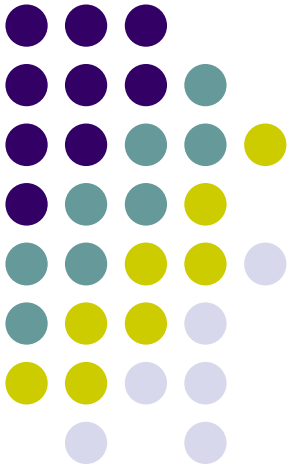
Mastering Data Structures and Algorithms: A Practical Approach

Bit manipulation, Arrays and Strings

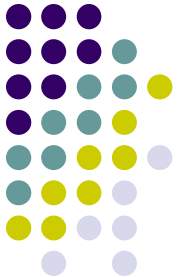


By Dr. Juan C. Gomez

Fall 2018

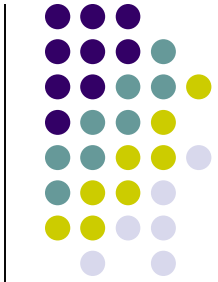
UC Santa Cruz Extension in Silicon Valley

Overview



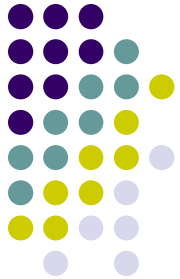
- The Master Recipe
- Bit Manipulation:
  - Bitwise Operators
  - Two’s complement representation
  - Arithmetic vs Logical Shift
  - Get/Set a Bit
  - Clear/Update a Bit
  - Multiplying and Dividing by powers of two
  - Examples

Overview



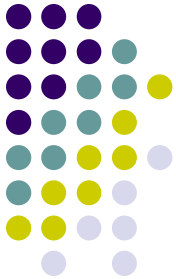
- Array and Strings
  - C++: vector
  - Java: ArrayList, StringBuilder
  - Strings: Java/C++
  - Hash Tables
    - C++ Map
    - Java HashMap
    - Java HashSet
  - Examples

The Master Recipe



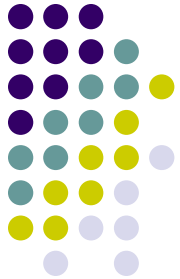
- Fully understand the problem
  - Ask about possible inputs
- Select the appropriate data structures
  - Do the most frequent operations have acceptable performance?
- Select or discover proper algorithm
  - Is the problem recursive? Iterative? Does it fit divide and conquer? Is your first pick exponential time? Does dynamic programming apply?

The Master Recipe



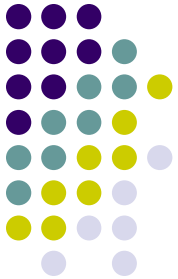
- Run select inputs through your algorithm: does it work?
  - Empty input.
  - Single entry input.
  - Full tree vs left or right biased tree.
  - Does it deal properly with tree nodes that:
    - Have all children
    - No children
    - Only one child

The Master Recipe



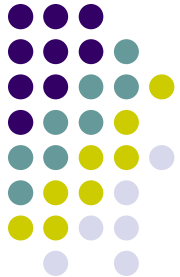
- Ask for feedback from interviewer:
  - Is performance acceptable
  - Is space complexity acceptable
- Start coding:
  - Use the language you know best for the given problem
  - Pay attention to border conditions.
  - Pay attention to error conditions.
  - Factor out trivial functionality
  - Eye-ball your code for inefficiencies and errors

The Master Recipe



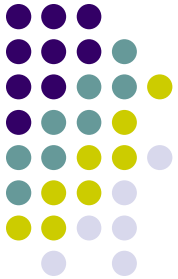
- Testing:
  - Do this before you hand back the solution to the interviewer!
  - Border case inputs.
  - Basic inputs representative of most possible inputs.
  - Use tables to track your variables.
  - Test loop border conditions.
  - Beware of Null pointers.
  - Handle all cases.

The Master Recipe



- Always remember at all times during the interview:
  - Take feedback at any time: If the interviewer wants to speak, listen to what she/he has to say!
  - Any solution is better than no solution, so do not get stuck trying to optimize your solution: often enough, the interviewer will follow up with questions to refine your solution.
  - Always expect follow up questions:
    - How can you improve your time for a given operation?
    - How can you reduce your memory consumption?

