Data Structures

Queues, Iterators, and maybe Trees?

CS 225
Brad Solomon

September 12, 2025

Department of Computer Science

Exam 1 (9/17 — 9/19)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be released on PL

Topics covered can be found on website

Register now

https://courses.engr.illinois.edu/cs225/fa2025/exams/

Learning Objectives

Discuss the importance of iterators

Review trees and binary trees

Practice tree theory with recursive definitions and proofs

Discuss the tree ADT

Stack ADT

- [Order]: LIFO

- [Implementation]: Array (such as std::vector)

- [Runtime]: O(1) Push and Pop

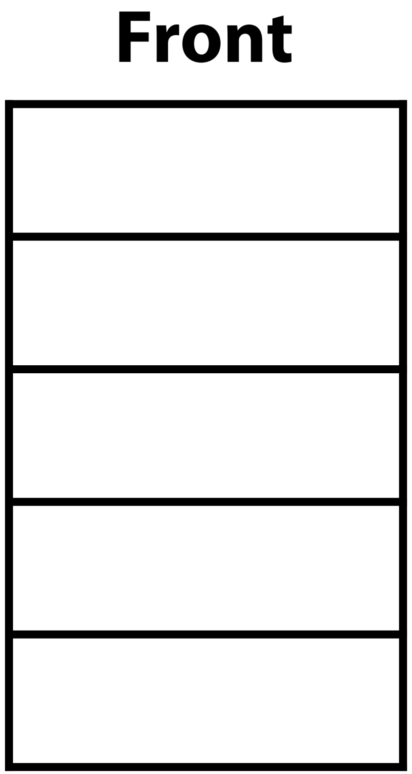
Queue Data Structure

A **queue** stores an ordered collection of objects (like a list)

However you can only do two* operations:

Enqueue: Put an item at the back of the queue

Dequeue: Remove and return the front item of the queue




```
enqueue(3); enqueue(5); dequeue(); enqueue(2)
```

Queue Data Structure

The queue is a **first in — first out** data structure (FIFO)

What data structure excels at removing from the front?

Can we make that same data structure good at inserting at the end?

Queue Data Structure

The C++ implementation of a queue is also a vector or deque — why?

Engineering vs Theory Efficiency

	Time x1 billion	Like
L1 cache reference	0.5 seconds	Heartbeat 💓
Branch mispredict	5 seconds	Yawn 😲
L2 cache reference	7 seconds	Long yawn 😲 😲 😲
Mutex lock/unlock	25 seconds	Make coffee ☕
Main memory reference	100 seconds	Brush teeth
Compress 1K bytes	50 minutes	TV show 📺
Send 2K bytes over 1 Gbps network	5.5 hours	(Brief) Night's sleep 🛌
SSD random read	1.7 days	Weekend
Read 1 MB sequentially from memory	2.9 days	Long weekend
Read 1 MB sequentially from SSD	11.6 days	2 weeks for delivery 📦
Disk seek	16.5 weeks	Semester
Read 1 MB sequentially from disk	7.8 months	Human gestation 🐣
Above two together	1 year	🌍 ☀️
Send packet CA->Netherlands->CA	4.8 years	Ph.D. 🎓

(Care of https://gist.github.com/hellerbarde/2843375)

Engineering vs Theory Efficiency

	Time x1 billion	Like
L1 cache reference	0.5 seconds	Heartbeat 💓
Main memory reference	100 seconds	Brush teeth
SSD random read	1.7 days	Weekend
Disk seek	16.5 weeks	Semester
Send packet CA->Netherlands->CA	4.8 years	Ph.D. 🎓

(Care of https://gist.github.com/hellerbarde/2843375)

Queue Data Structure

```
q.enqueue(8);
q.enqueue(4);
q.dequeue();
```

What do we need to track to maintain a queue with an array list?

8	4								

Queue Data Structure

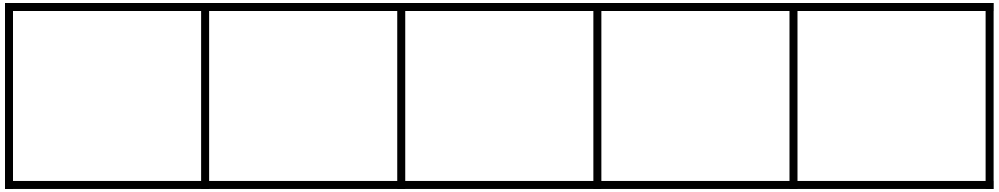
Unlike the array list, it is easier to implement a Queue using unsigned ints

Queue.h

```
#pragma once

template <typename T>
class Queue {
  public:
    void enqueue(T e);
    T dequeue();
    bool isEmpty();

  private:
    T *data_;
    unsigned size_;
    unsigned capacity_;
    unsigned front_;
};
```



