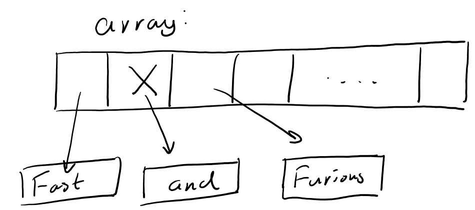
Data Structures vs Abstract Data Types

Friday, September 24, 2021 10:07 AM

Data Structures

- A collection containing:
  - Data values
  - Relationship among data
  - Operations appied to data
- Describes exactly how the data are organized and how tasks are performed

Eg. ArrayList



Counter: ~~0~~ ~~1~~ ~~2~~ ~~3~~ 2

Abstract Data Types

- Defines behavior relative to a user
  - what operations must it have?
- Describes only what needs to be done, not how it's done.

Eg. List

- add elements
- find elements
- remove elements
- size of list
- is ordered

- Fast
- ~~and~~
- Furious

C++ Reference Notes

Friday, September 24, 2021 4:15 PM

Data Types : Signed or Unsigned

- char (1 bytes)
- short (2 bytes)
- int (4 bytes)
- long (8 bytes)
- long long (16 bytes)

} integers

- float (4 bytes)
- double (8 bytes)
- long double (16 bytes)

} floating points

- bool (1 byte)

<u>Strings</u>

- string type.
- mutable (modifiable)
- can only concatenate strings

s.substr(beginIndex, <u>length</u>)

<u>Relational Operators</u>

- ==, !=, <, <=, >, >= work for all types, even classes due to operator overloading.

Variables

Friday, September 24, 2021 4:10 PM

In C++

- variable initialization is <u>not</u> checked

```
int fast;
int furious;
int fastFurious = fast + furious; ← undefined behavior
```

- narrowing is <u>not</u> checked

```
int x = 40000;
short y = x; ← overflow, truncation
```

- variables <u>can</u> be declared outside of a class:

```
int m = 42; ← global variable, poor practice except for constants
class MyClass {
  ...
}
```

Classes, Member Initializer List, Header Files and Source Files

Friday, September 24, 2021 4:15 PM

C++ Classes:

- public/private modifiers are grouped into regions, where all public/private fields/methods are grouped.
- methods do not require implementation at declaration.
- can use a Member Initializer List:

  private: int x; int y;

  Point :: Point (int i, int j) : x(i), y(j) {}

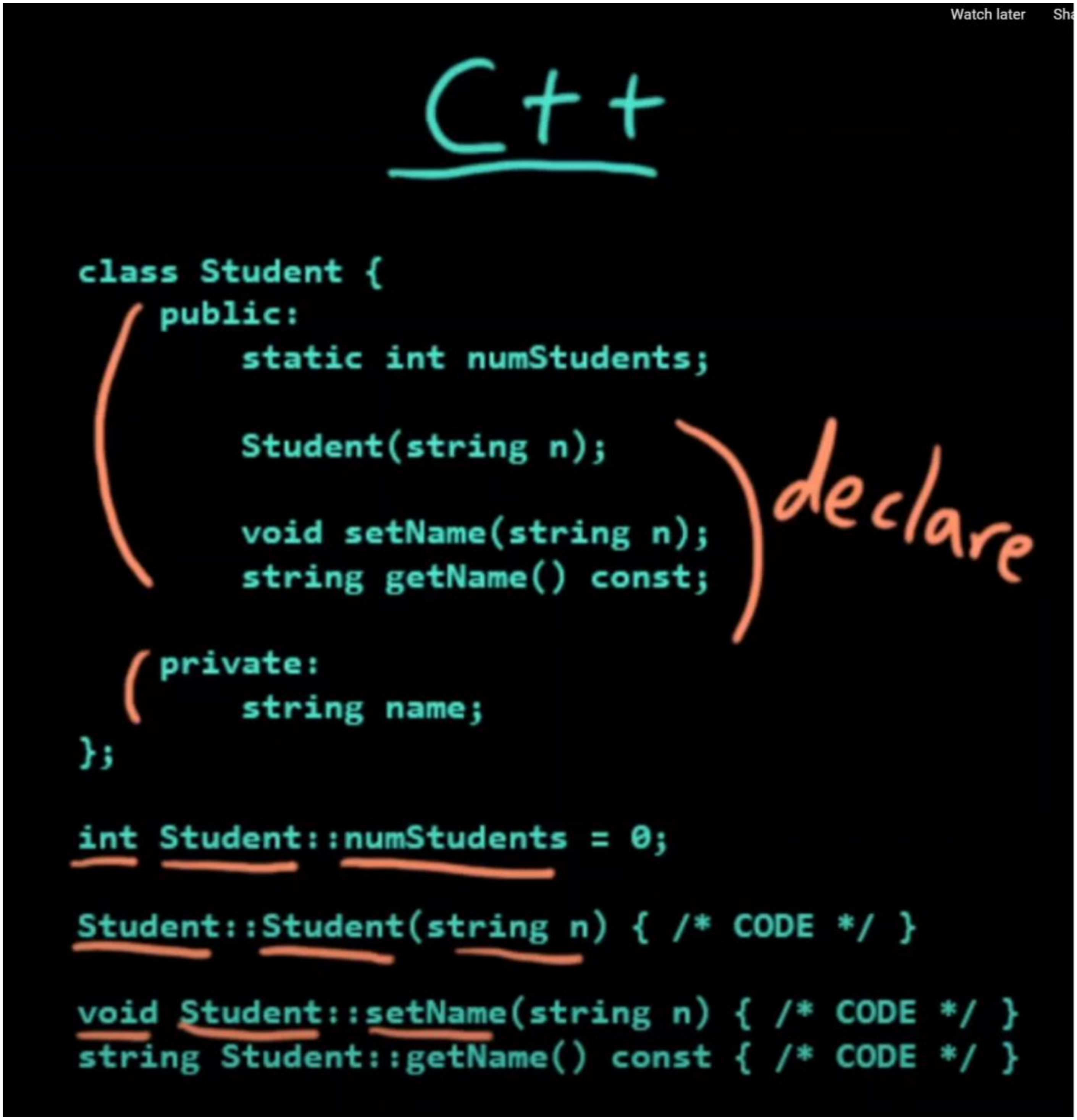
- header files and source files.