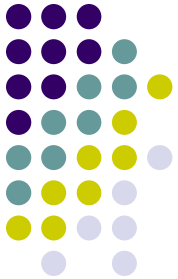
Bitwise Operators



0110 + 0010	0011 * 0101	0110 + 0110
0011 + 0010	0011 * 0011	0100 * 0011
0110 - 0011	1101 >> 2	1101 ^ (~1101)
1000 - 0110	1101 ^ 0101	1011 & (~0 << 2)

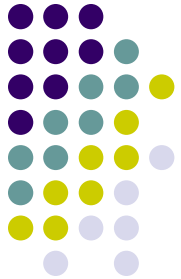
Solutions: line 1 (1000, 1111, 1100); line 2 (0101, 1001, 1100); line 3 (0011, 0011, 1111); line 4 (0010, 1000, 1000).

Bitwise Operators



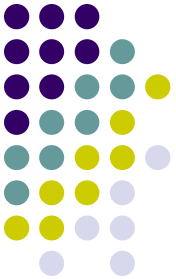
x ^ 0s = x	x & 0s = 0	x | 0s = x
x ^ 1s = ~x	x & 1s = x	x | 1s = 1s
x ^ x = 0	x & x = x	x | x = x

Two's Complement Representation



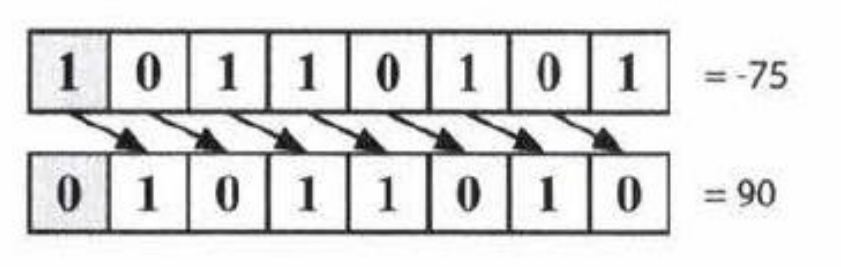
- Negative numbers are represented as follows:
  - Invert all bits
  - Add one
  - Left-most bit is the sign (one if negative, zero otherwise)

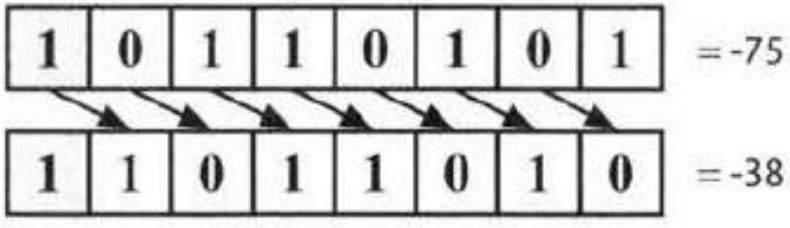
Two's Complement Representation



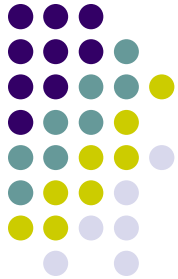
Positive Values		Negative Values	
7	0 111	-1	1 111
6	0 110	-2	1 110
5	0 101	-3	1 101
4	0 100	-4	1 100
3	0 011	-5	1 011
2	0 010	-6	1 010
1	0 001	-7	1 001
0	0 000		

Arithmetic vs Logical Shift



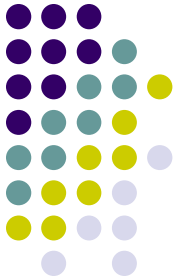


Arithmetic vs Logical Shift



```
1   int repeatedArithmeticShift(int x, int count) {
2     for (int i = 0; i < count; i++) {
3       x >>= 1; // Arithmetic shift by 1
4     }
5     return x;
6   }
7
8   int repeatedLogicalShift(int x, int count) {
9     for (int i = 0; i < count; i++) {
10      x >>>= 1; // Logical shift by 1
11    }
12    return x;
13  }
```

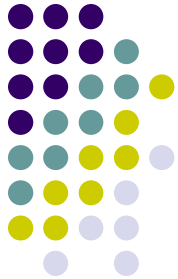
Get and Set a Bit



```
1   boolean getBit(int num, int i) {
2     return ((num & (1 << i)) != 0);
3   }
```

```
1   int setBit(int num, int i) {
2     return num | (1 << i);
3   }
```

Clear Bit(s)

