From bits to bytes to ints

- At some level everything is stored as either a zero or a one
  - A bit is a binary digit a byte is a binary term (8 bits)
  - We should be grateful we can deal with Strings rather than sequences of 0's and 1's.
  - We should be grateful we can deal with an int rather than the 32 bits that make an int

- int values are stored as two's complement numbers with 32 bits, for 64 bits use the type long, a char is 16 bits
  - Standard in Java, different in C/C++
  - Facilitates addition/subtraction for int values
  - We don't need to worry about this, except to note:
    - Infinity + 1 = -Infinity
    - Math.abs(-Infinity) > Infinity
More details about bits

- How is 13 represented?
  - ... 0 0 1 1 0 1
  - Total is $8 + 4 + 1 = 13$

- What is bit representation of 32? Of 15? Of 1023?
  - What is bit-representation of $2^n - 1$?
  - What is bit-representation of 0? Of -1?
    - Study later, but -1 is all 1’s, left-most bit determines $< 0$

- How can we determine what bits are on? How many on?
  - Useful in solving problems, understanding machine
How are data stored?

- **To facilitate Huffman coding we need to read/write one bit**
  - Why do we need to read one bit?
  - Why do we need to write one bit?
  - When do we read 8 bits at a time? Read 32 bits at a time?

- **We can't actually write one bit-at-a-time. We can't really write one char at a time either.**
  - Output and input are buffered, minimize memory accesses and disk accesses
  - Why do we care about this when we talk about data structures and algorithms?
    - Where does data come from?
How do we buffer char output?

- Done for us as part of InputStream and Reader classes
  - InputStreams are for reading bytes
  - Readers are for reading char values
  - Why do we have both and how do they interact?

```java
Reader r = new InputStreamReader(System.in);
```

- Do we need to flush our buffers?

- In the past Java IO has been notoriously slow
  - Do we care about I? About O?
  - This is changing, and the java.nio classes help
    - Map a file to a region in memory in one operation
Buffer bit output

- **To buffer bit output we need to store bits in a buffer**
  - When the buffer is full, we write it.
  - The buffer might overflow, e.g., in process of writing 10 bits to 32-bit capacity buffer that has 29 bits in it
  - How do we access bits, add to buffer, etc.?

- **We need to use bit operations**
  - Mask bits -- access individual bits
  - Shift bits – to the left or to the right
  - Bitwise and/or/negate bits
Representing pixels

- A pixel typically stores RGB and alpha/transparency values
  - Each RGB is a value in the range 0 to 255
  - The alpha value is also in range 0 to 255

```java
Pixel red = new Pixel(255,0,0,0);
Pixel white = new Pixel(255,255,255,0);
```

- Typically store these values as int values, a picture is simply an array of int values

```java
void process(int pixel){
    int blue = pixel & 0xff;
    int green = (pixel >> 8) & 0xff;
    int red = (pixel>> 16) & 0xff;
}
```
Bit masks and shifts

```c
void process(int pixel){
    int blue  = pixel & 0xff;
    int green = (pixel >> 8) & 0xff;
    int red   = (pixel >> 16) & 0xff;
}
```

- **Hexadecimal number: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f**
  - Note that f is 15, in binary this is 1111, one less than 10000
  - The hex number 0xff is an 8 bit number, all ones

- **The bitwise & operator creates an 8 bit value, 0—255 (why)**
  - 1&1 == 1, otherwise we get 0, similar to *logical and*
  - Similarly we have |, bitwise or
Problem: finding subsets

- See CodeBloat APT, requires finding sums of all subsets
  - Given \{72, 33, 41, 57, 25\} what is sum closest (not over) 100?
  - How do we do this in general?

- We’ll look at three solutions
  - Recursively generate all sums: similar to backtracking
  - Generate all sets of sums by considering each value
  - Representing which elements in subset by bits

- For the last solution, how many subsets?
  - What do 10110, 10001, 00111, 00000, and 11111 represent?
  - How do we generate sums from these representations?
From subsets to graphs with bits

- We’ll consider SequenceSync APT
  - What is a “vertex” in the graph? Where are arcs?

- For state-0, we have \{1,5,4,2\} for transitions

- We’ll consider a graph in which vertices are sets of states
  - Start with every possible state in our initial vertex
How do we search graph?

- **Given a vertex (collection of states) how do we determine what vertex it’s connected to?**
  - Consider each transition from each state in our vertex (remember this is a set of states)
  - This yields a new set of states/vertex 1-away from vertex

- **What does the code look like for bfs? When do we stop?**

```java
while (q.size() != 0){
    TreeSet<Integer> current = q.remove();
    for(int k=0; k < 4; k++){
        TreeSet<Integer> next = new TreeSet<Integer>();
        for(int val : current){
            next.add(matrix[val][k]);
        }
        q.add(next); // if not already seen
    }
}
```
Problems with approach?

- Creating sets and looking them up in map takes time
  - This solution times out, how to improve it?

- Don’t represent set of states explicitly, use sequence of bits
  - Similar to CodeBloat, advantages? Disadvantages?
  - How do we determine when we’re done?
  - How do store distances (how is array like a map?)

- Rewrite solution to be efficient using int for set
  - Initial set is all ones, how to make this?
A Rose by any other name…C or Java?

● Why do we use Java in our courses (royal we?)
  ➢ Object oriented
  ➢ Large collection of libraries
  ➢ Safe for advanced programming and beginners
  ➢ Harder to shoot ourselves in the foot

● Why don't we use C++ (or C)?
  ➢ Standard libraries weak or non-existant (comparatively)
  ➢ Easy to make mistakes when beginning
  ➢ No GUIs, complicated compilation model
Why do we learn other languages?

- Perl, Python, PHP, mySQL, C, C++, Java, Scheme, ML, ...
  - Can we do something different in one language?
    - Depends on what different means.
    - In theory: no; in practice: yes
  - What languages do you know? All of them.
  - In what languages are you fluent? None of them

- In later courses why do we use C or C++?
  - Closer to the machine, we want to understand the machine at many levels, from the abstract to the ridiculous
    - Or at all levels of hardware and software
  - Some problems are better suited to one language
    - What about writing an operating system? Linux?
C++ on two slides

- Classes are similar to Java, compilation model is different
  - Classes have public and private sections/areas
  - Typically declaration in .h file and implementation in .cpp
    - Separate interface from actual implementation
    - Good in theory, hard to get right in practice
  - One .cpp file compiles to one .o file
    - To create an executable, we link .o files with libraries
    - Hopefully someone else takes care of the details

- We #include rather than import, this is a preprocessing step
  - Literally sucks in an entire header file, can take a while for standard libraries like iostream, string, etc.
  - No abbreviation similar to java.util.*;
C++ on a second slide

- We don't have to call new to create objects, they can be created "on the stack"
  - Using new creates memory "on the heap"
  - In C++ we need to do our own garbage collection, or avoid and run out of memory (is this an issue?)

- Vectors are similar to ArrayLists, pointers are similar to arrays
  - Unfortunately, C/C++ equate array with memory allocation
  - To access via a pointer, we don't use . we use ->

- Streams are used for IO, iterators are used to access begin/end of collection
  - ifstream, cout correspond to Readers and System.out
How do we read a file in C++ and Java?

```java
Scanner s = new Scanner(new File("data.txt"));
TreeSet<String> set = new TreeSet<String>();
while (s.hasNext()){
    String str = s.next;
    set.add(str);
}
myWordsAsList = new ArrayList<String>(set);

string word;
set<string> unique;
ifstream input("data.txt");
while (input >> word){
    unique.insert(word);
}
myWordsAsVector = vector<string>(unique.begin(), unique.end());
```

- What are similarities? Differences?
How do we read a file in C?

```c
FILE * file = fopen("/u/ola/data/poe.txt","r");
char buf[1024];
char ** words = (char **) malloc(5000*sizeof(char **));
int count = 0;
int k;
while (fscanf(file,"%s",buf) != EOF){
    int found = 0;    // look for word just read
    for(k=0; k < count; k++){
        if (strcmp(buf,words[k]) == 0){
            found = 1;
            break;
        }
    }
    if (!found){    // not found, add to list
        words[count] = (char *) malloc(strlen(buf)+1);
        strcpy(words[count],buf);
        count++;
    }
}
```

- What if more than 5000 words? What if string length > 1024? What if?
  - What is complexity of this code?