  header files store class, field, method declaration

  source files store field, method implementations

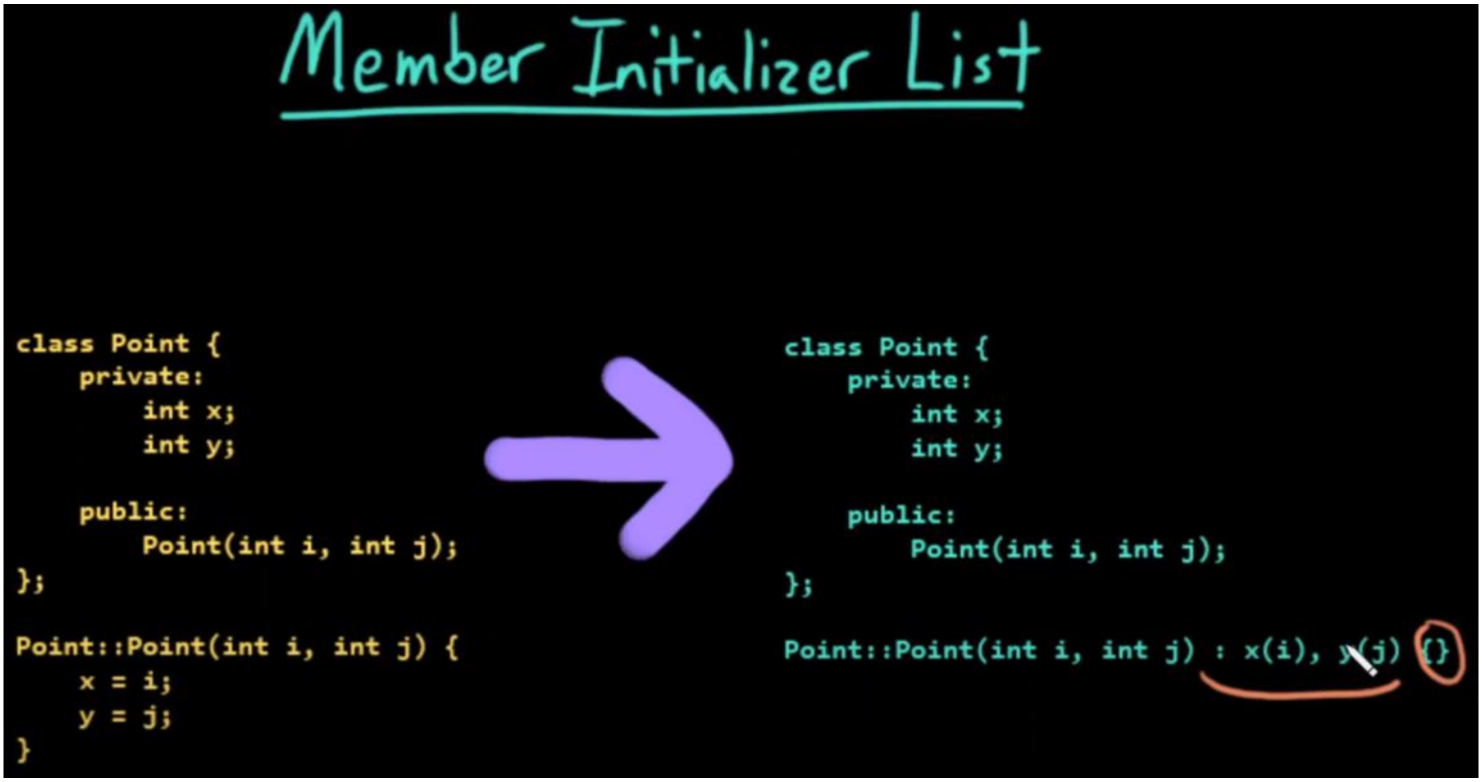
Example Class

Friday, September 24, 2021 4:25 PM



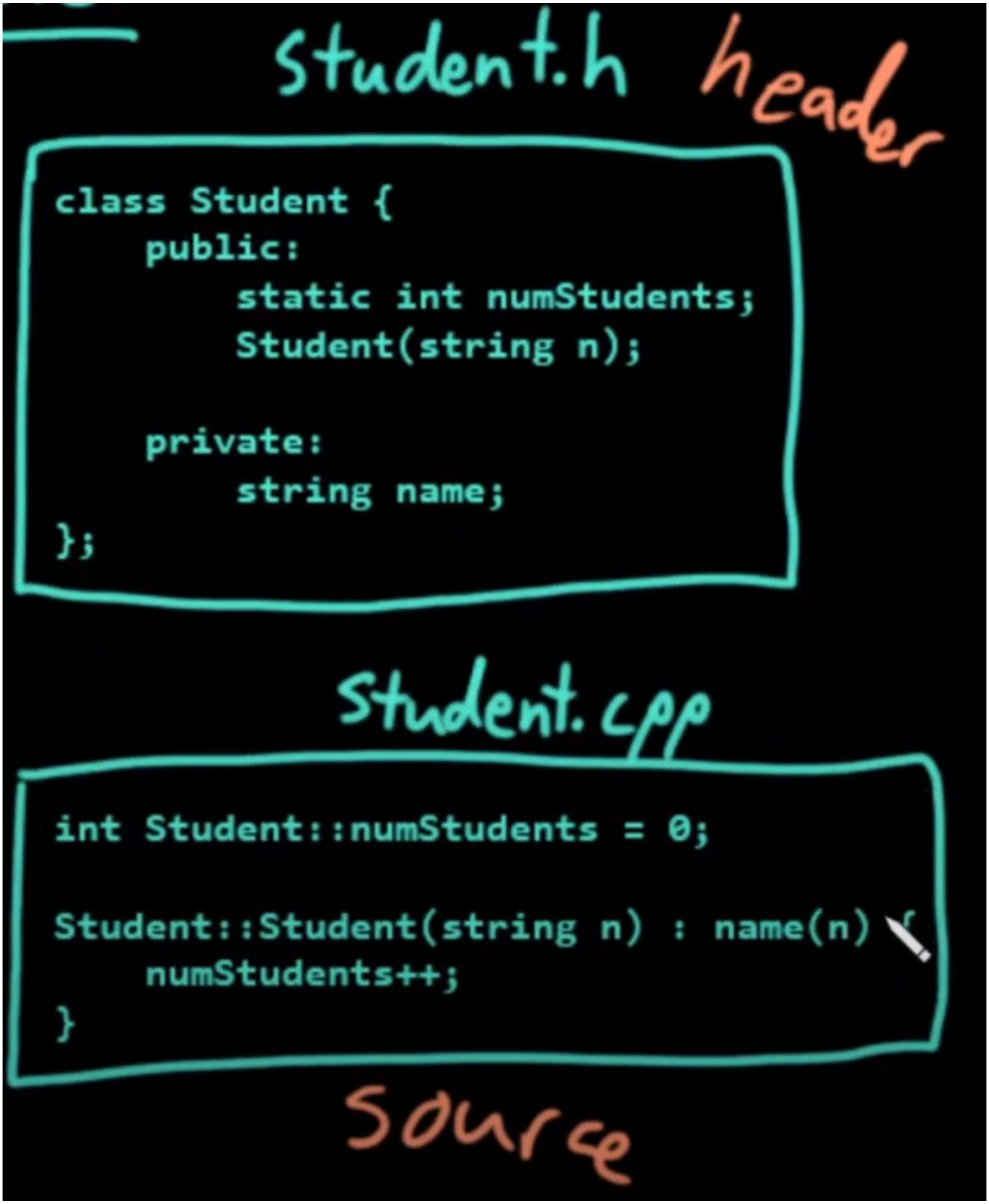
Example Member Initializer List

Friday, September 24, 2021 4:26 PM



Example Header and Source Files

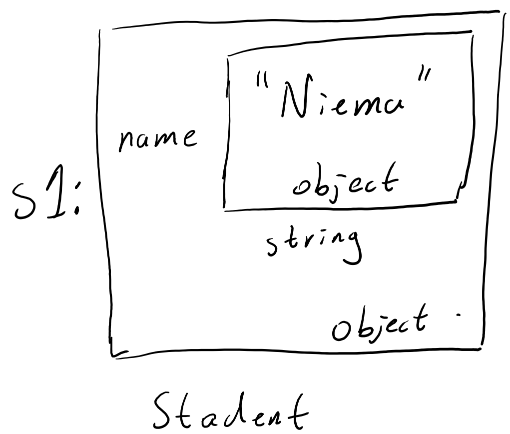
Friday, September 24, 2021 4:26 PM



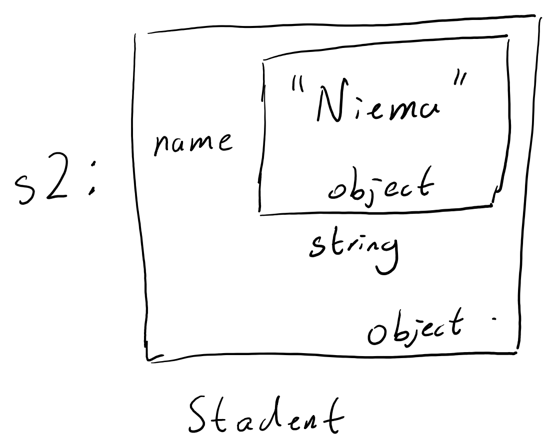
Memory Models (Classes), Reference

Friday, September 24, 2021 4:27 PM

```
Student s1("Niema");
Student s2 = s1;
```



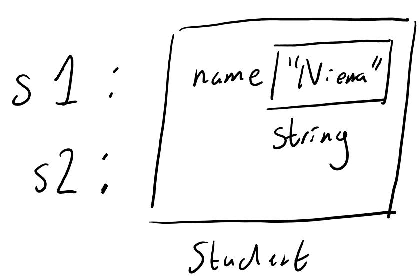
s1 is <u>not a reference</u> to the Student object, <u>it is</u> the object.



s2 is a <u>copy</u> of s1. It is <u>not a reference</u>.

To create <u>a reference</u> use:

```
Student & s2 = s1;
```

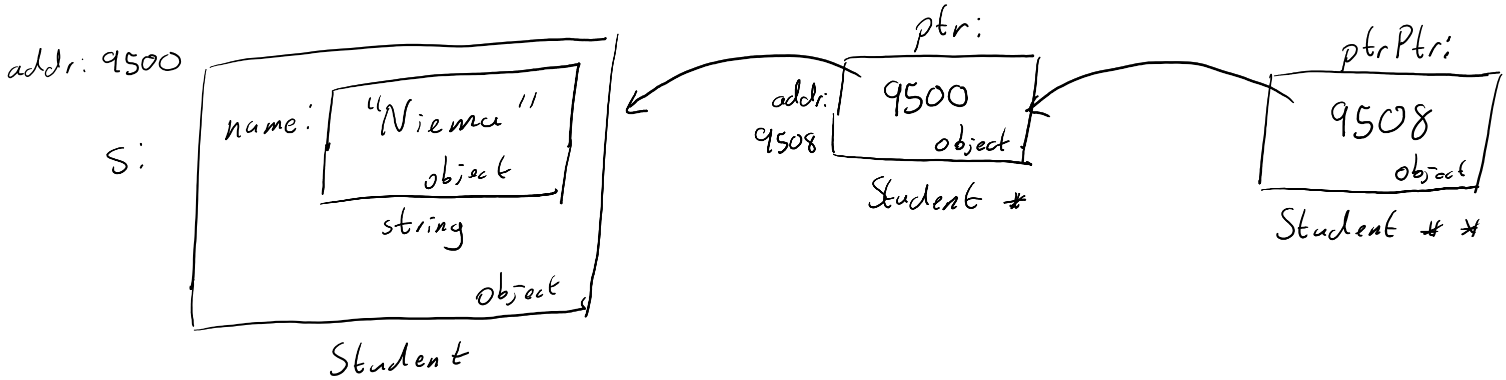


Pointers

Friday, September 24, 2021 4:35 PM

```
Student s = Student("Niema");

Student * ptr = &s;
Student ** ptrPtr = &ptr;
```