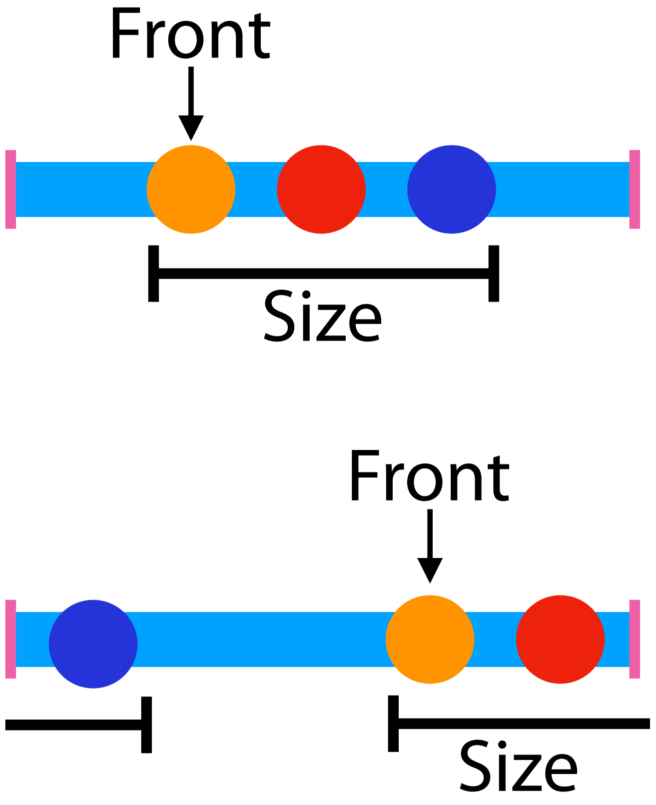
(Circular) Queue Data Structure

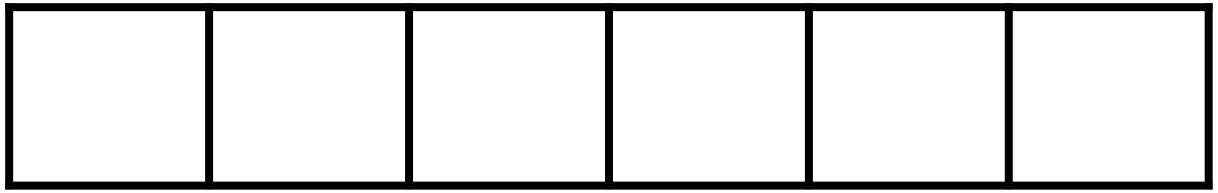
Queue.h

```
#pragma once

template <typename T>
class Queue {
  public:
    void enqueue(T e);
    T dequeue();
    bool isEmpty();

  private:
    T *data_;
    unsigned capacity_;
    unsigned size_;
    unsigned front_;
};
```





Enqueue(D):

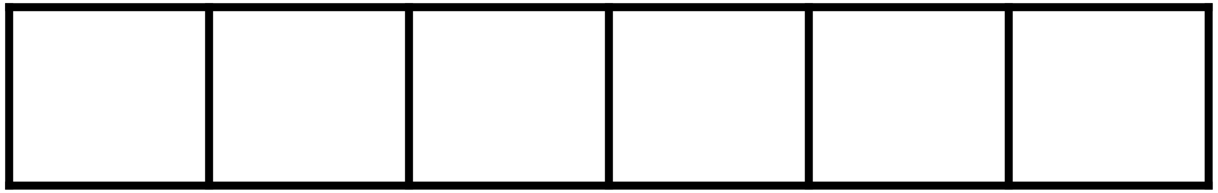
Dequeue():

Size:

Front:

Capacity:

```
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);
```



```
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);
```

Enqueue(D): Insert @ (size+front) % capacity

size++ until size == capacity

Dequeue(): Remove @front

front = (front+1) % capacity

size--

Size:

Front:

Capacity:

			A	B	C

```
Queue<int> q;
…
q.enqueue(D);
q.dequeue();
q.dequeue();
q.dequeue();
q.dequeue();
q.enqueue(E);
```

Enqueue(D): Add data to ‘back’ of queue

Insert D at index **(size+front) % capacity**

size++ (as long as size != capacity)

Dequeue(): Remove data at index front

front = **(front+1) % capacity**

size-- (as long as size != 0)

