

Data Structures

Iterators & Exam 1 Review

CS 225

September 11, 2026

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Learning Objectives

Finish discussing queue implementation details

Discuss the importance of iterators

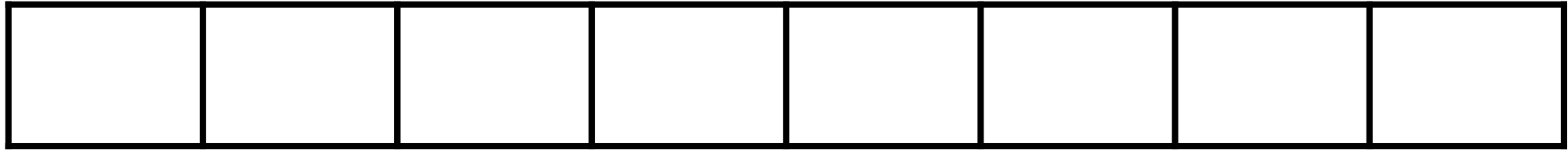
Review content seen on exam 1 (and how to prepare)

Introduce trees and the tree ADT (Time permitting)

Stack and Queue

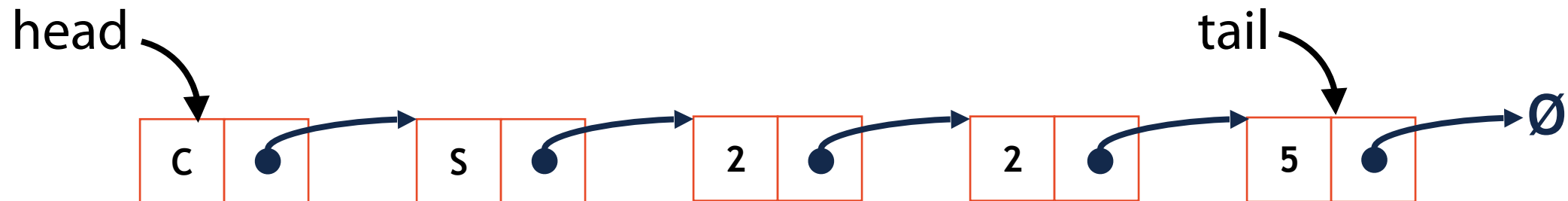
Taking advantage of special cases in lists / arrays

insertBack $O(1)$



insertFront $O(1)$

insertBack $O(1)$



Stack ADT

- [Order]: LIFO (Last in first out)
- [Implementation]: Array (such as `std::vector`)

Linked List also works using insert / remove Front

- [Runtime]: $O(1)$ Push, Pop, Top

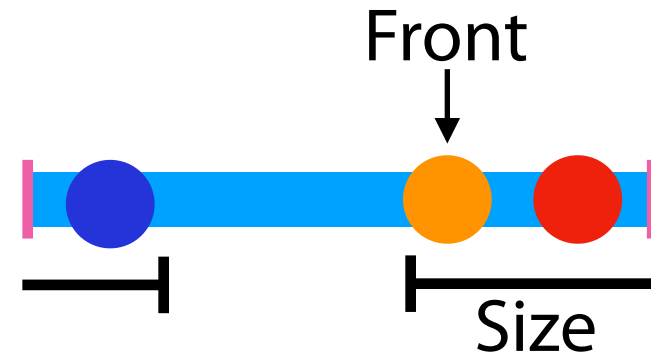
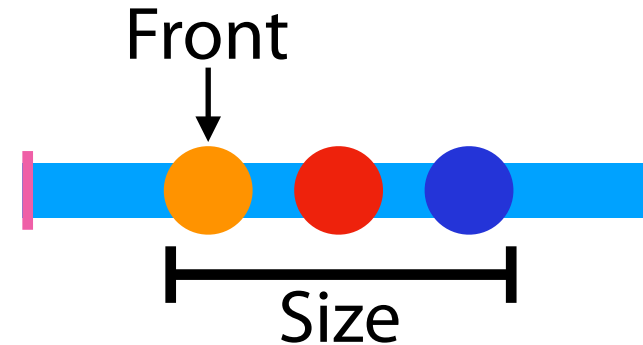
If using array, $O(1)^*$ if we need to resize.