To use the value at a pointer:

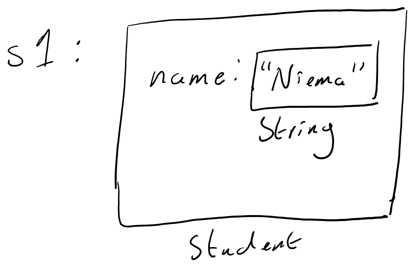
1) Dereference `(*ptr).name`
2) Arrow `ptr -> name`

equivalent

Memory Management (new, delete)

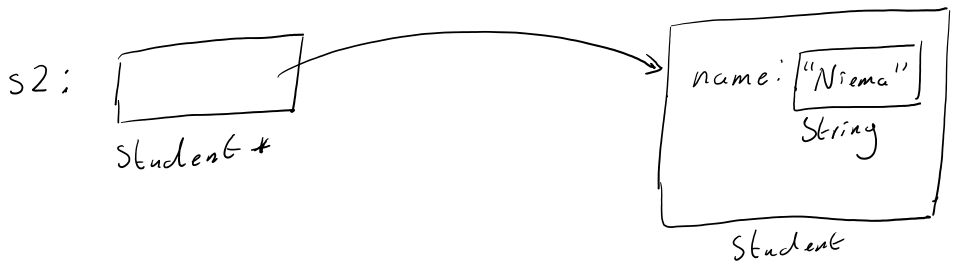
Friday, September 24, 2021 4:40 PM

```
Student s1 = Student("Niema");
Student *s2 = new Student("Ryan");
```



s1 resides on the runtime stack, meaning it is destroyed once outside the scope

- The new keyword is dynamic memory allocation



s2 resides on the heap and will not be automatically deallocated

To deallocate memory, use the delete keyword.

Const(ant) Variables

Friday, September 24, 2021 4:45 PM

- const variables can't be modified
  trying to do so will result in a compiler error.

```
const int a = 42;
int const b = 42;
```

} identical

Const References

Friday, September 24, 2021 4:54 PM

```
int a=42;
const int & ref1 = a;
int const & ref2 = a;
```

equivalent, cannot modify the variable through the reference

can be reassigned.

Const Pointers

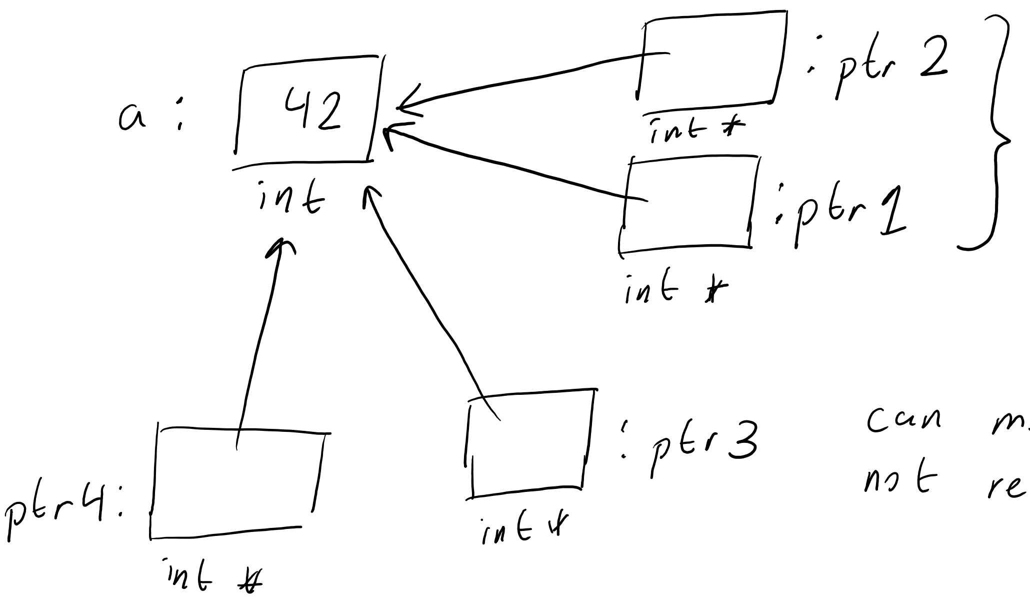
Friday, September 24, 2021 4:49 PM

```
int a = 42;
const int * ptr1 = &a;
int const * ptr2 = &a;
```

} equivalent

```
int * const ptr3 = &a;

const int * const ptr4 = &a;
```



ptr2, ptr1: cannot modify a
reassignable

ptr3: can modify a
not reassignable

ptr4: cannot modify a
cannot reassign

Const Functions

Friday, September 24, 2021 4:56 PM

const functions cannot modify the object

- cannot modify instance variables
- can only call other const functions.

eg: public : string getName() const;

Functions, Global Functions

Friday, September 24, 2021 7:16 PM

- Functions may be defined <u>outside any class</u>, these are <u>global functions</u>
  - main function is a global function, returns an exit code

    ie:

```
int main (){
    ...
}
class MyClass {
    ...
};
```

Pass by Value, Pass by Reference

Friday, September 24, 2021 7:18 PM

Pass by Value

```
void swap (int a, int b){
    int tmp = a;
    a = b;
    b = tmp;
}
```

Note because a,b are passed by value, this method creates no change

Pass by Reference

```
void swap (int &a, int &b){
    int tmp = a;
    a = b;
    b = tmp;
}
```

Note because a,b are passed by reference, this method swaps the values of a,b

Vectors

Friday, September 24, 2021 7:22 PM

C++ Vector

- is a dynamic array

  vector <int> a;

- add elements with:

  a.push_back(42);

  a.push_back(21);

- remove elements with:

  a.pop_back();

- access specific index with:

  a[0];

  Note accessing out of bounds does not throw an error

- assigning one vector to another copies all elements.

  vector <int> b = a;

  b is a seperate vector copy of a.

Input, Output

Friday, September 24, 2021 7:26 PM

Standard output: cout << "message" << endl;

Standard input: cin >> n;

Standard error: cerr << "error" << endl;

To read an entire line from stdin:

getline (cin, message)

// get from cin, store to message

<u>Note</u> cout, cin, cerr, endl are part of the std namespace.

to use a namespace:

using namespace std;

Templates

Friday, September 24, 2021 7:30 PM

- Similar to generics in Java.
- We can specify a template in C++ with:

```
template <typename T> class Node {
        public:
            T const data;
            Node(const T &d): data(d) {}
};
```

- We instantiate a template object with:

```
Node <string> a(s);
Node <int> b(n);
```

Iterators

Friday, September 24, 2021 7:37 PM

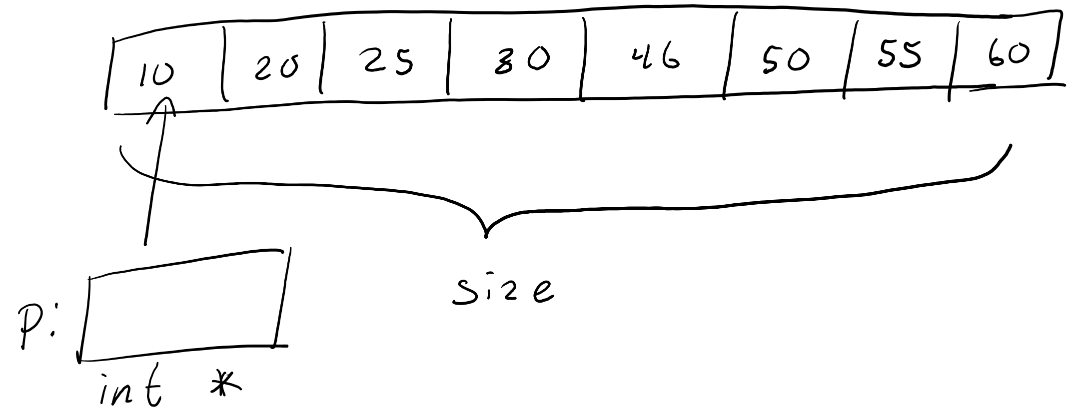
Motivation: given a for each loop like:

```
for ( string name : names) {
    cout << name << endl;
}
```

names could be a vector, set, unordered_set, but with iterators, we don't need to know what specific type names is.