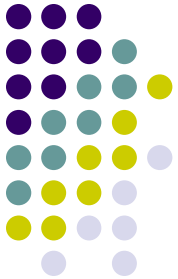


```
1  int clearBit(int num, int i) {
2    int mask = ~(1 << i);
3    return num & mask;
4  }
```

```
1  int clearBitsMSBthroughI(int num, int i) {
2    int mask = (1 << i) - 1;
3    return num & mask;
4  }
```

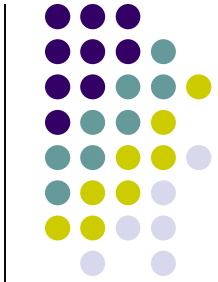
```
1  int clearBitsIthrough0(int num, int i) {
2    int mask = (-1 << (i + 1));
3    return num & mask;
4  }
```

Update Bit



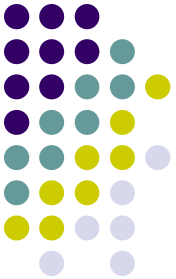
```
1  int updateBit(int num, int i, boolean bitIs1) {
2    int value = bitIs1 ? 1 : 0;
3    int mask = ~(1 << i);
4    return (num & mask) | (value << i);
5  }
```

Multiplying and Dividing by Powers of Two



- Optimize multiply or divide by powers of two:
  - $X / 2^n = X >> n$
  - $X * 2^n = X << n$
  - Caveats?

Examples



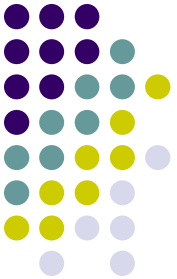
5.1 **Insertion:** You are given two 32-bit numbers, N and M, and two bit positions, i and j. Write a method to insert M into N such that M starts at bit j and ends at bit i. You can assume that the bits j through i have enough space to fit all of M. That is, if M = 10011, you can assume that there are at least 5 bits between j and i. You would not, for example, have j = 3 and i = 2, because M could not fully fit between bit 3 and bit 2.

EXAMPLE

Input: N = 10000000000, M = 10011, i = 2, j = 6

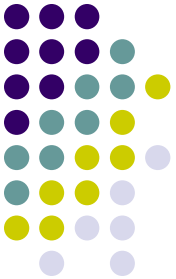
Output: N = 10001001100

Examples



5.2 **Binary to String:** Given a real number between 0 and 1 (e.g., 0.72) that is passed in as a double, print the binary representation. If the number cannot be represented accurately in binary with at most 32 characters, print "ERROR."

Examples



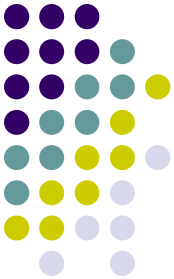
5.3 **Flip Bit to Win:** You have an integer and you can flip exactly one bit from a 0 to a 1. Write code to find the length of the longest sequence of 1s you could create.

EXAMPLE

Input: 1775 (or: 11011101111)

Output: 8

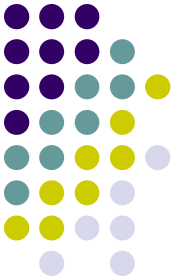
Examples



5.4 **Next Number:** Given a positive integer, print the next smallest and the next largest number that have the same number of 1 bits in their binary representation.

5.5 **Debugger:** Explain what the following code does: `((n & (n-1)) == 0)`.

Examples



5.6 **Conversion:** Write a function to determine the number of bits you would need to flip to convert integer A to integer B.

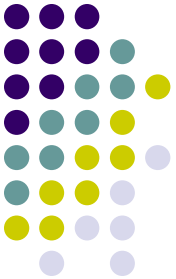
EXAMPLE

Input: `29 (or: 11101), 15 (or: 01111)`

Output: 2

5.7 **Pairwise Swap:** Write a program to swap odd and even bits in an integer with as few instructions as possible (e.g., bit 0 and bit 1 are swapped, bit 2 and bit 3 are swapped, and so on).

Examples



5.8 **Draw Line:** A monochrome screen is stored as a single array of bytes, allowing eight consecutive pixels to be stored in one byte. The screen has width w, where w is divisible by 8 (that is, no byte will be split across rows). The height of the screen, of course, can be derived from the length of the array and the width. Implement a function that draws a horizontal line from `(x1, y)` to `(x2, y)`.