Size: 3

Front: 3

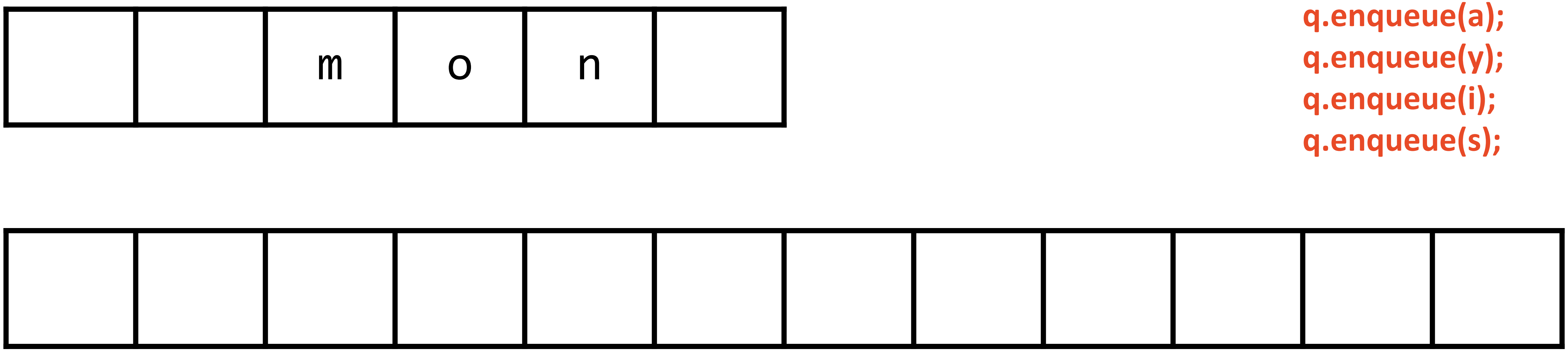
Capacity: 6

Queue Data Structure: Resizing

		m	o	n	

```
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

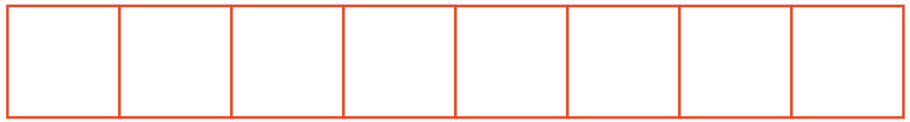
- [Order]:

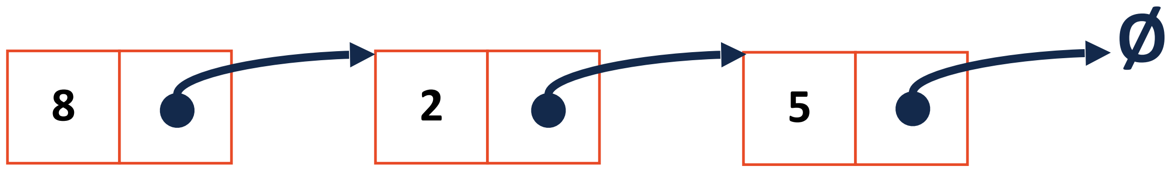
- [Implementation]:

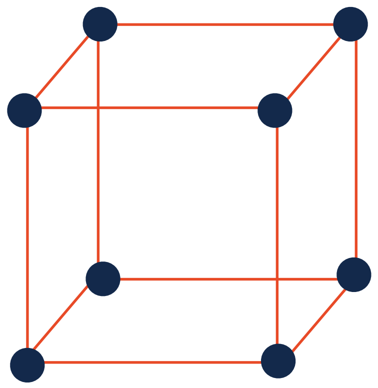
- [Runtime]:

Iterators

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

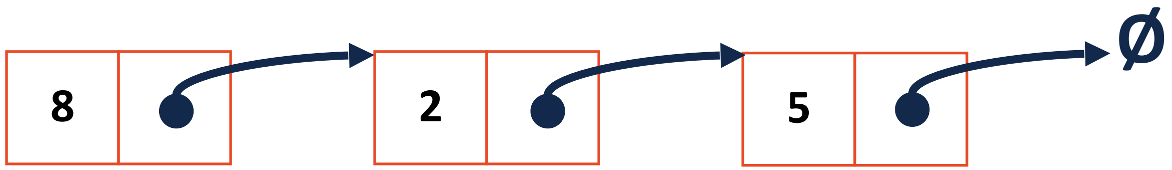


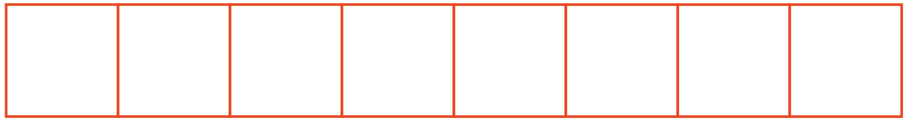


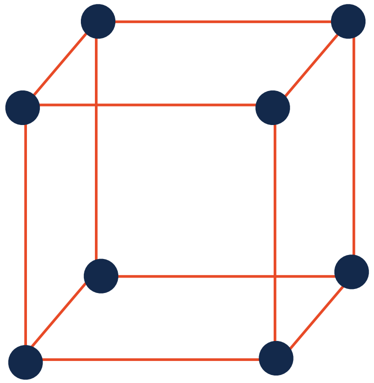


Iterators

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



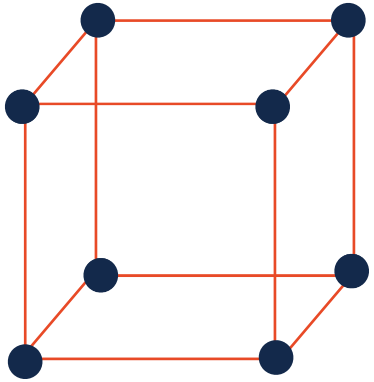




Cur. Location	Cur. Data	Next
`ListNode * curr`		
`unsigned index`		
`Some form of (x, y, z)`		

Iterators

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