Iterating over an Array with Pointer

Friday, September 24, 2021 7:40 PM



We can iterate over the array with:

```
for (int i = 0; i < size; ++i) {
    cout << *p << endl;
    ++p;
}
```

*p ← dereference

++p ← pointer arithmetic

Steps:

1) Dereference pointer
2) Used data to print
3) Incremented pointer

Iterating through a Vector using Iterators

Friday, September 24, 2021 7:46 PM

- Given some vector:

```
vector<string> names;
```

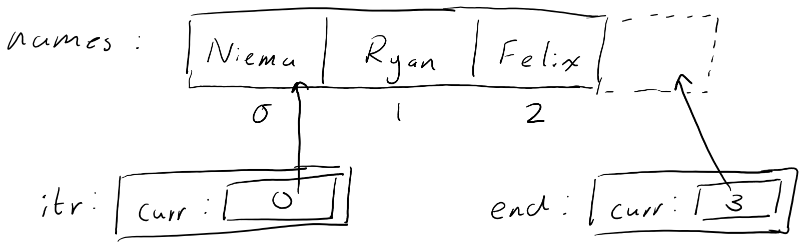
- We can get an iterator to the beginning of the vector with:

```
vector<string>::iterator itr = names.begin();  ← first item
```

and an iterator to the end with:

```
vector<string>::iterator end = names.end();  ← just after the last item
```

- Visualization:



- We can then iterate through the vector with:

```
while (itr != end) {
    cout << *itr << endl;
    ++itr;
}
```

Iterating through a LinkedList using Iterators

Friday, September 24, 2021 7:51 PM

- Similar to iterators of vectors:

```
LinkedList <string> names;
LinkedList <string>:: iterator itr = names.begin();
LinkedList <string>:: iterator end = names.end();
while (itr != end){
        cout << *itr << endl;
        ++itr;
}
```

Custom Iterators

Friday, September 24, 2021 7:59 PM

General idea of any Iterator class:

Overload operators in the Iterator class:

- == true if iterators are pointing to the same item
- != true if iterators are not pointing to the same item
- * (dereference) return a reference to the current data value
- ++ (pre/post increment) move the iterator to the next item

New functions in the DataStructure class:

- begin() return an iterator to the first element
- end() returns iterator to just after the last element

Time Complexity

Sunday, September 26, 2021 6:07 PM

- How <u>fast</u> is my program?
  - ↳ hours?
  - ↳ minutes?
  - ↳ nanoseconds?

  } human time

Problem: human time depends on hardware

- How <u>fast</u> is my algorithm
  - given an input size (n)

Notations of Time Complexity

Sunday, September 26, 2021 6:11 PM

- Big-O : upper bound $f(n) \in O(g(n))$ iff $A \cdot g(n) \geq f(n) \; \forall n$
- Big-Ω : lower bound $f(n) \in \Omega(g(n))$ iff $B \cdot g(n) \leq f(n) \; \forall n$
- Big-Θ : upper & lower bound $f(n) \in \Theta(g(n))$ iff $f(n) \in O(g(n))$ & $f(n) \in \Omega(g(n)) \; \forall n$

  $f(n) \in \Theta(g(n))$ iff $B \cdot g(n) \leq f(n) \leq A \cdot g(n) \; \forall n$

Finding Big-O, common Big-O

Sunday, September 26, 2021 6:18 PM

1) Determine $f(n)$, number of operations given input size n
2) Drop all terms of n except highest term
3) Drop all constant coefficients.

Common Time Complexities.

- $O(1)$ constant time
- $O(\log n)$ logarithmic time
- $O(n)$ linear time
- $O(n \log n)$
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time

$O(n^c)\ \forall c$ polynomial time

- $O(k^n)$ exponential time
- $O(n!)$ factorial time

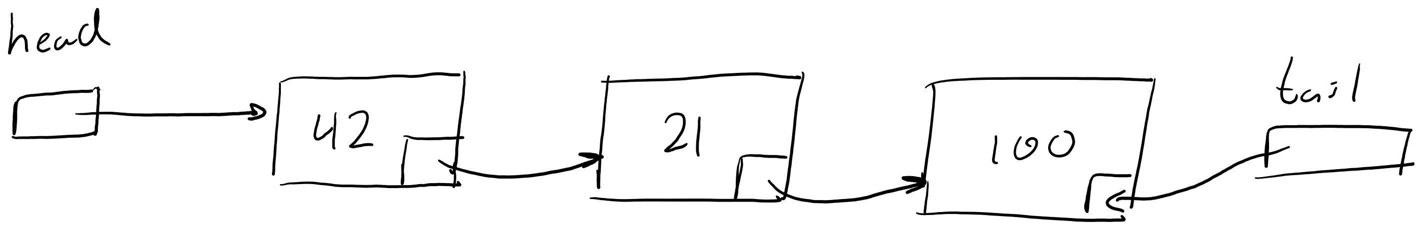
bad time complexities

Space Complexity

Sunday, September 26, 2021 7:54 PM

- We can use Big-O notation to generalize.

Eg: Linked List



space used: $O(n)$

Graphs, Tree

Saturday, October 2, 2021 12:53 PM

<u>Def</u>: Graphs are a collection of <u>Nodes</u> and <u>Edges</u>

- <u>Edges</u> can be directed or undirected:

undirected directed

<u>Def</u>: Trees are Graphs with:

- no undirected cycles (no cycles of undirected edges)
- connected (must exist a path between any two nodes)

Special Trees

Saturday, October 2, 2021 1:04 PM

- Empty / Null Tree:

  0 nodes

  0 edges

- Single Node Tree:

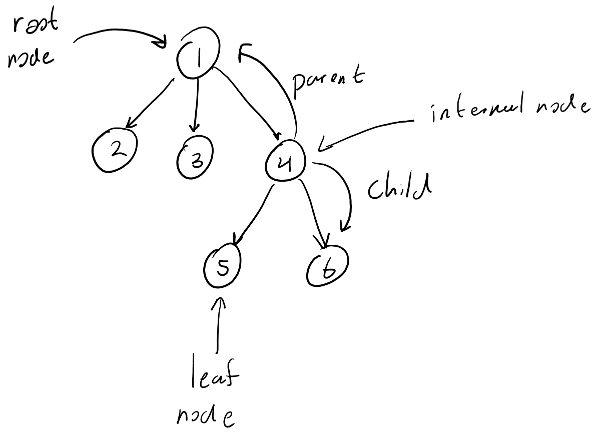
  1 Node

  0 Edges

Rooted vs Unrooted

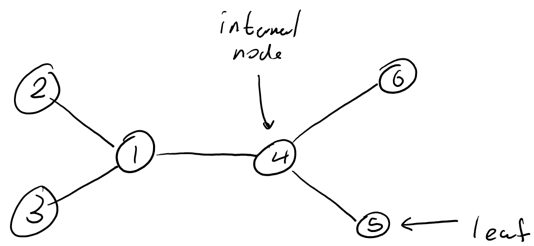
Saturday, October 2, 2021 1:08 PM

Rooted: Heirarchical Structure



root node: has no parent
internal node: has parent and children
leaf node: has no children

Unrooted: No structure



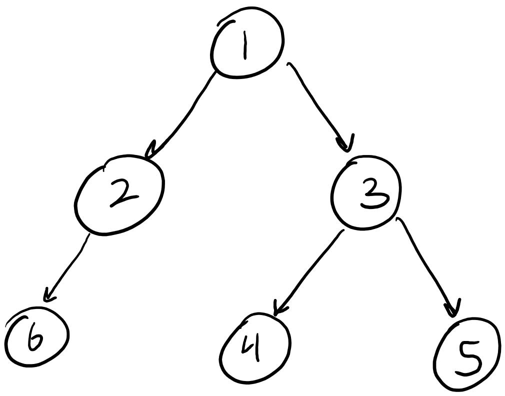
internal node: more than one neighbor
leaf node: only one neighbor

Rooted Binary Trees

Saturday, October 2, 2021 1:12 PM

Rooted: parent-child relationships

Binary: each node has at most 2 children



Tree Traversals

Saturday, October 2, 2021 1:17 PM

Preorder Traversal: Visit, Left, Right

In-Order Traversal: Left, Visit, Right DFS

Post-Order Traversal: Left, Right, Visit

Level-Order Traversal: 1st level (left to right), 2nd level (left to right) ... BFS

Binary Search Tree

Saturday, October 2, 2021 1:46 PM

- Binary Search Trees are:
  - Rooted Binary Tree
  - Every node is <u>larger than all</u> nodes in its <u>left subtree</u>

  Every node is smaller than all nodes in its <u>right subtree</u>

- BST Find Algorithm:
  1) Start at the root
  2) If query == current, return
  3) If query >current, traverse right goto #2
  4) If query <current, traverse left goto #2