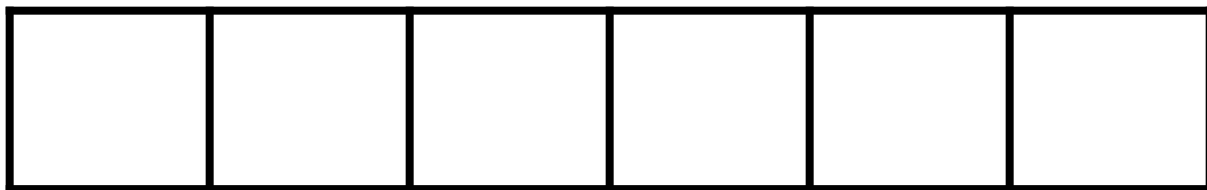
(Circular) Queue Data Structure



Queue.h

```
1 #pragma once
2
3 template <typename T>
4 class Queue {
5     public:
6         void enqueue(T e);
7         T dequeue();
8         bool isEmpty();
9
10    private:
11        T *data_;
12        unsigned capacity_;
13        unsigned size_;
14        unsigned front_;
15 };
```





Enqueue(D) :

Dequeue() :

Size:

Front:

```
Queue<int> q;  
q.enqueue(3);  
q.enqueue(8);  
q.enqueue(4);  
q.dequeue();  
q.enqueue(7);  
q.dequeue();  
q.dequeue();  
q.enqueue(2);  
q.enqueue(1);  
q.enqueue(3);  
q.enqueue(5);  
q.dequeue();  
q.enqueue(9);
```

Capacity:



Both actions have required checks

Enqueue(D): Add data to 'back' of queue

Insert D at index **$(\text{size} + \text{front}) \% \text{capacity}$**

size++ (as long as size $\neq$ capacity)

Dequeue(): Remove data at index front

front = $(\text{front} + 1) \% \text{capacity}$

size-- (as long as size $\neq$ 0)

Size: 3

Front: 3

Capacity: 6

Queue<int> q;

...

q.enqueue(D);

q.dequeue();

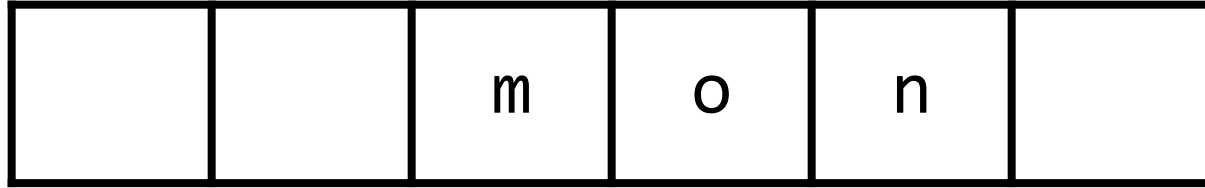
q.dequeue();

q.dequeue();

q.dequeue();

q.enqueue(E);

Queue Data Structure: Resizing



```
Queue<char> q;
```

```
...
```

```
q.enqueue(d);
```

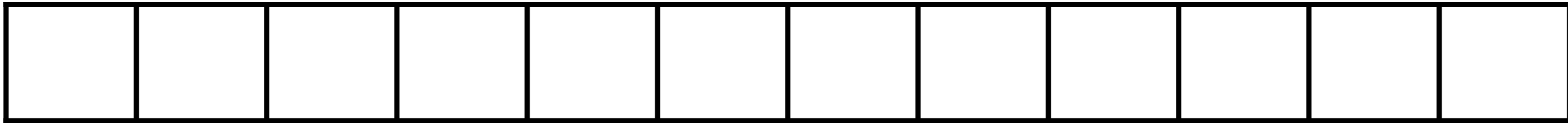
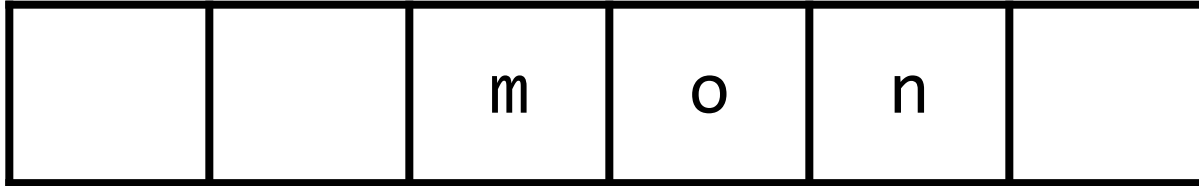
```
q.enqueue(a);
```

```
q.enqueue(y);
```

```
q.enqueue(i);
```

```
q.enqueue(s);
```