```
Cube::Iterator it = myCube.begin();

while (it != myCube.end()) {
    std::cout << *it << " ";
    it++;
}
```

Iterators

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

```
Iterator begin()
```

```
Iterator end()
```

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:

```
template <class T>
class List {

    class ListIterator : public std::iterator<std::bidirectional_iterator_tag, T> {
      public:

        ListIterator& operator++();

        ListIterator& operator--()

        bool operator!=(const ListIterator& rhs);

        const T& operator*();
    };

    ListIterator begin() const;

    ListIterator end() const;
};
```

stlList.cpp

```
#include <list>
#include <string>
#include <iostream>

struct Animal {
  std::string name, food;
  bool big;
  Animal(std::string name = "blob", std::string food = "you", bool big = true) :
    name(name), food(food), big(big) { /* nothing */ }
};

int main() {
  Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
  std::vector<Animal> zoo;

  zoo.push_back(g);
  zoo.push_back(p);   // std::vector's insertAtEnd
  zoo.push_back(b);

  for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
    std::cout << (*it).name << " " << (*it).food << std::endl;
  }

  return 0;
}
```

```
std::vector<Animal> zoo;


/* Full text snippet */

  for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
    std::cout << (*it).name << " " << (*it).food << std::endl;
  }


/* Auto Snippet */

  for ( auto it = zoo.begin(); it != zoo.end; ++it ) {
    std::cout << (*it).name << " " << (*it).food << std::endl;
  }