     If there are no nodes to tranerse left/right, fail

- BST Insert Algorithm:
  1) Perform Find Algorithm: duplicate if successful
  2) If find doesn't succeed, insert new element at site of failure

- BST Successor Algorithm: Successor: next largest node
  1) If the node has a right tree, traverse right once, then all the way left
  2) Otherwise, travese up until the node was the parent's left child: the parent is the successor

- BST Remove Algorithm:
  1) Perform Find Algorithm: exists if successful

     If Node has no children
     - delete the Node

     If Node has one child
     - replace Node with Node's child

     If Node has two children

- swap node with successor, apply remove algorithm to Node

BST Height

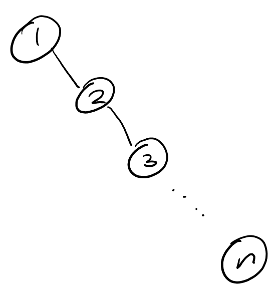
Saturday, October 2, 2021 2:11 PM

Def: Node height: Longest distance from a node to a leaf

Def: Tree height: height of the root of the tree

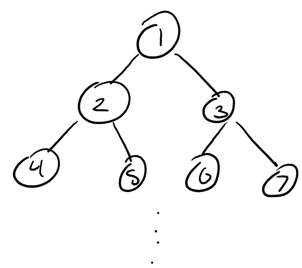
Def: Balance: ratio between tree height and number of elements

Perfectly Unbalanced:



height $= n-1$

Perfectly Balanced:



height $= \log_2(n+1) - 1$

BST Time Complexities

Saturday, October 2, 2021 2:21 PM

- Best vs. Worst vs. Average for Find Algorithm

Best: query is the root $\longrightarrow O(1)$

Worst: perfectly unbalanced, query not found $\rightarrow O(n)$

Average: theoretical expected value $\longrightarrow O(\log(n))$

- Average Case
  - Assumptions:
    1) All n elements are equally likely to be searched
    2) All $n!$ possible insertion orders are equally likely

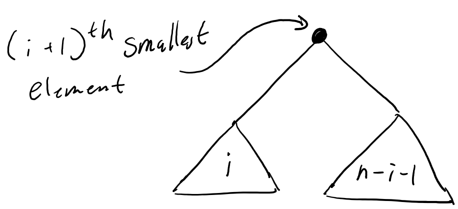
<u>Def</u>: Node <u>depth</u>: number of nodes in the path from Node to root

avg case time complexity = expected # operations to find query

= expected depth

Total expected depth: sum of all node depths, $D(n)$

<u>Def</u>: Every tree can be represented with:



$D(n \mid i)$ = expected total depth w/ n nodes and i nodes in left subtree

$D(i)$ = expected total depth of left subtree

$$D(n \mid i) = D(i) + D(n-i-1) + i + (n-i-1) + 1$$
$$= D(i) + D(n-i-1) + n$$

<u>Def</u>: The expected number of operations for some BST is: $\boxed{\frac{1}{n} D(n)}$

$$D(n) = \sum_{i=0}^{n-1} D(n \mid i) \underbrace{P(I=i)}$$ $\leftarrow$ = probability i nodes in left subtree = $(i+1)^{th}$ smallest node was inserted first $\} \frac{1}{n}$

$$D(n) = \frac{1}{n} \sum_{i=0}^{n-1} D(i) + D(n-i-1) + n$$

Thus:

$$D(n) = \frac{1}{n} \sum_{i=0}^{n-1} D(i) + \sum_{i=0}^{n-1} D(n-i-1) + n$$

$$D(n) = \frac{1}{n} \sum_{i=0}^{n-1} D(i) + \sum_{i=0}^{n-1} D(n-i-1) + n$$

$$= \frac{2}{n} \sum_{i=0}^{n-1} D(i) + n$$

$$n\, D(n) = 2 \sum_{i=0}^{n-1} D(i) + n^2$$

$$-\quad (n-1)\, D(n-1) = 2 \sum_{i=0}^{n-2} D(i) + (n-1)^2$$

$$n\, D(n) - (n-1) D(n-1) = 2\, D(n-1) + n^2 - (n-1)^2$$

$$n\, D(n) = (n+1) D(n-1) + 2n - 1$$

$D(0) = 0$

$$\frac{D(n)}{n+1} = \frac{D(n-1)}{n} + \frac{2n-1}{n(n+1)} \xrightarrow{n=1} \frac{D(1)}{2} = \frac{D(0)}{1} + \frac{2-1}{(1)(2)} \rightarrow D(1) = 1$$

$$\frac{D(n)}{n+1} = \sum_{i=1}^{n} \frac{2i-1}{i(i+1)} = \sum_{i=1}^{n} \frac{2}{i+1} - \sum_{i=1}^{n} \frac{1}{i(i+1)} \rightarrow \text{(expanded)}$$

$$D(n) = 2(n+1) \sum_{i=1}^{n} \frac{1}{i} - 3n$$

We can approximate this to $2\frac{n+1}{n} \ln(n) - 3 \approx 1.386 \log_2(n)$

which is $O(\log(n))$

Treeps

Saturday, October 9, 2021 3:32 PM

- Treep = Tree + Heap stores objects of (key, priority)

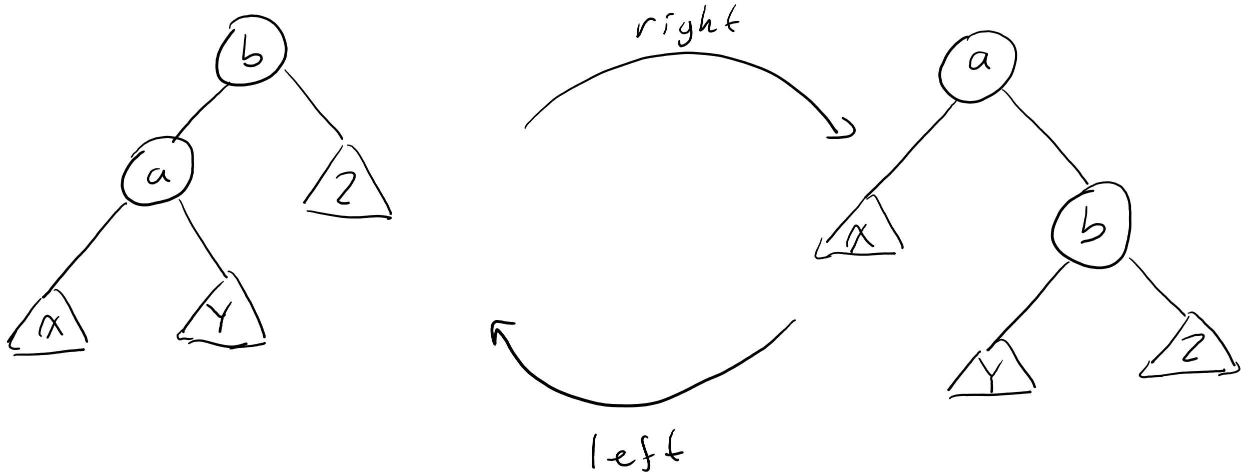
1) BST properties wrt <u>keys</u> for any given node:
   - larger than all keys in left subtree
   smaller than all keys in right subtree

2) Heap property wrt <u>priorities</u> for any given node:
   - larger than all priorities <u>below</u>

AVL Rotation

Saturday, October 9, 2021 3:36 PM

- Given some tree: AVL right/left rotations are:



Note: BST properties are maintained

Treep Insertion

Saturday, October 9, 2021 3:47 PM

Treep Insertion Algorithm:

1) Insert via <u>BST insert</u> algorithm wrt <u>keys</u>

2) Use <u>AVL Rotations</u> to "bubble up" to fix priorities

   - If the inserted node has a higher priority than parent
      - If inserted node is left child → AVL right
      - If inserted node is right child → AVL left
   - repeat until inserted node has lower priority than parent

Random Search Trees

Saturday, October 9, 2021 3:57 PM

RSTs are implemented with Treaps:

- use elements (values) as <u>keys</u>
- Randomly generate <u>priorities</u> maintaining Heap property

Advantage: addresses the assumption of BSTs that each element has an equal chance of being searched, creates a more balanced tree.