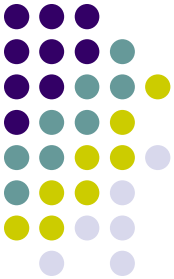
The method signature should look something like:

```
drawLine(byte[] screen, int width, int x1, int x2, int y)
```

Arrays and Strings: C++

- Static vs Resizable arrays
  - C++:
    - int intArray[10] = { 0 };
    - std::vector<int> intVector;
      - intVector.push_back(0); // Grows as you append
      - Other handy methods:
        - V.size()
        - V.clear()
        - V.empty()

Array and Strings: C++

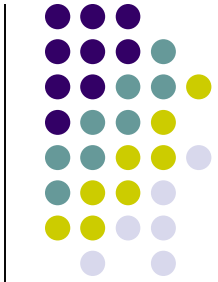


- Enumerating elements in a vector:

```
for( vector<int>::iterator itr = v.begin( ); itr != v.end( ); ++itr )
   cout << *itr << endl;
```

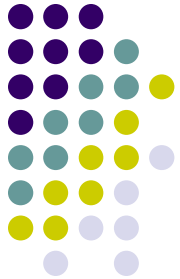
```
vector<int>::iterator itr = v.begin( );
while( itr !=v.end( ) )
   cout << *itr++ << endl;
```

Array and Strings: C++



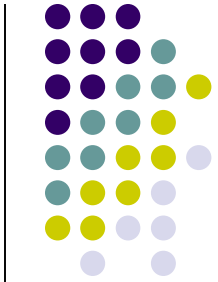
- Other things to keep in mind:
  - Const_iterator
  - Reverse_iterator (rbegin(), rend())

Array and Strings: Java



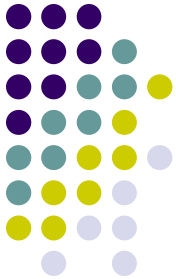
```
1  ArrayList<String> merge(String[] words, String[] more) {
2    ArrayList<String> sentence = new ArrayList<String>();
3    for (String w : words) sentence.add(w);
4    for (String w : more) sentence.add(w);
5    return sentence;
6  }
```

Array and Strings: Java



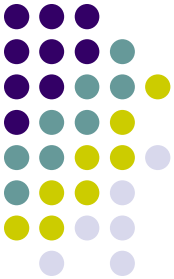
- Implementation of Array List:
  - Beware of non-O(1) complexity for:
    - AL.add(idx, Anytype x) (O(n)!)
    - AL.remove(idx) (O(n)!)

Array and Strings: Java



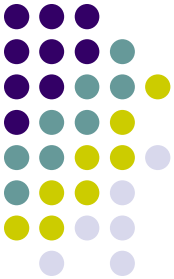
```
1   public class MyArrayList<AnyType> implements Iterable<AnyType>
2   {
3       private static final int DEFAULT_CAPACITY = 10;
4
5       private int theSize;
6       private AnyType [ ] theItems;
7
8       public MyArrayList( )
9         { doClear( ); }
10
11      public void clear( )
12        { doClear( ); }
13
14      private void doClear( )
15        { theSize = 0; ensureCapacity( DEFAULT_CAPACITY ); }
16
17      public int size( )
18        { return theSize; }
19      public boolean isEmpty( )
20        { return size( ) == 0; }
21      public void trimToSize( )
22        { ensureCapacity( size( ) ); }
```

Array and Strings: Java



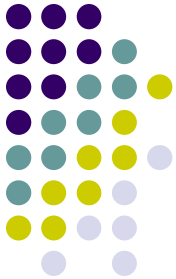
```
24        public AnyType get( int idx )
25        {
26            if( idx < 0 || idx >= size( ) )
27                throw new ArrayIndexOutOfBoundsException( );
28            return theItems[ idx ];
29        }
30
31        public AnyType set( int idx, AnyType newVal )
32        {
33            if( idx < 0 || idx >= size( ) )
34                throw new ArrayIndexOutOfBoundsException( );
35            AnyType old = theItems[ idx ];
36            theItems[ idx ] = newVal;
37            return old;
38        }
```

Array and Strings: Java



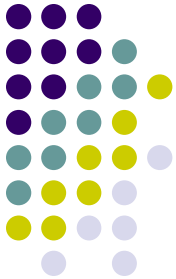
```
40      public void ensureCapacity( int newCapacity )
41      {
42          if( newCapacity < theSize )
43              return;
44
45          AnyType [ ] old = theItems;
46          theItems = (AnyType []) new Object[ newCapacity ];
47          for( int i = 0; i < size( ); i++ )
48              theItems[ i ] = old[ i ];
49      }
```