Queue Data Structure: Resizing



```
Queue<char> q;
```

```
...
```

```
q.enqueue(d);
```

```
q.enqueue(a);
```

```
q.enqueue(y);
```

```
q.enqueue(i);
```

```
q.enqueue(s);
```

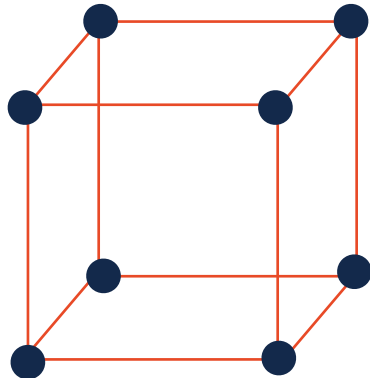
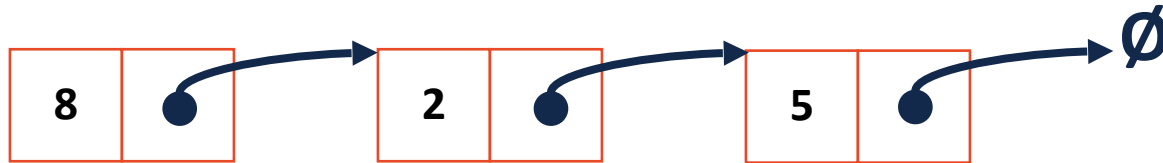
Queue ADT

- [Order]:
- [Implementation]:
- [Runtime]:



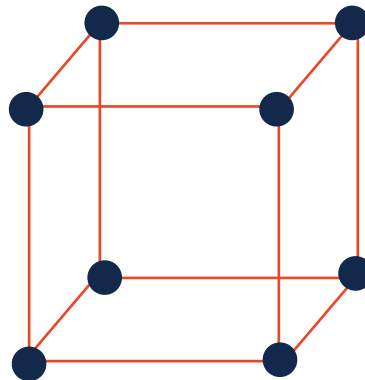
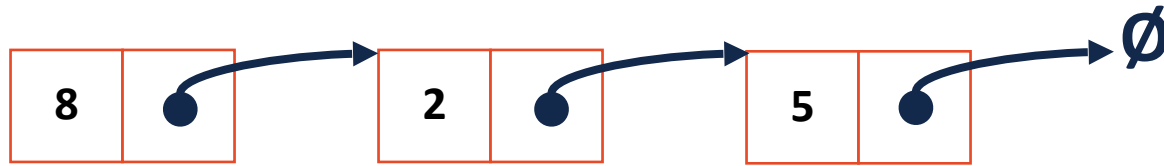
Iterators

We want to be able to loop through all elements for any underlying implementation in a systematic way



Iterators

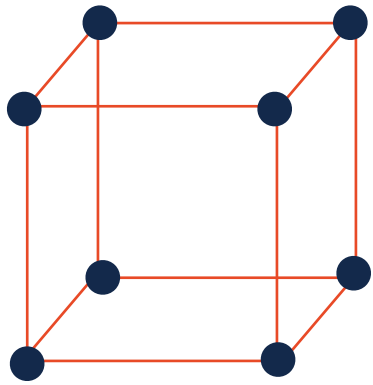
We want to be able to loop through all elements for any underlying implementation in a systematic way



Cur. Location	Cur. Data	Next
<code>ListNode *</code> <code>curr</code>		
<code>unsigned</code> <code>index</code>		
Some form of <code>(x, y, z)</code>		

Iterators

Iterators provide a way to access items in a container without exposing the underlying structure of the container



```
1 Cube::Iterator it = myCube.begin();  
2  
3 while (it != myCube.end()) {  
4     std::cout << *it << " ";  
5     it++;  
6 }  
7
```

Iterators

For a class to implement an iterator, it needs two functions:

Iterator begin()

Iterator end()

Iterators

For a class to implement an iterator, it needs two functions:

Iterator begin()

Returns an Iterator object pointing at the 'first item'

Iterator end()

Returns an Iterator object pointing one entry past end of dataset

Iterators

The actual iterator is defined as a class **inside** the outer class:

1. It must be of base class **std::iterator**

2. It must implement at least the following operations:

Iterator& operator ++()

const T & operator *()

bool operator !=(const Iterator &)



Iterators

Here is a (truncated) example of an iterator:

```
1  template <class T>
2  class List {
3
4      class ListIterator : public
5  std::iterator<std::bidirectional_iterator_tag, T> {
6      public:
7
8          ListIterator& operator++();
9
10         ListIterator& operator--();
11
12         bool operator!=(const ListIterator& rhs);
13
14         const T& operator*();
15     };
16
17     ListIterator begin() const;
18
19     ListIterator end() const;
20 };
```

MP_List Iterator

```
1 class ListIterator {  
2     private:  
3         // @TODO: graded in mp_lists part 1  
4         ListNode* position_  
  
5     public:  
6         ...  
7  
8         ListIterator() : position_(NULL) { }  
9         ListIterator(ListNode* x) : position_(x) { }
```

```
1  #include <list>
2  #include <string>
3  #include <iostream>
4
5  struct Animal {
6      std::string name, food;
7      bool big;
8      Animal(std::string name = "blob", std::string food = "you", bool big = true) :
9          name(name), food(food), big(big) { /* nothing */ }
10 };
11
12 int main() {
13     Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
14     std::vector<Animal> zoo;
15
16     zoo.push_back(g);
17     zoo.push_back(p);    // std::vector's insertAtEnd
18     zoo.push_back(b);
19
20     for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
21         std::cout << (*it).name << " " << (*it).food << std::endl;
22     }
23
24     return 0;
25 }
```



```
1
2 std::vector<Animal> zoo;
3
4
5 /* Full text snippet */
6
7     for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
8         std::cout << (*it).name << " " << (*it).food << std::endl;
9     }
10
11
12 /* Auto Snippet */
13
14     for ( auto it = zoo.begin(); it != zoo.end(); ++it ) {
15         std::cout << (*it).name << " " << (*it).food << std::endl;
16     }
17
18 /* For Each Snippet */
19
20     for ( const Animal & animal : zoo ) {
21         std::cout << animal.name << " " << animal.food << std::endl;
22     }
23
24
25
```

Exam 1 Review

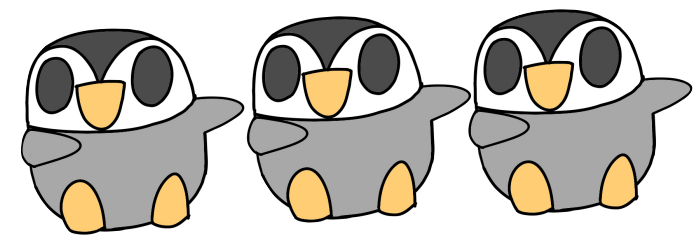
Exam content focuses on fundamentals and lists

To prepare for MCQ:

- Have a clear understanding of the list ADT
- Understand the Big O details behind linked list and array list
- Be able to apply this knowledge in new contexts

e.g. 'What if I had a linked list and made this modification...'

Lists



The not-so-secret underlying implementation for many things

	Singly Linked List	Array
Look up arbitrary location	$O(n)$	$O(1)$
Insert after given element	$O(1)$	$O(n)$
Remove after given element	$O(1)$	$O(n)$
Insert at arbitrary location	$O(n)$	$O(n)$
Remove at arbitrary location	$O(n)$	$O(n)$
Search for an input value	$O(n)$	$O(n)$

Special Cases:

insertFront

insertBack (not full)

Exam 1 Review

To prepare for the free response question:

- Have a clear understanding of the list ADT
- Understand the Big O details behind linked list and array list
- **Be able to apply this knowledge in new contexts**

Brainstorm: What are some variations on linked lists / arrays?

Exam 1 Review

Preparing for the coding question:

- Make sure you understand the fundamentals of debugging
- Review your coding assignments so far (debug / stickers)
- The coding question will be similar to that of practice exam

Brainstorm: What are some ways we could test your knowledge of PNGs / HSLAPixels in the context of arrays?