Runtime: Worst: $O(n)$ Average: $O(\log(n))$

AVL Trees

Saturday, October 9, 2021 4:12 PM

- AVL Tree : BST with balancing
  - Balance factor = (Height of right subtree) - (Height of left subtree)
  - AVL Tree: BST where every node has BF of (-1, 0, 1)

AVL Time Complexities

Saturday, October 9, 2021 4:18 PM

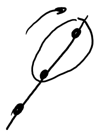
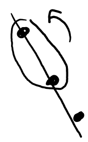
Find algorithm: Best = O(1) Average = O(log(n)) <u>Worst = O(log(n))</u>

Note: AVL Trees have better worst time complexity than BST

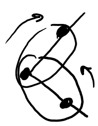
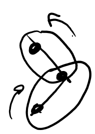
AVL Insertion Algorithm

Saturday, October 9, 2021 4:31 PM

- AVL Insertion Algorithm:
  - Perform regular BST insertion
  - Update Balance Factors
  - Fix broken Balance Factors using AVL Rotations
    - If out of balance node, child, grand child are:

 or  then single rotation

but if

 or  then double rotation starting on the child-grand child then node-child

Red-Black Tree, RB vs AVL

Saturday, October 9, 2021 5:00 PM

- RB Tree = BST with balancing
  - All nodes must be red or black
  - The root must be black
  - If a node is red, all of its children must be black
  - For every node, every possible path to a null reference must have the same # of black nodes
  - null references are black

- RB vs AVL:

- RB Trees are not always AVL Trees, all AVL Trees could be RB Trees

RB Time Complexities

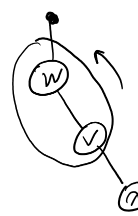
Saturday, October 9, 2021 5:09 PM

- Like AVL Trees, RB Trees have worst case time complexity for find algorithm of $O(\log(n))$

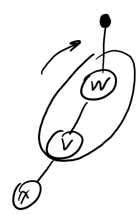
RB Insertion Algorithm

Saturday, October 9, 2021 5:11 PM

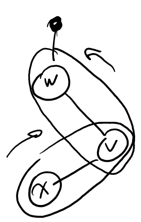
- Case 1: Empty Tree
  - Insert the new node as root
  - Color it black
- General steps for inserting in non-empty tree
  - Insert using regular BST
    - during traversal, if there is ever a black node w/ two red children, swap all colors red ↔ black for node and two children
    - insert node and color new node red
  - potentially fix tree for Red-Black tree properties
- Case 2: Child of Black Node
  - no conflicts, done!
- Case 3: Child of Red Node, Straight Line



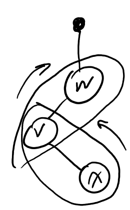
or



  - perform insertion
  - rotate parent-grandparent
  - swap color of parent-grandparent red ↔ black
- Case 4: Child of Red Node, Crooked



or



  - perform insertion
  - rotate node-parent then (parent - grandparent) ← case 3
  - swap color of node-parent

Set, Map ADTs

Friday, October 15, 2021 11:21 PM

- Set: Store multiple elements (keys)
  - find(x): true if x exists in the set, false otherwise
  - insert(x): add x to the set
  - remove(x): remove x from the set
- Map: Store multiple (key, value) pairs
  - get(k): return value associated with key k
  - put(k,v): add the key value pair
  - remove(k): remove the key and associated value

Implementing Set/Map ADTs

Friday, October 15, 2021 11:25 PM

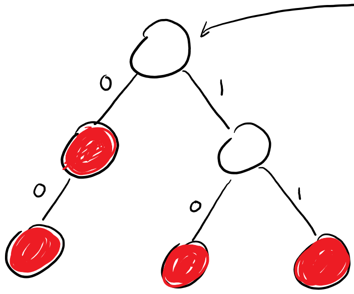
	Find/Remove	Insert	Notes
Unsorted Linked List :	$O(n)$	$O(1)$	
Sorted Linked List :	$O(n)$	$O(n)$	Can iterate in sorted order
Unsorted Array List :	$O(n)$	$O(1)$	
Sorted Array List :	$O(\log n)$	$O(n)$	can iterate in sorted order
Self Balancing BST :	$O(\log n)$	$O(\log n)$	can iterate in sorted order
Hash Table :	$O(1)$	$O(1)$	need to calculate $O(k)$ hash

Tries, Binary Tries

Friday, October 15, 2021 11:31 PM

Def Trie is a tree structure in which elements are represented by paths

Ex Binary Trie :



nodes have no content

binary trie

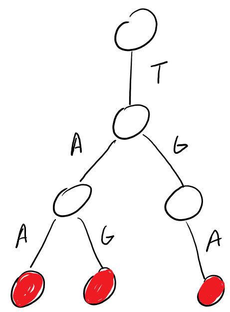
red nodes are word nodes : 0, 00, 10, 11

Multiway Tries, Algorithms (Find, Insert, Remove)

Friday, October 15, 2021 11:35 PM

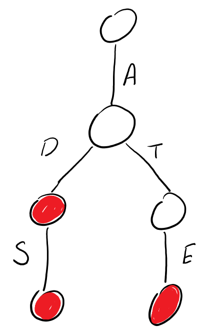
<u>Def</u> Multiway Tries are Tries which nodes can have more than 2 children

DNA Trie



words: TAA, TAG, TGA

English Trie



words: AD, ADS, ATE

Algorithms:

Initialize: 1) Create root node non-word

Insert : 1) Start at the root

2) If there exists an edge for each letter, traverse the edge
Else create the edge and next node

3) Repeat complete, mark last letter's node as word node

Find : 1) Follow edges corresponding with each letter until word node or run out of edges

Remove : 1) Perform find algorithm

2) Mark node as non-word if it exists

Multiway Tries Time, Space Complexity

Friday, October 15, 2021 11:42 PM

Given a MWT and : n = number of words, k = length of longest word

Time Complexity:

Insert $O(k)$

Find $O(k)$

Remove $O(k)$

Given a MWT and: $|\Sigma|$ = length of alphabet, n = number of words, k = length of longest word

For each node, we can have $|\Sigma|$ children

There can be n^k possible words

Space occupied is $O(|\Sigma|^{k+1})$

Significant Algorithms on Multiway Tries

Friday, October 15, 2021 11:58 PM

- Iterate in ascending order → preorder traversal
- Iterate in descending order → postorder traversal
- Iterate in order of length → level order traversal
- Autocomplete: Given some prefix, return all words in subtree → find, traverse from found node

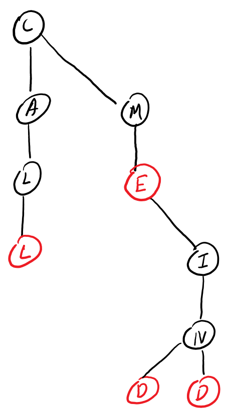
Ternary Search Trees

Saturday, October 16, 2021 12:06 AM

- BST: $O(k \log n)$, memory efficient
- MWT: $O(k)$, memory inefficient
- TST: Somewhere in between

Def: Ternary Search Trees are trees with each node having some value, and also is or is not a word node, when traversing a node is only used if the traversal brings it down.

Ex:



words: call
me
mind
mid

TST Find

Saturday, October 16, 2021 12:13 AM

Algorithm: 1) current node = root
current letter = first letter of query

2) If c > current node :
traverse right

If c < current node :
traverse left

If c is last letter and current node is word node
success!

Else :
traverse down the middle child and go to next letter in query

TST Insert, Remove

Saturday, October 16, 2021 12:17 AM

Insert Algorithm:

- Traverse using comparisons with the query, if it does not exist create the node
- Make the last node ended at a word node

Remove Algorithm:

- Perform find algorithm
- Make the last node not a word node

TST Time Complexity

Saturday, October 16, 2021 12:30 AM

Given a TST with n words, k longest word, $|\Sigma|$ alphabet size

$O(n)$ worst case

$O(\log(n))$ avg. case

Hash Functions, Tables, Maps

Saturday, October 23, 2021 9:43 PM

Hash Function:

Input: Object

Output: Integer representation of x ← <u>hash value</u>

Property of equality: if $x = y$, then $h(x) = h(y)$

Property of inequality: if $x \neq y$, then $h(x) \neq h(y)$ ← not nessasary

Hash Table: Stores elements by hash value : implements <u>Set ADT</u>

0	1	2	3	4
A	G	C		

Size = 5

A : $h(A) = 65 \% 5 = 0$

C : $h(C) = 67 \% 5 = 2$

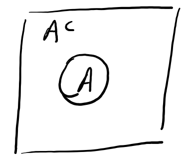
G : $h(G) = 71 \% = 1$

Hash Map: Hash table where each element is a key-value pair : implements <u>Map ADT</u>

Collisions, Probability

Saturday, October 23, 2021 10:02 PM

- Basic Probability Review



$$P(A) + P(A^c) = 1 \rightarrow P(A) = 1 - P(A^c)$$

$P(A \text{ and } B) = P(A) \cdot P(B)$ for independent events A, B

$P(A \text{ or } B) = P(A) + P(B)$ for mutually exclusive A, B

- Propability of At Least 1 collision Given N elements, M slots:

$$P_{N,M}(\geq \text{collision}) = 1 - P_{N,M}(0 \text{ collisions})$$

$$P_{N,M}(0 \text{ collisions}) = 1 \cdot \frac{M-1}{M} \cdot \frac{M-2}{M} \cdot \ldots \cdot \frac{M-N+1}{M}$$

- Load Factor:

$$LF = \frac{N}{M} = \alpha$$

$$\text{expected collisions} = \sum_{i=1}^{N-1} \frac{1}{m} = \frac{N(N-1)}{2M} = 1 \rightarrow N = \sqrt{2M}, \quad M = O(N^2)$$

$$= \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

Collision Resolution

Saturday, October 23, 2021 10:47 PM

- Open Addressing (Linear Probing) → chance of future collisions get more likely
  - Attempt to insert as normal
    If collision: iterate through array until an empty slot is found
  - Attempt to find as normal
    If occupied: iterate through array until query is found or empty slot is found
- Double Hashing:
  - Create a new hash function, so that when we try to insert, we use the next hash function
  - Finding an element: hash once, if not query, hash again, if not query then it does not exist
- Closed Addressing (Separate Chaining) → chance of future collisions are always the same
  - Make each array index a linked list. During each insertion search through to check
  - Find same as insertion.