Array and Strings: Java



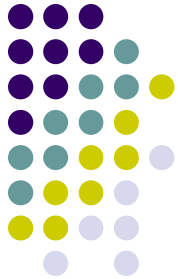
```
50      public boolean add( AnyType x )
51      {
52          add( size( ), x );
53          return true;
54      }
55
56      public void add( int idx, AnyType x )
57      {
58          if( theItems.length == size( ) )
59              ensureCapacity( size( )*  2 + 1 );
60          for( int i = theSize; i > idx; i-- )
61              theItems[ i ] = theItems[ i - 1 ];
62          theItems[ idx ] = x;
63
64          theSize++;
65      }
```

Array and Strings: Java



```
67      public AnyType remove( int idx )
68      {
69          AnyType removedItem = theItems[ idx ];
70          for( int i = idx; i < size( ) - 1; i++ )
71              theItems[ i ] = theItems[ i + 1 ];
72
73          theSize--;
74          return removedItem;
75      }
```

Array and Strings: Java

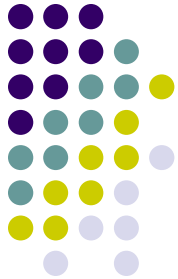


- StringBuilder

```
1  String joinWords(String[] words) {
2    String sentence = "";
3    for (String w : words) {
4      sentence = sentence + w;
5    }
6    return sentence;
7  }
```

On each concatenation, a new copy of the string is created, and the two strings are copied over, character by character. The first iteration requires us to copy x characters. The second iteration requires copying 2x characters. The third iteration requires 3x, and so on. The total time therefore is $O(x + 2x + \ldots + nx)$. This reduces to $O(xn^2)$.

Array and Strings: Java



- StringBuilder

StringBuilder can help you avoid this problem. StringBuilder simply creates a resizable array of all the strings, copying them back to a string only when necessary.

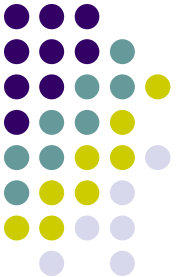
```
1  String joinWords(String[] words) {
2     StringBuilder sentence = new StringBuilder();
3     for (String w : words) {
4        sentence.append(w);
5     }
6     return sentence.toString();
7  }
```

Strings: Java

- Concatenation:

```
public class StringDemo {

   public static void main(String args[]) {
      String string1 = "saw I was ";
      System.out.println("Dot " + string1 + "Tod");
   }
}
```

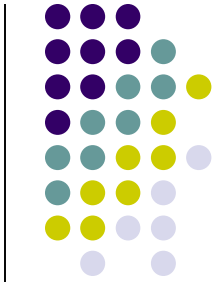
Strings: Java



- Formatting:

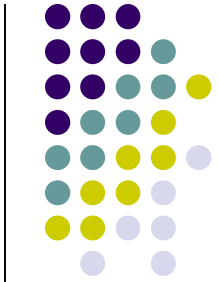
```
String fs;
fs = String.format("The value of the float variable is " +
                   "%f, while the value of the integer " +
                   "variable is %d, and the string " +
                   "is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

Strings: Java



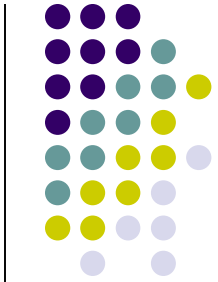
- Popular methods:
  - S.equals(S)
  - S.indexOf/lastIndexOf(C|S)
  - S.isEmpty()
  - S.length()
  - S.replace(C1,C2)
  - S.substring(Sidx,Eidx)
  - S.toLower/toUpper()

Strings: C++



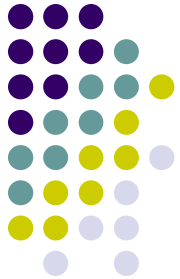
- Popular methods:
  - S.c_str()
  - S.append()/+
  - S.find(S)/rfind(S)
  - S.find_first_of(C)/find_last_of(C)
  - S.substr(idx1,idx2)
  - S.replace(idx,length, S)

Strings: Java and C++



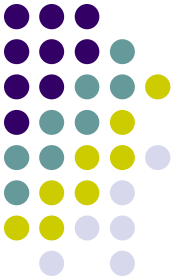
- String Concatenation Semantic:
  - S3 = S1 + S2
  - Time Complexity?
  - Space Complexity?
- Append to end of string:
  - Sn = Sn-1 + Sc
  - Time Complexity?
  - Space Complexity?

Strings: C



- char string[n];
- Null terminated: string[n] = '\0'
  - strcat(str1, str2);
  - strchr(str, c);
  - strcmp(str1, str2);
  - strcpy(dst, src);
  - strlen(str);
  - Use strn*() versions!

Strings: C memcpy vs memmove

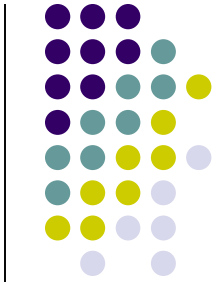


