive data structure. A linked list is composed of simple pieces called list nodes. A list node contains data (of generic type $T$) and a reference (a “link”) to either another list node or null.

The empty list is defined as null.

Every other list has at least one list node.
A list node stores data of type $T$. Here, $T$ is Integer.

The next field stores a reference ("link") to the next node. If the node is the last node, the next node is null.

If the next node is not null, it is, recursively, a list node. The last node in the list must always point to null.

A list has parts.
When we add data to a list, we always append to the head.

To find a value, we must always traverse the list starting from the head. E.g., looking for 2…
Linked List

To find a value, we must always traverse the list starting from the head. E.g., looking for \(2\)…

Example code

The purpose of a class:

To “abstract away” implementation details.

Abstraction

Abstraction is the process of removing irrelevant information so that a program is easier to understand.
Think of a class as having two sides.

**The outside:** A class should represent one idea, and the class's methods should support working with that one idea.

**E.g., Vector:** Represents an arbitrarily long sequence of elements. Ideally, it also has the same asymptotic properties as an array.

You can:

- add to it
- remove from it
- ask it for its size...
- convert it to `toString`
- etc.

The user of a class should not need to know how a class works.

Design so user never needs to “look inside”.

---

**Vector**

- `object[] elementData`
- `size`
- `add`
- `remove`
- `toString`
- `int elementCount`
- `clear`

**LinkedList**

- `ListNode list`
- `size`
- `add`
- `remove`
- `toString`
- `int elementCount`
- `clear`
Do you see any similarities?

The two classes share the same interface.

Interface

An interface defines boundary between two systems across which they share information. An interface is a contract: calling a method defined in an interface returns the data as promised.

Because an interface contains no implementation, programmers who use them cannot rely on implementation details.

E.g., the List interface states that there must be an add method but does not say how it should be implemented.

structure5 List implementations

In structure5, the following classes are all a kind of List:

- Vector
- SinglyLinkedList
- DoublyLinkedList
-CircularList

So what is a List exactly?

Recap & Next Class

Today:

- Why Vector should double
- Lists

Next class:

- ADTs
- More lists