Bloom Filters

Saturday, October 23, 2021 11:24 PM

Def Probabilistic datastructure, memory-efficient
- ↳ no false negatives
- ↳ possible false positives

Bloom Filters are hash tables of booleans, multiple hash functions for double hashing

For each item:

insert: set each index from every hash function to true

find: if all index from every hash function is true, then it likely exists

Design:

Assume 1) Each hash function uniformly distributes across the array

2) Each insertion is independent

Given: k = # hash functions m = # bits n = # insertions

Probability of false-positive in the Bloom Filter

Probability that a specific bit is set to true: $\approx 1 - e^{-\frac{kn}{m}}$

$P(\text{false-positive}) = \varepsilon \approx (1 - e^{-\frac{kn}{m}})^k$

Thus: $k = \frac{m}{n}\ln(2) = \lceil -\log_2(\varepsilon) \rceil$, $m = \dfrac{-n\ln(\varepsilon)}{\ln(2)^2}$

1) Estimate n

2) Pick ε that is appropriate

3) $k = -\log_2(\varepsilon)$

4) $m = \dfrac{-n\ln(\varepsilon)}{\ln(2)^2}$

Count-Min Sketch

Saturday, October 23, 2021 11:46 PM

Def Probabilistic data structure, memory efficient

Used to count number of occurrences. provides upper-bound on counts

	0	1	2	3	4
h_1	0	0	0	0	0
h_2	0	0	0	0	0
h_3	0	0	0	0	0

$k \begin{cases} h_1 \\ h_2 \\ h_3 \end{cases}$

insert: for each element, compute the hash functions, increment corresponding row's count at the corresponding index.

find: compute all hash functions, the maximum possible is the minimum individual count

Design:

Given n = # elements in stream, c_x the true count of element x, $\hat{c_x}$ the estimated count $c_x \leq \hat{c_x}$

$\hat{c_x} \leq c_x + \varepsilon n$ with probability $1 - \delta$

1) Guess value of n
2) Pick upper bound $\hat{c}_x - c_x \leq \varepsilon n$ for ε
3) Pick probability of being in range δ
4) $m = \lceil \frac{e}{\varepsilon} \rceil$ where m = num columns and $k = \lceil \ln(\frac{1}{\delta}) \rceil$

String Sequencing

Saturday, October 30, 2021 6:37 PM

Problem: Matching database of short strings to a long string query

$$m\left\{\begin{array}{l}\text{ACC} \leftarrow \text{database}\\ \text{ATC}\\ \text{CAT}\\ \underbrace{\text{GCC}}_{k}\end{array}\right.$$

$$\overbrace{\text{GCATCG}}^{n}$$

↑ query

Time complexity: $O(n \cdot k \cdot m)$

Aho-Corasick Automaton

Saturday, October 30, 2021 4:34 PM

Solution: Aho-Corasick Automaton

1) Construct the multiway trie relating to the database
2) for every node, connect a failure link to the longest suffix that exists in MWT excluding itself. CATG has suffixe ATG, TG, G, ϕ
   root node links only to itself
3) for every node, connect a dictionary link to the first word node encountered after traversing the failure links until reaching a root node;

for each query: traverse the MWT given the query. If we reach a word node, then it exists in the query
If there is no further traversal, traverse the failure link
If there is a dictionary link on any node, then we encountered that word.

Shortcut for dictionary links: subwords of a word which are also in the automaton get connected to the word. the last letter of longest subword connects to last letter of word in automaton.

Suffix Arrays

Saturday, October 30, 2021 6:37 PM

Solution: swap database and queries

database:

$\underbrace{GCATCGC}_{n}$

queries:

ACC
ATC
GC

$\left.\begin{matrix} ACC \\ ATC \\ GC \end{matrix}\right\} k$

suffix array:

0: GCATCGC		2 : ATCGC
1: CATCGC		6 : C
2: ATCGC		1 : CATCGC
3: TCGC	sort	4 : CGC
4: CGC		5 : GC
5: GC		0 : GCATCGC
6: C		3 : TCGC

for each query: we search the suffix array by

1) Binary search for the lowest index occurance of the query, prefix matches are counted
2) Binary search for the highest index occurance of the query, prefix matches are counted
3) every index between low and high are instances of query

time complexity: sort: $O(n \log n) \rightarrow$ closer to $O(n^2 \log n)$

search: $O(k \cdot \log n)$ per query sequence

space complexity: $O(n^2)$

Burrows - Wheeler Transform

Saturday, October 30, 2021 7:01 PM

Solution: Burrows - Wheeler Transform

database

$\underbrace{\text{BANANA\$}}_{n}$ ← termination character

queries

$\left.\begin{array}{l}\text{AN}\\ \text{NANA}\\ \text{BAN}\\ \underbrace{\text{NA}}_{k}\end{array}\right\} m$

To create the BWT:

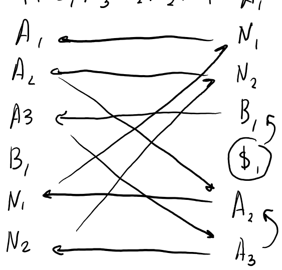
1) create all rotations of the database
2) sort rotations alphabetically, assume termination character is the smallest letter
3) BWT is the last column of all the rotations

Note: BWT is invertible:

To create string from BWT:

1) the first character are the characters in the BWT in alphabetical order
2) the number of times each character in each column are the same across all columns
3) Starting from the termination character the character above each character must have followed it, jumping from column to collumn:

$\$_1 B_1 A_3 N_2 A_2 N_1$ A_1



Searching a BWT for pattern matching:

Given a BWT: we can match queries:

i	first	last	Last → First
0	$\$_1$	A_1	1
1	A_1	N_1	5
2	A_2	N_2	6
3	A_2	B_1	4
4	B_1	\$	0
5	N_1	A_2	2
6	N_2	A_3	3

1) start with top and bottom rows
2) search for first and last instances of each character looking at last column and using the Last → First mapping
3) The top bottom pointers contain all instances of the query
4) convert positions by

0 N_2 A_3 3

4) convert positions by

Encoding/Decoding, Entropy

Saturday, November 6, 2021 5:44 PM

Goal Given some message, we can convert it between raw and encoded formats:

$$\begin{tikzcd} \text{message} \arrow[r, bend left, "\text{encode}"] & \text{encoding} \arrow[l, bend left, "\text{decode}"] \end{tikzcd}$$

for example, strings can be encoded in ASCII

Def Information $\neq$ Data:

Information: content of some message

Tells us details about some system

Data: Raw unit of information

Representation (or encoding) of information

Def Entropy is the measure of disorder or non-uniformness of a system

Shannon Entropy: Expected value of the information contained in some data

Ex: A coin flip has two outcomes H or T, which is 1 bit of information

A baised coin flip that has only H, then 0 bits of information

Def when there are n outcomes then it contains $\lceil \log_2(n) \rceil$ bits of information

Def Uniformity: Lack of variation among symbols in a message

Ex:

Uniform	Variable
AAAAAAAA	ACGTACGT

generally, the more uniform the data, the less entropy and thus less information

Coding Trees, Prefix Codes

Saturday, November 6, 2021 5:48 PM

<u>Idea</u> We can encode a message using a tree. These are called encoding trees.

<u>Ex</u> Given the encoding map:

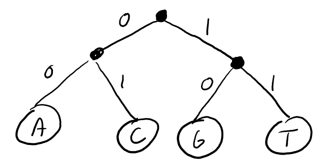
A → 00

C → 01

G → 10

T → 11

we can create the tree:



<u>Def</u> An encoding tree is a rooted tree where each leaf represents a character and the path to the node is its encoding.

<u>Def</u> A <u>Prefix Code</u> is an encoding which no symbol is represented by a code that is a prefix of another symbol's code.