```
void * Memcpy(void* dst, const void* src, unsigned int cnt)
{
 char *pszDest = (char *)dst;

 const char *pszSource =( const char*)src;

 while(cnt) //till cnt
  {
   //Copy byte by byte
     *(pszDest++)= *(pszSource++);
     --cnt;
  }

 return dst;
}
```

Strings: C memcpy vs memmove

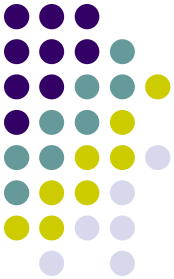
```
void * Memmove(void* dst, const void* src, unsigned int cnt)
{

 char *tmp = NULL;

 unsigned int uiLoop = 0;

 char *pszDest = (char *)dst;

 const char *pszSource =( const char*)src;

 //allocate memory for tmp array
 tmp = (char *)malloc(sizeof(char ) * cnt);
 if(NULL == tmp)
 {
 return NULL;
 }
 else
 {
 // copy src to tmp array
 for(uiLoop =0;uiLoop < cnt ; ++uiLoop)
 {
 *(tmp + uiLoop) = *(pszSource + uiLoop);
 }

 //copy tmp to dst
 for(uiLoop =0 ;uiLoop < cnt ; ++uiLoop)
 {
 *(pszDest + uiLoop) = *(tmp + uiLoop);
 }