/* For Each Snippet */

  for ( const Animal & animal : zoo ) {
    std::cout << animal.name << " " << animal.food << std::endl;
  }
```

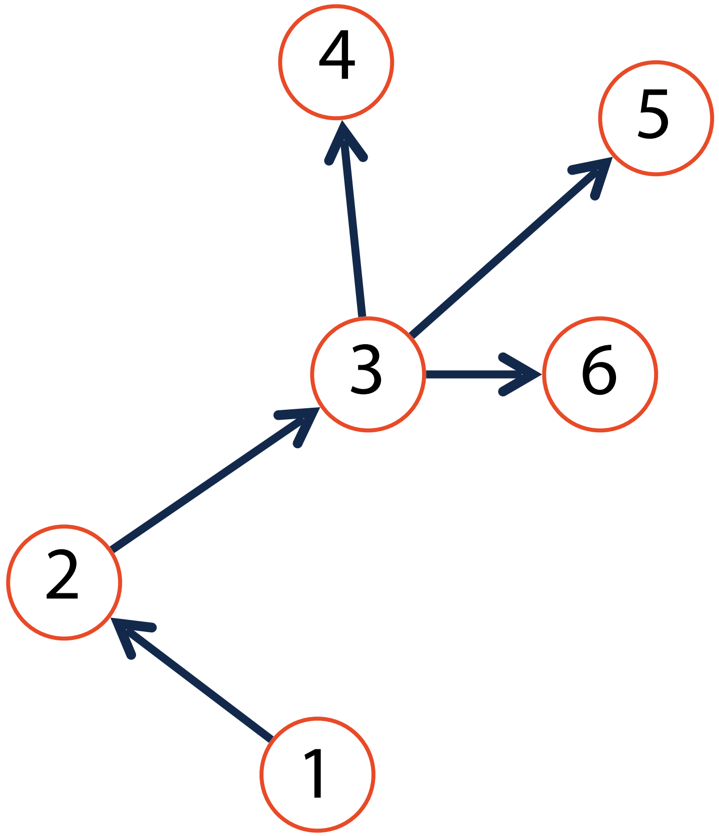
Trees

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

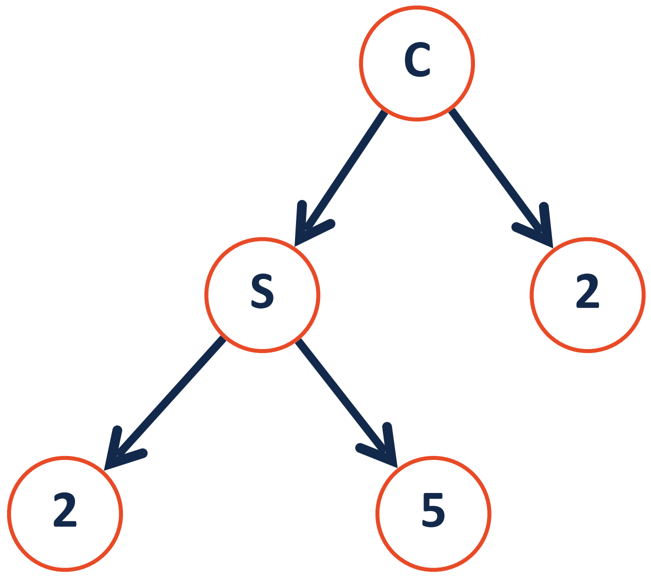


Binary Tree

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

1. $T = \emptyset$

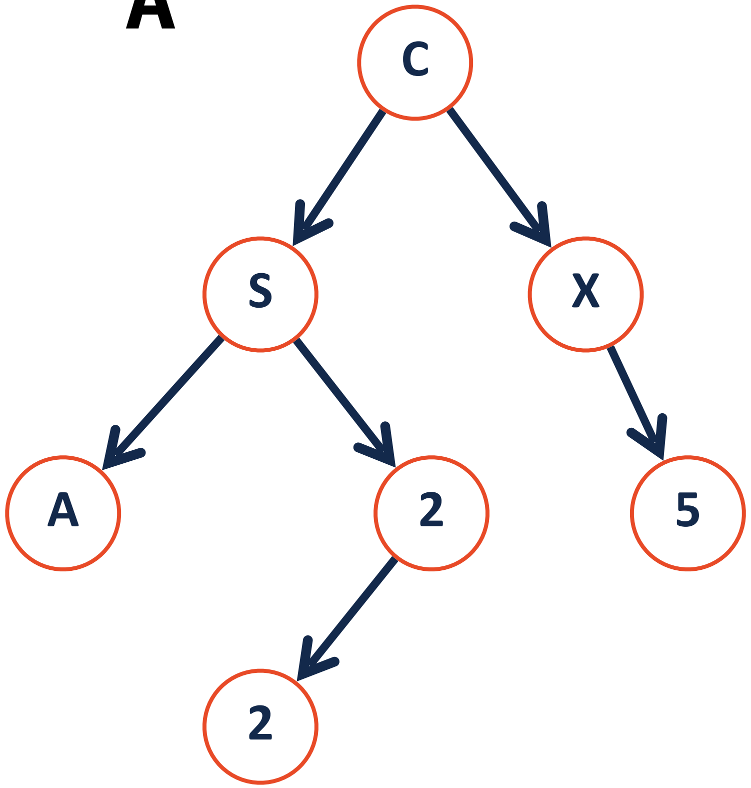
2. $T = (data, T_L, T_R)$



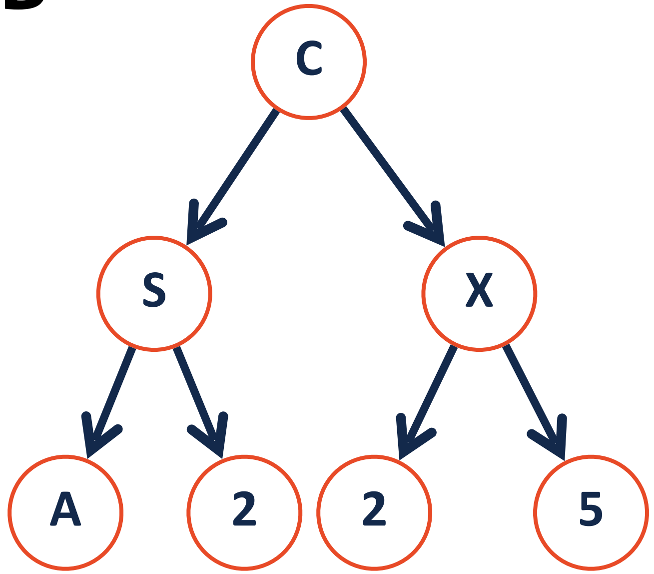
Which of the following are binary trees?

Join Code: 225

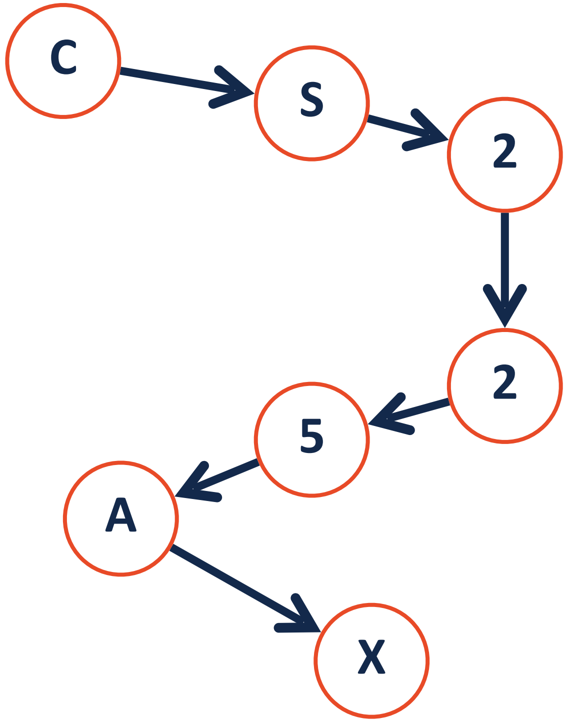
A



B



C



Binary Tree

Lets define additional terminology for different **types** of binary trees!

1.

2.

3.

Binary Tree: full

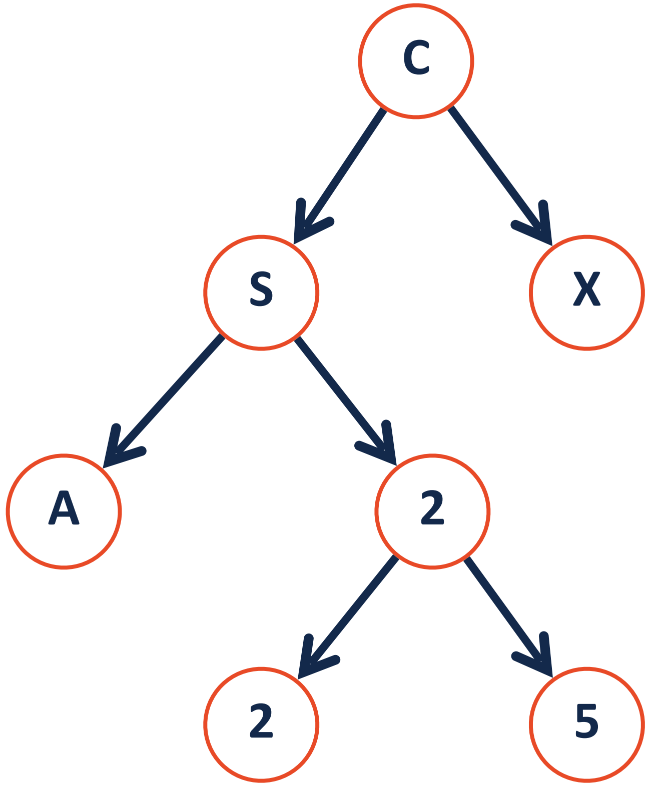
A **full tree** is a binary tree where every node has either 0 or 2 children

A tree **F** is **full** if and only if:

1.

2.

3.



Binary Tree: full

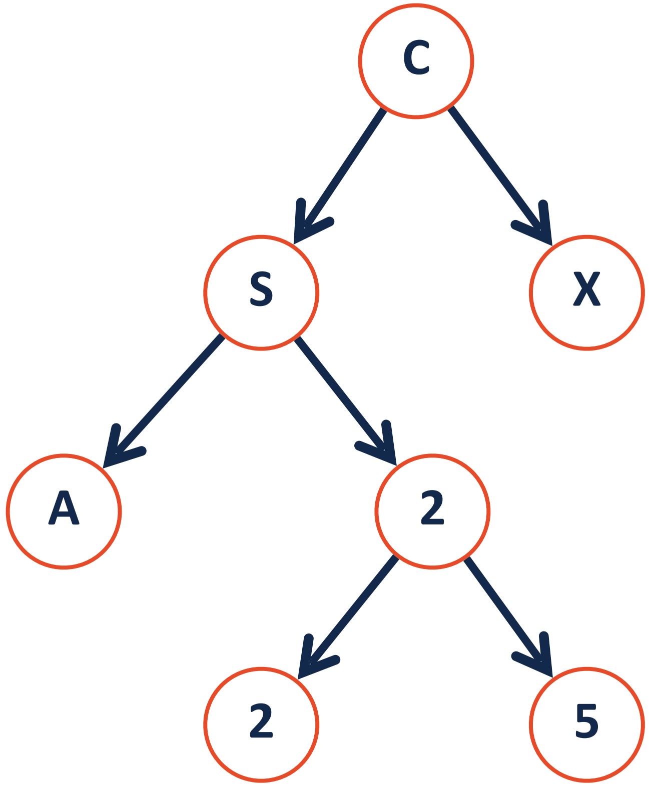