A Prefix Code can always be represented by a coding tree

Data Compression

Saturday, November 6, 2021 6:01 PM

Idea We can try to compress some data

lossless: retains all data in tact

lossy: smaller compression but may lose some data

Ex Given the two encoding methods:

ASCII	Two-Bit
A → 65	A → 0
C → 67	C → 1
G → 71	G → 2
T → 84	T → 3

A compression method could convert ASCII to the Two-Bit method and save 6 bits per character. It is a lossless compression

Def The lowest bound on Data Compression:

Shannon Entropy = $\sum_{x} p_x \log_2\left(\frac{1}{p_x}\right)$ where x are the symbols in a message

the Shannon Entropy is the lowest bound on Data Compression

Huffman Trees

Saturday, November 6, 2021 6:06 PM

Def A Huffman Tree is the optimal lossless compression

Alg: Construction:

1) Compute frequencies of symbols
2) Start with forest of individual trees with single symbol
3) While there are more than 1 tree in the forest
   - → remove the 2 trees with lowest frequencies
   - → create new node with combined frequencies of both trees
   - → make new node parent of both trees, lower frequency on left
   - → insert new tree into forest.

Alg: Encoding: Given a message, encode each symbol using the Huffman Tree as a coding tree.

Alg: Decoding: Given an encoding, decode each symbol by traversing the tree according to the encoding, when a leaf is reached that is the next decoded symbol and restart at the root.

Byte Buffers, Bit Buffers (Read/Write)

Saturday, November 6, 2021 6:45 PM

<u>Idea</u> We can use a buffer to write bytes to I/O efficiently:

memory $\xrightarrow{\text{bytes}}$ [output stream] $\xrightarrow{\text{bytes}}$ [buffer] $\xrightarrow{\text{4KB}}$ [disk]

We can use a buffer to read bytes from I/O efficiently:

[disk] $\xrightarrow{\text{4KB}}$ [input stream] $\xrightarrow{\text{bytes}}$ [buffer] $\xrightarrow{\text{bytes}}$ memory

<u>Idea</u> Instead, if we want to write/read bits instead of bytes, we can use a bitwise buffer:

memory $\xrightarrow{\text{bits}}$ [bitwise buffer] $\xrightarrow{\text{byte}}$ [Output stream] $\xrightarrow{\text{bytes}}$ [buffer] $\xrightarrow{\text{4KB}}$ [disk]

[disk] $\xrightarrow{\text{4KB}}$ [buffer] $\xrightarrow{\text{bytes}}$ [input stream] $\xrightarrow{\text{byte}}$ [bitwise buffer] $\xrightarrow{\text{bits}}$ memory

<u>Def</u> Reading from a bitwise buffer: Given a byte:

0	1	2	3	4	5	6	7

we can extract a specific bit using bit masks and shifts

thus: to extract the bit at position c:

```
(buff >> (7-c)) & 1;
```

<u>Def</u> Writing to a bitwise buffer: Given some bits:

we can shift the bit and OR with the buffer.

Graphs, Directed/Undirected, Weighted/Unweighted

Saturday, November 13, 2021 8:54 PM

Def Graphs are a collection of nodes and edges

Def Nodes are a single entity

Def Edges are a relationship between nodes

Def Undirected graphs: an edge (u,v) is $u \to v$ and $v \to u$

Directed graph: an edge (u,v) is only $u \to v$

Def Unweighted graph: edges do not have a weight

Weighted graph: edges have a cost/weight value

Cycles

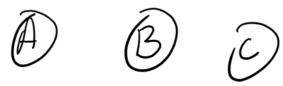
Saturday, November 13, 2021 8:59 PM

<u>Def</u> A <u>cycle</u> is a path starting from and ending at the same node

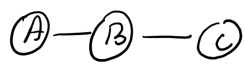
Unstructured, Sequential, Hierarchical, Structured

Saturday, November 13, 2021 9:01 PM

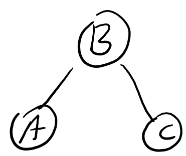
Def Unstructured: A disconnected collection of nodes:



Def Sequential: An ordered collection of connected nodes:



Def Hierarchical: A ranked collection of connected nodes:

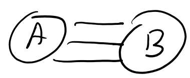


Def Structured: A collection of connected and unconnected nodes:

Multigraphs

Saturday, November 13, 2021 9:04 PM

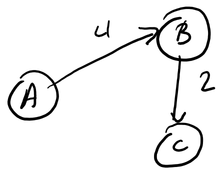
Def A multigraph is a graph in which a pair of nodes can have multiple distinct edges:



Graph Representations in Memory

Saturday, November 13, 2021 9:05 PM

Given some graph:



thus V = set of vertices

E = set of edges

<u>Def</u> Adjacency matrix:

	A	B	C
A	X	4	X
B	X	X	2
C	X	X	X

size: $O(|V|^2)$

<u>Def</u> Adjacency list:

A : { (B, 4) }

B : { (C, 2) }

C : { }

size: $O(|V| + |E|)$

Graph Traversals, BFS, DFS, Complexities

Saturday, November 13, 2021 9:10 PM

Time complexity:

BFS: Add (0, start) to the queue
While the queue is not empty:
    pop (d, curr) from the queue
    if curr has not been visited:
        mark curr as visited with distance d
        for all outgoing edges (curr, w):
            if w has not been visited, add (d+1, w) to queue

$O(|V| + |E|)$

DFS: Add start to the stack
While the stack is not empty:
    pop curr from the stack
    if curr has not been visited:
        mark curr as visited
        for all edges (curr, w):
            if w has not been visited, add w to stack

$O(|V| + |E|)$

Dijkstra's Algorithm

Saturday, November 13, 2021 9:38 PM

0) Set all node distances to ∞ and start to 0
1) Add (0, start) to priority queue
2) While the priority queue is not empty:

        pop (d, curr) from priority queue
        if curr is not visited:
            mark curr as visited
            for all edges (curr, w, e):
                if d + e < w's current distance:
                    w's distance = d + e
                    w's previous node is curr
                    add (d + e, w) to priority queue

Time complexity:

$O(|V| + |E|)$

Spanning Trees, Minimum Spanning Trees

Sunday, November 21, 2021 5:47 PM

Def a Spanning Tree is a subtree of any graph G:

1) contains all nodes of G
2) contains a subset of the edges of G
3) Has no cycles
4) Is connected

Def a Minimum Spanning Tree minimizes the edge cost of creating the spanning tree

Def Prim's Algorithm: $O(|E|\log|E|)$

Start at any node
for $|V|-1$ times:
find the smallest weight edge such that one node on edge is in MST and the other is not
add edge to the MST

Def Kruskal's Algorithm: $O(|E|\log|E|)$

Repeat $|E|$ times:
Find the smallest edge such that adding it would not cause a cycle
Add edge to the MST

Disjoint Set ADT

Sunday, November 21, 2021 6:18 PM

<u>Def</u> Disjoint set ADT:

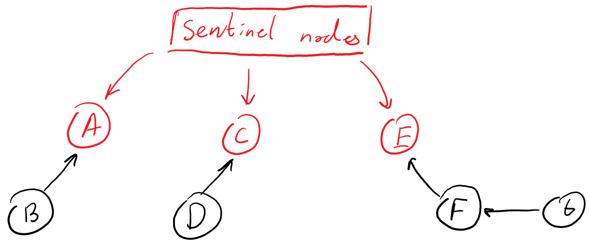
Union : Given two elements u & v, merge the sets in which they belong

Find : Given an element u, returns the set in which it belongs.

Up-Trees

Sunday, November 21, 2021 6:22 PM

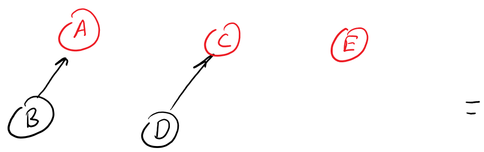
Def Up-Tree is a graph where each element is a single node. Each set is represented as a sentinel node. Sentinel nodes are nodes without a parent.



Find: Return the element's sentinel node

Union: Find elements' sentinel nodes and make one the child of the other.

Idea We can implement an Up-Tree using an array:



null	A	null	C	null
A	B	C	D	E

Def Union operation By Size: whichever sentinel node has more children should be the parent

Def Union operation By Height: whichever sentinel node has greater height should be the parent

Def Path compression: For every node under a sentinel, call find and point each node directly to sentinel.

Def Time complexity: Given an Up-Tree with n nodes

Union: $O(1)$ By Size, $O(n)$ By Height.

Find: Uncompressed: $O(n)$, Compressed: $O(1)$

Problem Complexity, P = NP

Sunday, November 28, 2021 9:57 PM

<u>Def</u> We can define a problem by describing its <u>Inputs</u> and <u>Outputs</u>

<u>Ex</u> Find maximum of list:

Input: A list of n numbers x_n

Output: A number $v \in x$ such that v is the maximum

<u>Def</u> We can group problems by complexity:

P: Problems can be solved in $O(n^c)$

NP: Problems can be verified in $O(n^c)$

- P is a subset of NP

NP-Hard: Problems at least as hard as the hardest problem in NP

NP-Complete: The intersection of NP and NP-Hard

- The hardest problems in NP

<u>Def</u> P = NP problem: