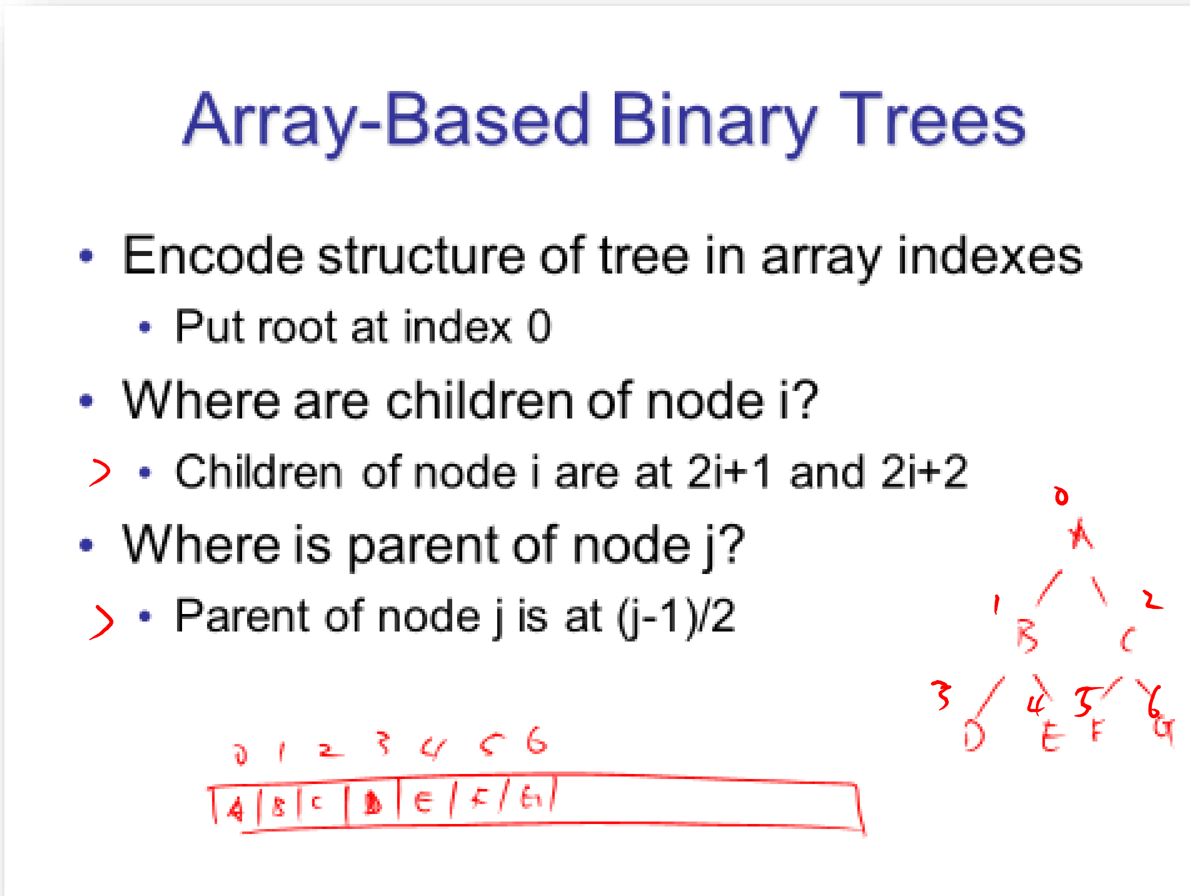
# [TAP:JZXUF] Array based Tree

- What is the index of E's parent?



# Administrative Details

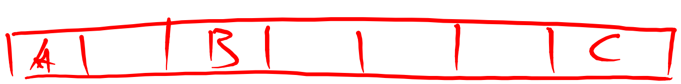
- CS Colloquium?!?!
  - Meets (almost) every Friday at 2:30pm
  - Guest speaker presents their research
  - Next Friday (4/20) we will have an information session instead of a normal speaker
    - Discussion of courses offered next semester
    - Advising about majoring in CS
      - We can sign major declaration sheets there
  - Food!

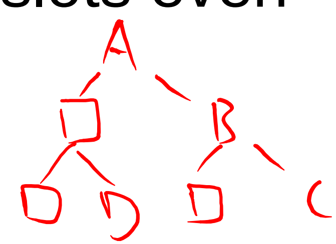
# Today's Outline

- Array-based Trees
- Huffman Encoding
- Priority Queue
  - Heap data structure

# ArrayTree Tradeoffs

- Why are ArrayTrees good?
  - Save space for links
  - No need for additional memory allocated/garbage collected
  - Works well for full or complete trees
    - No wasted space
    - Quick access to nodes (given the size of the tree)
- Why bad?
  - Could waste a lot of space for other trees
    - Tree of height of n requires $2^{n+1}$-1 array slots even if only O(n) elements



# Today's Outline

- Array-based Trees
- Huffman Encoding
- Priority Queue
  - Heap data structure

# Default Encoding of Characters

- Computers encode a text as a sequence of bits

## ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | | |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

char c

c ≥ 97

# Motivation

- In ASCII: 1 character = 8 bits (1 byte)
  - Allows for $2^8 = 256$ different characters

20

- Space to store "AN_ANTARCTIC_PENGUIN"

20 x 8 = 160 bits

- Is there a better way?
  - Note that only 11 symbols are used: ANTRCIPEGU_
  - "ASCII-lite" only needs 4 bits per symbol (since $2^3<11<2^4$)

20 x 4 = 80 bits

- Can we still do better??

# Huffman Codes

- Example
  - AN_ANTARCTIC_PENGUIN
  - Compute letter frequencies

| A | C | E | G | I | N | P | R | T | U | _ |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 1 | 1 | 2 | 4 | 1 | 1 | 2 | 1 | 2 |

- Key Idea: Use fewer bits for most common letters

| A | C | E | G | I | N | P | R | T | U | _ |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 1 | 1 | 2 | 4 | 1 | 1 | 2 | 1 | 2 |
| 110 | 111 | 1011 | 1000 | 000 | 001 | 1001 | 1010 | 0101 | 0100 | 011 |

# Features of Good Encoding

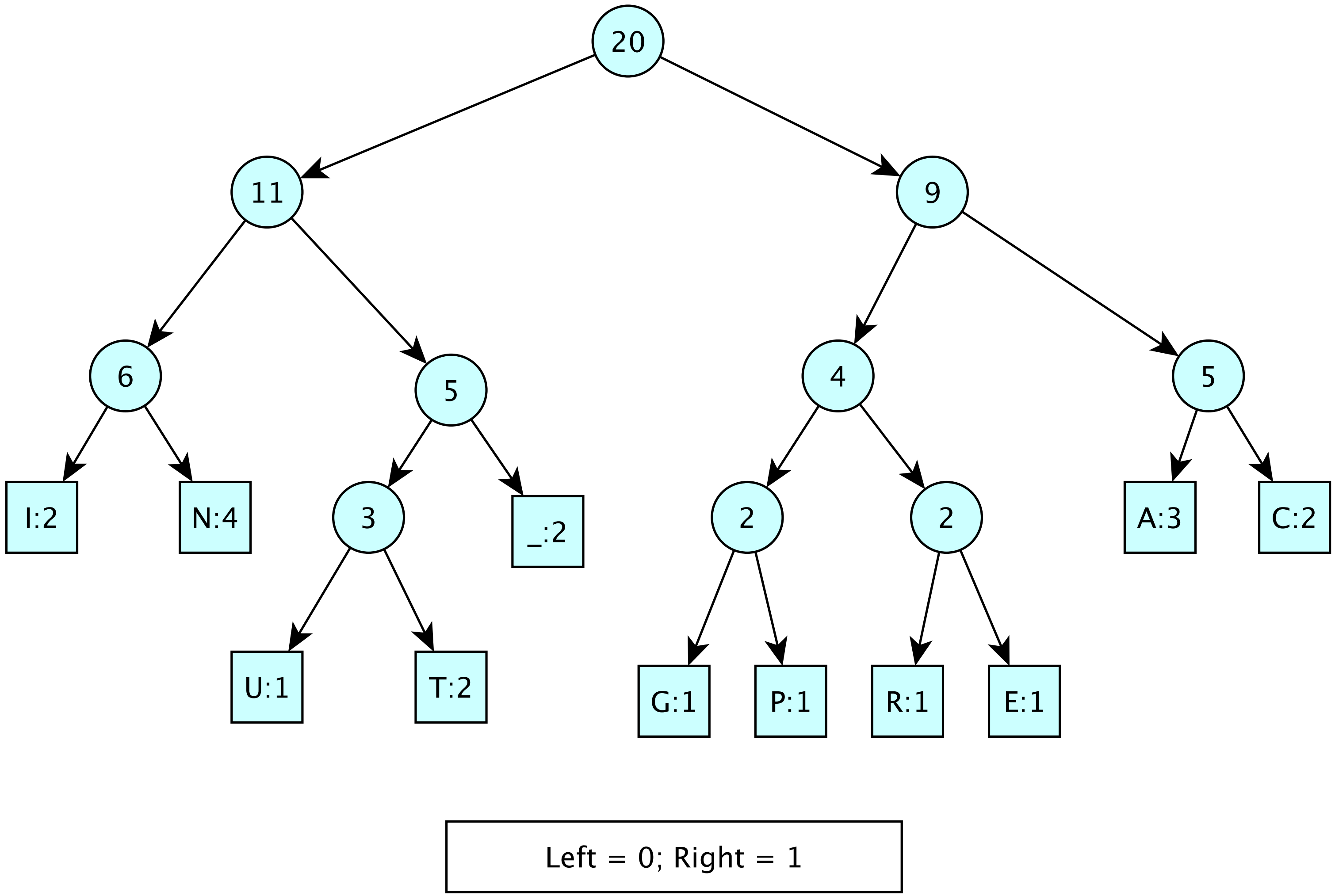
- Prefix property: No encoding is a prefix of another encoding (letters appear at leaves)
- No internal node has a single child
- Nodes with lower frequency have greater depth

- All optimal length unambiguous encodings have these features

# Huffman Encoding

- Input: symbols of alphabet with frequencies
- Huffman encode as follows
  - Create a single-node tree for each symbol: key is frequency; value is letter
  - while there is more than one tree
    - Find two trees T1 and T2 with lowest keys
    - Merge them into new tree T with dummy value and key= T1.key+ T2.key
- Theorem: The tree computed by Huffman is an optimal encoding for given

# The Huffman Tree




*Each node’s value is the sum of the frequencies of all its children

# How To Implement Huffman

- Keep a Vector of Binary Trees
- Sorted them by decreasing frequency
  - Removing two smallest frequency trees is fast
- Insert merged tree into correct sorted location in Vector
- Running Time:
  - O(n log n) for initial sorting
  - $O(n^2)$ for while loop
- Can we do better...?

# What Huffman Encoder Needs

- A structure S to hold items with *priorities*
- S should support operations
  - add(E item); // add an item
  - E removeMin(); // remove highest priority item
- S should be designed to make these two operations fast
- If, say, they both ran in O(log n) time, the Huffman while loop would take O(n log n) time instead of $O(n^2)$!

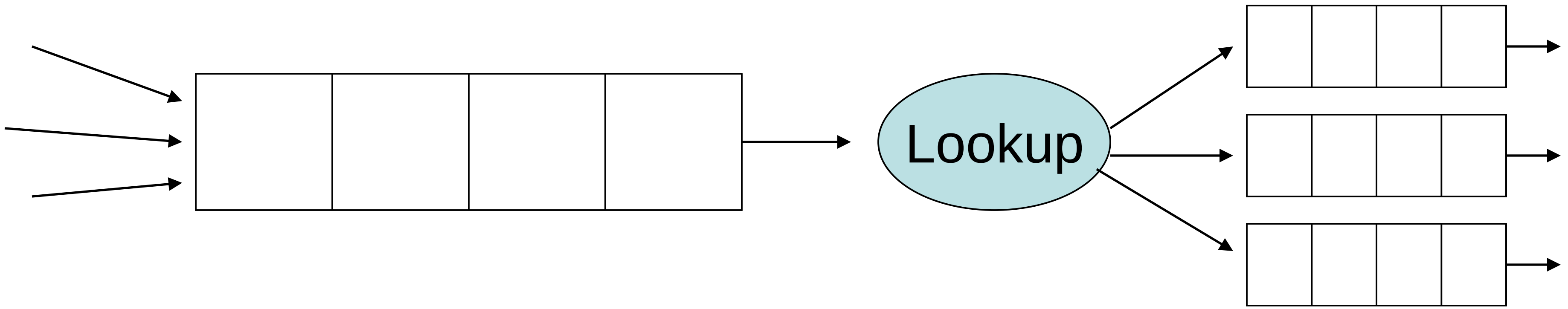
# Today's Outline

- Array-based Trees
- Huffman Encoding
- Priority Queue
  - Heap data structure

# Priority Queues

- Which data structure would you use to keep track of ~~customers~~ people in line?
  - What if it's a line in the Emergency Room?

# Priority Queue Application




## <u>Packet Sources May Be Ordered by Sender</u>

| | |
|---|---|
| `sysnet.cs.williams.edu` | `priority = 1 (best)` |
| `bull.cs.williams.edu` | `2` |
| `yahoo.com` | `10` |
| `spammer.com` | `100 (worst)` |

# Priority Queues

- Always dequeue object with **highest priority** (smallest rank) first regardless of when it was enqueued
- Data can be received/inserted in any order, but it is always returned/removed according to priority
- PQs require the values to be comparable

# PQ Interface

class person implements Comparable<person>

```
public interface PriorityQueue<E extends Comparable<E>> {
    public E getFirst(); // peeks at minimum element
    public E remove();     // removes minimum element
    public void add(E value); // adds an element
    public boolean isEmpty();
    public int size();
    public void clear();
}
```

dequeue

highest priority

enqueue

# Implementing PQs

- How would you implement PQs?
  - Build off of a Queue implementation?

# Today's Outline

- Array-based Trees
- Huffman Encoding
- Priority Queue
  - Heap data structure

ADT

# Heap

- A heap is a tree "sorted top to bottom":
  - any parent has a higher priority than it's children
    - Heap invariant: value <= values of children
  - Recursive definition:
    - Root holds the highest priority value
    - Subtrees are also heaps
- Not Unique: Several valid heaps can be constructed for the same data set

because no ordering exists between siblings

# Inserting into a PQ

- Steps
  - Add new value as a leaf
  - while (value < parent's value)
    - swap with parent

"bubble up"
"percolate up"

- Efficiency depends upon speed of
  - Finding a place to add new node
  - Finding parent
  - Tree height