A **full tree** is a binary tree where every node has either 0 or 2 children

A tree **F** is **full** if and only if:

1. $F = \emptyset$

2. $F = (data, \emptyset, \emptyset)$

3. $F = (data, F_l \neq \emptyset, F_r \neq \emptyset)$



Binary Tree: perfect

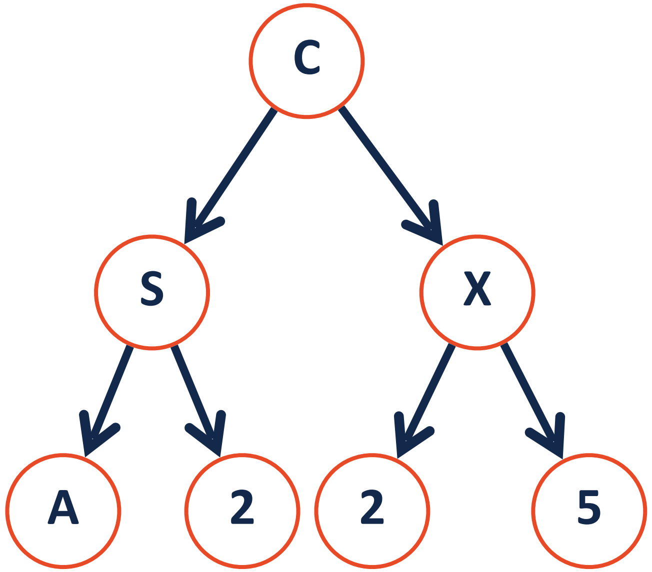
A **perfect tree** is a binary tree where…

Every internal node has 2 children and all leaves are at the same level.

A tree **P** is **perfect** if and only if:

1.

2.



Binary Tree: perfect

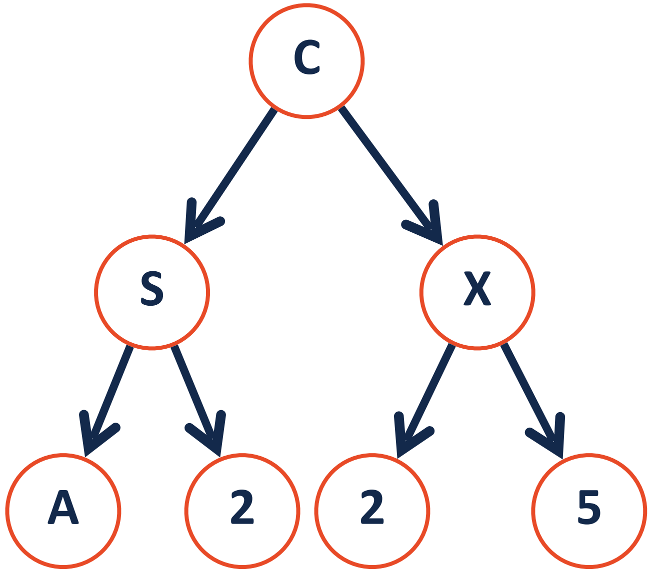
A **perfect tree** is a binary tree where…

Every internal node has 2 children and all leaves are at the same level.

A tree **P** is **perfect** if and only if:

1. $P_h = (data, P_{h-1}, P_{h-1})$

2. $P_0 = (data, \emptyset, \emptyset) \equiv P_{-1} = \emptyset$



Binary Tree: complete

A **complete tree** is a B.T. where…

All levels except the last are completely filled.

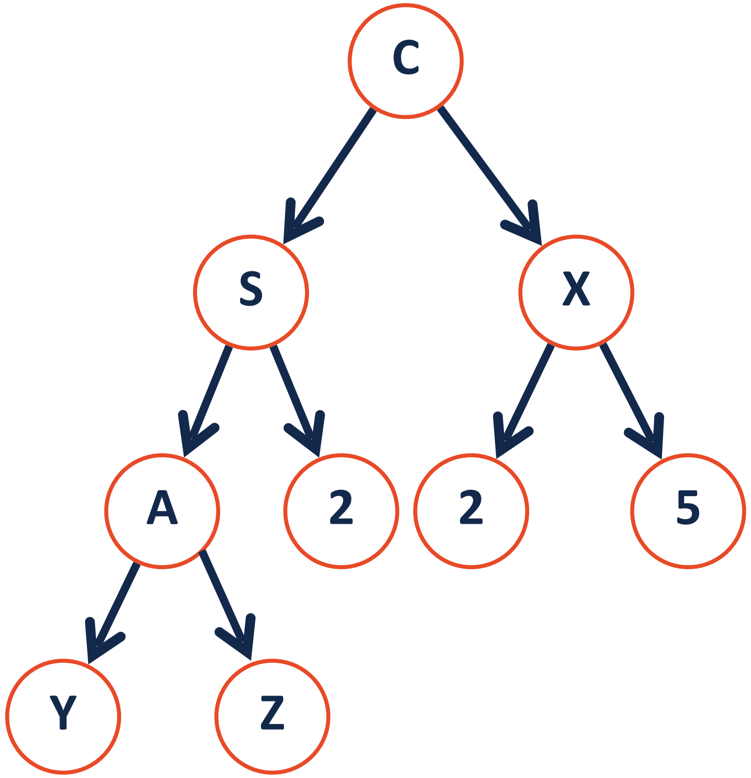
The last level contains at least one node (and is pushed to left)

A tree **C** is **complete** if and only if:

1.

2.

3.



Binary Tree: complete

A **complete tree** is a B.T. where…

All levels except the last are completely filled.

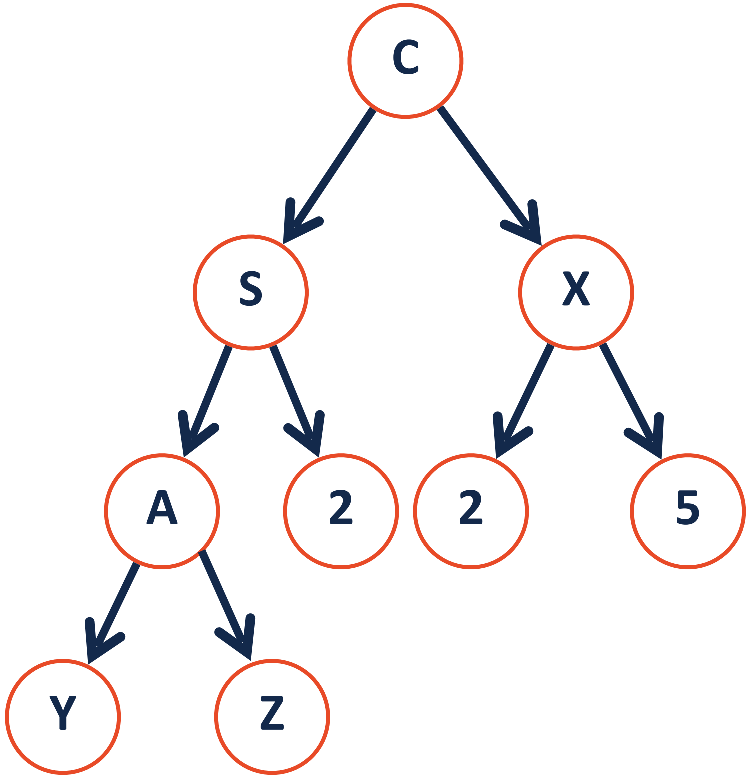
The last level contains at least one node (and is pushed to left)

A tree **C** is **complete** if and only if:

1. $C_h = (data, C_{h-1}, P_{h-2})$

2. $C_h = (data, P_{h-1}, C_{h-1})$

3. $C_{-1} = \emptyset$



Binary Tree

Why do we care?

1. Terminology instantly defines a particular tree structure

2. Understanding how to think 'recursively' is very important.

Binary Tree: Thinking with Types

Is every **full** tree **complete**?

Is every **complete** tree **full**?

Binary Tree: Practicing Proofs

Theorem: If there are **n** objects in our representation of a binary tree, then there are __________ NULL pointers.

Binary Tree: Practicing Proofs

Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

Base Case:

Binary Tree: Practicing Proofs

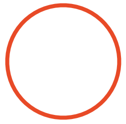
Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

Base Case:

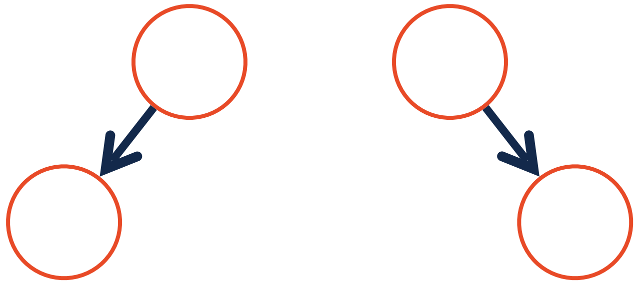
Let F(n) be the max number of NULL pointers in a tree of n nodes

N=0 has one NULL

N=1 has two NULL



N=2 has three NULL



Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

Induction Step:

Theorem: If there are **n** objects in our representation of a binary tree, then there are **n+1** NULL pointers.

IS: Assume claim is true for $|T| \leq k-1$**, prove true for** $|T| = k$

By def, $T = r, T_L, T_R$. Let q be the # of nodes in T_L

Since r exists, $0 \leq q \leq k-1$. By IH, T_L has $q+1$ NULL

All nodes not in r or T_L exist in T_R. So T_R has $k-q-1$ nodes

$k-q-1$ is also smaller than k so by IH, T_R has $k-q$ NULL

Total number of NULL is the sum of T_L and T_R: $q+1+k-q = k+1$

Tree ADT