 free(tmp); //free allocated memory
 }

 return dst;
}
```

Strings: C memcpy vs memmove



- Can you implement memove in O(1) memory?
- Board exercise!

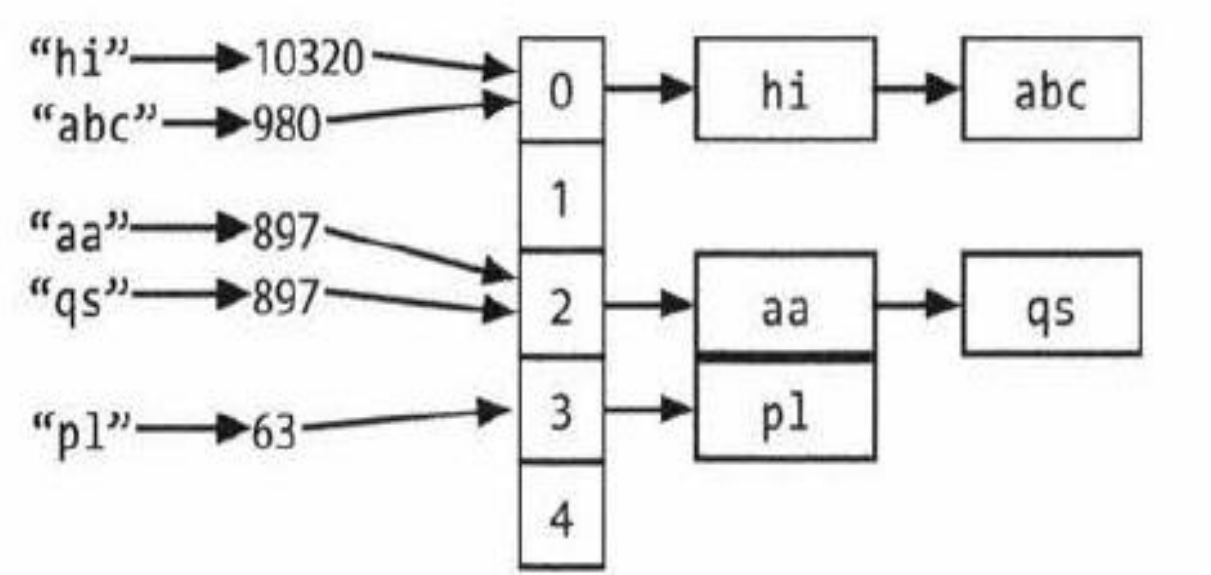
Hash Tables



- Maps Keys to Values in O(1) time

1. First, compute the key's hash code, which will usually be an `int` or `long`. Note that two different keys could have the same hash code, as there may be an infinite number of keys and a finite number of ints.
2. Then, map the hash code to an index in the array. This could be done with something like `hash(key) % array_length`. Two different hash codes could, of course, map to the same index.
3. At this index, there is a linked list of keys and values. Store the key and value in this index. We must use a linked list because of collisions: you could have two different keys with the same hash code, or two different hash codes that map to the same index.

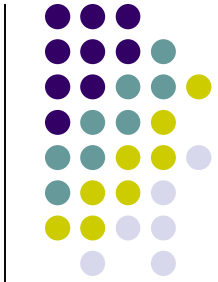
Hash Tables



Hash Tables

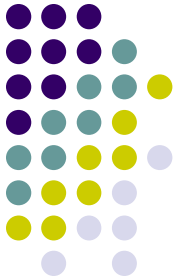
- Other ways to solve collisions:
  - Separate Chaining
  - Linear Probing $f(i) = i$.
  - Quadratic Probing $f(i) = i^2$.
  - Double Hashing $f(i) = i \cdot hash_2(x)$.

Hash Tables



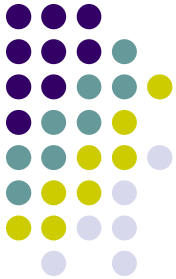
- Interface:
  - insert(K,V) O(1)
  - get(K) O(1)
  - delete(K) O(1)
  - enumerate()?
  - get(V)?

Hash Tables: C++ map



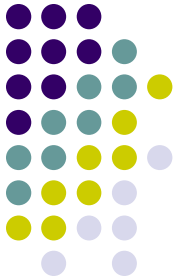
Alternatively, we can implement the hash table with a balanced binary search tree. This gives us an O(log N) lookup time. The advantage of this is potentially using less space, since we no longer allocate a large array. We can also iterate through the keys in order, which can be useful sometimes.

Hash Tables: C++ map



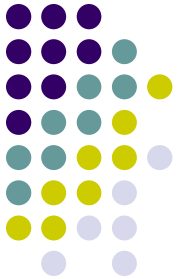
```cpp
// constructing maps
#include <iostream>
#include <map>

bool fncomp (char lhs, char rhs) {return lhs<rhs;}

struct classcomp {
  bool operator() (const char& lhs, const char& rhs) const
  {return lhs<rhs;}
};

int main ()
{
  std::map<char,int> first;

  first['a']=10;
  first['b']=30;
  first['c']=50;
  first['d']=70;

  std::map<char,int> second (first.begin(),first.end());

  std::map<char,int> third (second);

  std::map<char,int,classcomp> fourth;                 // class as Compare

  bool(*fn_pt)(char,char) = fncomp;
  std::map<char,int,bool(*)(char,char)> fifth (fn_pt); // function pointer as Compare

  return 0;
}
```

Hash Tables: C++ map



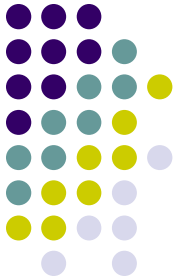
```
using namespace std;

int main()
{
    map <int, int> gquiz1;        // empty map container

    // insert elements in random order
    gquiz1.insert(pair <int, int> (1, 40));
    gquiz1.insert(pair <int, int> (2, 30));
    gquiz1.insert(pair <int, int> (3, 60));
    gquiz1.insert(pair <int, int> (4, 20));
    gquiz1.insert(pair <int, int> (5, 50));
    gquiz1.insert(pair <int, int> (6, 50));
    gquiz1.insert(pair <int, int> (7, 10));

    // printing map gquiz1
    map <int, int> :: iterator itr;
    cout << "\nThe map gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)
    {
        cout  <<  '\t' << itr->first
              <<  '\t' << itr->second << '\n';
    }
    cout << endl;
```

Hash Tables: C++ map



- Map<K,V>::iterator it = map.find(K);
- If it != map.end()
  - It->first() => K
  - It->second() => V

Hash Tables: Java



- HashSet<K>
  - Methods
    - add(K)
    - contains(K)
    - remove(K)

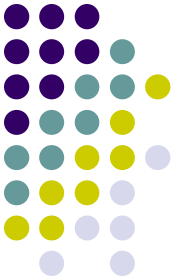
```
public class HashSetDemo {
   public static void main(String args[]) {

      // create hash set
      HashSet <String> newset = new HashSet <String>();

      // populate hash set
      newset.add("Learning");
      newset.add("Easy");
      newset.add("Simply");

      // create an iterator
      Iterator iterator = newset.iterator();

      // check values
      while (iterator.hasNext()) {
         System.out.println("Value: "+iterator.next() + " ");
      }
   }
}
```

Hash Tables: Java

- HashMap<K, V>
  - Methods
    - put(K, V)
    - containsKey(K)
    - get(K) => V
    - entrySet()
    - keySet()
    - values()
    - remove(K)
    - replace(K,V)

```
public static void printMap(Map mp) {
    Iterator it = mp.entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry pair = (Map.Entry)it.next();
        System.out.println(pair.getKey() + " = " + pair.getValue());
        it.remove(); // avoids a ConcurrentModificationException
    }
}
```

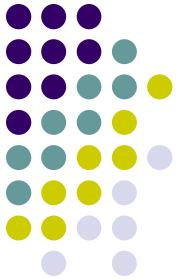
Examples



1.1 **Is Unique:** Implement an algorithm to determine if a string has all unique characters. What if you cannot use additional data structures?

1.2 **Check Permutation:** Given two strings, write a method to decide if one is a permutation of the other.

Examples



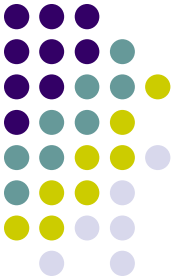
1.3 **URLify:** Write a method to replace all spaces in a string with '%20'. You may assume that the string has sufficient space at the end to hold the additional characters, and that you are given the "true" length of the string. (Note: If implementing in Java, please use a character array so that you can perform this operation in place.)

EXAMPLE

Input: "Mr John Smith ", 13

Output: "Mr%20John%20Smith"

Examples



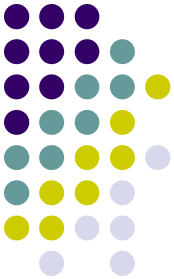
1.4 Palindrome Permutation: Given a string, write a function to check if it is a permutation of a palindrome. A palindrome is a word or phrase that is the same forwards and backwards. A permutation is a rearrangement of letters. The palindrome does not need to be limited to just dictionary words.

EXAMPLE

Input: `Tact Coa`

Output: `True (permutations: "taco cat", "atco cta", etc.)`

Examples

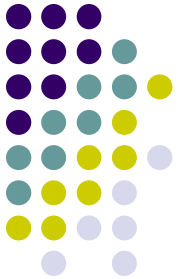


1.5 One Away: There are three types of edits that can be performed on strings: insert a character, remove a character, or replace a character. Given two strings, write a function to check if they are one edit (or zero edits) away.

EXAMPLE

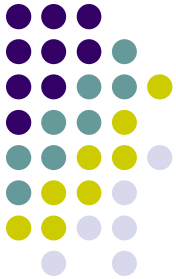
```
pale,  ple  -> true
pales, pale -> true
pale,  bale -> true
pale,  bake -> false
```

Examples



1.6 **String Compression:** Implement a method to perform basic string compression using the counts of repeated characters. For example, the string `aabcccccaaa` would become `a2b1c5a3`. If the "compressed" string would not become smaller than the original string, your method should return the original string. You can assume the string has only uppercase and lowercase letters (a - z).

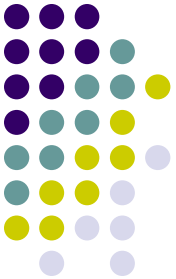
Examples



1.7 **Rotate Matrix:** Given an image represented by an NxN matrix, where each pixel in the image is 4 bytes, write a method to rotate the image by 90 degrees. Can you do this in place?

1.8 **Zero Matrix:** Write an algorithm such that if an element in an MxN matrix is 0, its entire row and column are set to 0.

Examples



1.9 **String Rotation:** Assume you have a method `isSubstring` which checks if one word is a substring of another. Given two strings, `s1` and `s2`, write code to check if `s2` is a rotation of `s1` using only one call to `isSubstring` (e.g., "`waterbottle`" is a rotation of "`erbottlewat`").

Questions?