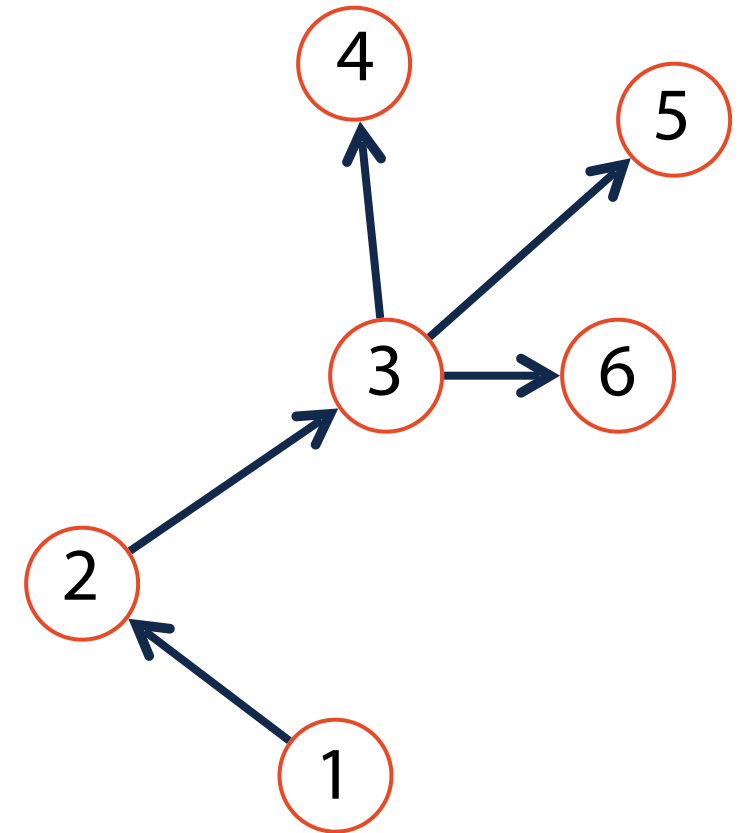
Questions?

Trees

A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

[In CS 225] a tree is also:

- 1) Acyclic — No path from node to itself
- 2) Rooted — A specific node is labeled root

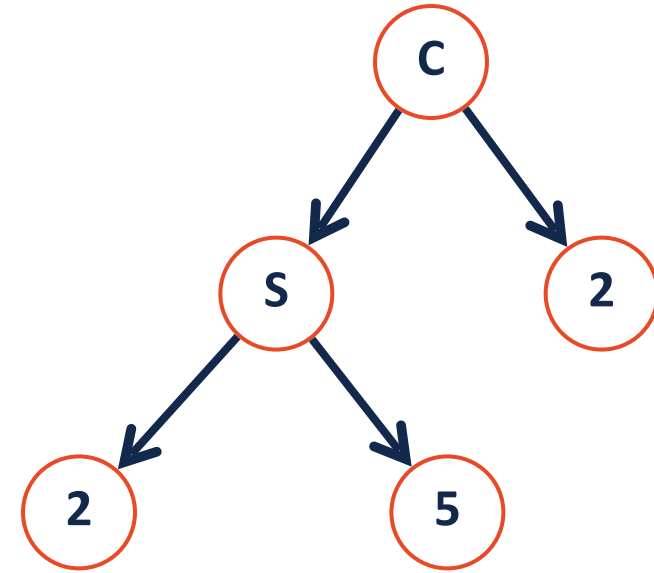


Binary Tree

A **binary tree** is a tree T such that:

1. $T = \emptyset$

2. $T = (data, T_L, T_R)$

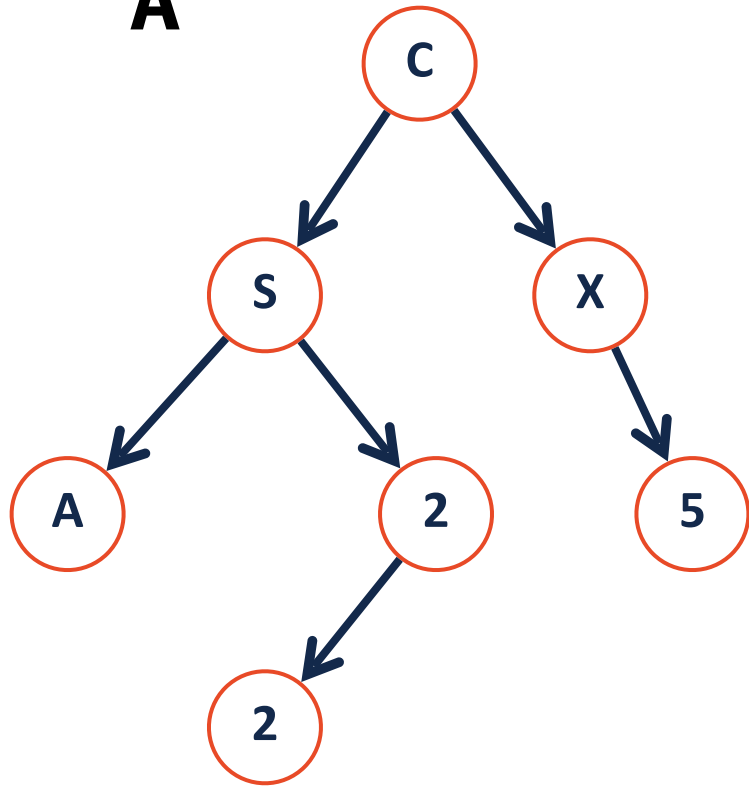


Which of the following are binary trees?

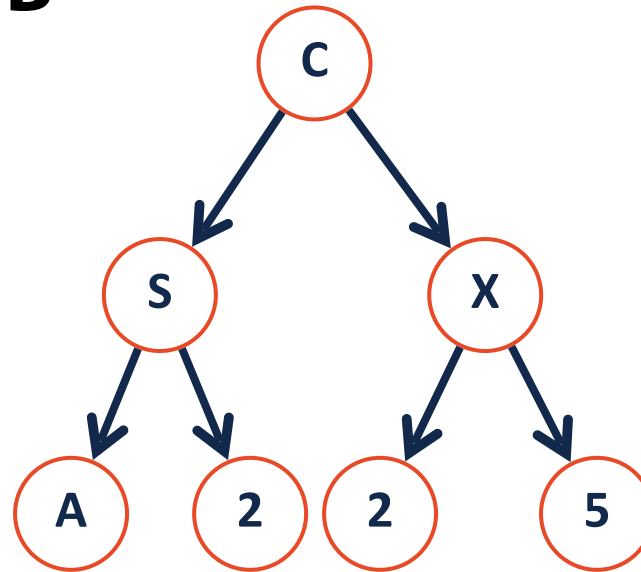


Join Code: 225

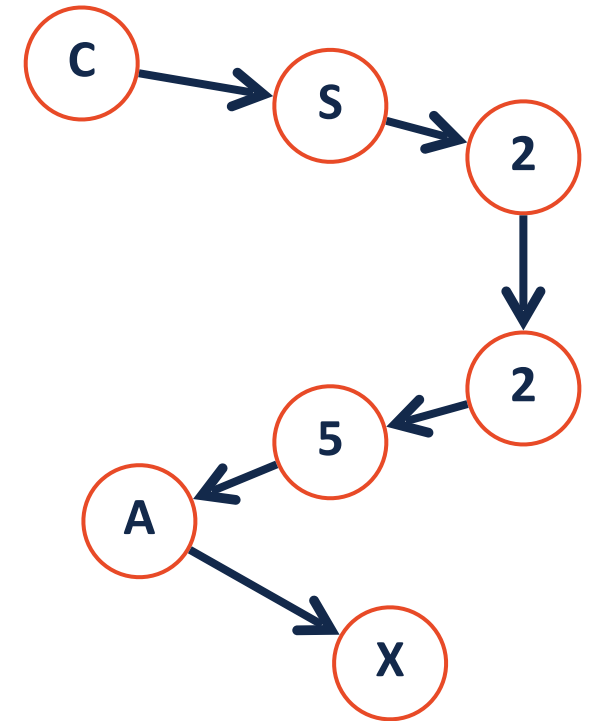
A



B



C





Tree ADT