Useful Algorithms and Programming Techniques

Billy Moses

What is this and why should I care?

This lecture is designed to help give basic coding skills and mindsets to enable you the programmer to solve problems quickly and more easily. This lecture will include the following subjects:

- Good Programming Practice and enhancing run-time speed
- Euler's GCD algorithm
- Stacks vs Queues
- Floodfill
- Graph Theory and Djikstra's minimum distance algorithm
- Important data structures
- Dynamic Programming
- Language Differences

For more information on any of these topics please consult the SCT website: http://www.tjhsst.edu/studentlife/activ/sct/lectures.php

Good Programming Practice and enhancing run-time speed

Clean code

In your CS class you may have learned about the importance of having "clean code" by making well-named variables, well-structured code, comments describing what your code is doing, and catching all the nasty Exceptions. This, while nice from time to time, is completely unnecessary when it comes to solving USACO-type problems. When you're on a time limit, you should worry more about how you code works rather than how to make it look nice. That said, you should still make sure that you the programmer can read and understand your code if, knock on wood, your code somehow doesn't work the first time. Finally instead of using try-catch statements just say that your main method throws an exception (e.g. public static main(String[] args) throws Exception)

Pre-coding

Before you sit down and start madly typing on your computer, sit and read the problem through once or twice and solve or diagram the problem out on paper first. This, while seemingly unnecessary, is an extremely important step in coding and will allow you to find easier, less time-taking solutions to the problem before you become a coding zombie and spontaneously forget what an integer is.

Know your parameters

It is very important when you get a problem to realize the bounds of what you could be asked. While this may seem simple at first (like knowing whether to use a double or an integer), make sure that the input of the problem will actually fall in the range of your variable. If your input is above 2,147,483,647, you may want to use a long or even BigInteger instead of an integer. Finally, if you know that your input parameters are small (eg 1<=x<=10) you may want to use a smaller data type such as a short to save space. Below is a list of the limits of all the java and signed C++ primitive types. The range of unsigned primitive types in C++ is 0 to double the signed max.

Type	Min (inclusive)	Max (inclusive)
Byte	-128 (java) / -127 (c++)	127 (both)
Short	-32,768 (java) / -32,768(c++)	32,767 (both)
Integer	-2,147,483,648 (java) / -32,768(c++)	2,147,483,64 (java) / 32,768(c++)
Long (java)	-9,223,372,036,854,775,808	9,223,372,036,854,775,808
Long (c++)	-2147483647	2147483647
Boolean	false (0)	true (1)
Float	-4,294,967,295	4,294,967,296
Double	-18,446,744,073,709,551,615	18,446,744,073,709,551,616
Char	'\u0000' (0)	'\uffffff' (65,535)

The following are not primitives but are still important!

- String
- BigInteger
- BigDecimal

Finally try to avoid doubles and floats at all costs since they often spontaneously produce error. Because of this, to correctly compare if two numbers are equal you need to account for a given amount of error like below.

```
if(abs(a-b)<.0000001)
```

Scanner vs BufferedReader (Java coders only)

Whether you like it or not, java's standard i/o operations are extremely time-taking and could play a major factor in getting under the 1 or 2 second limit. Below are much faster alternatives to using the standard java i/o.

1. Don't use a Scanner, use a BufferedReader

```
BufferedReader f = new BufferedReader(new FileReader(”program.in"));
String s = f.readLine();
```

2. Don't use a FileOutputStream, use a PrintWriter

```
PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("program.out")));
out.println(“42");
```

3. Don't use .split(), use a StringTokenizer

```
StringTokenizer st = new StringTokenizer(f.readLine());
int row = Integer.parseInt(st.nextToken());
```

Finally, always end you code by closing the output and ending the program.

```
    out.close();
    System.exit(0);
```

Euler’s GCD Algorithm

How do you find the greatest common factor of two numbers? Lets take an example: 12 and 18.
The naive method is to simply iterate over integers less than both of them and take the greatest one that divides them both evenly.

Again let’s try our example

	12	18	GCD
1	Yes	Yes	1
2	Yes	Yes	2
3	Yes	Yes	3
4	Yes	No	3
5	No	No	3
6	Yes	Yes	6
7	No	No	6
8	No	No	6
9	No	Yes	6
10	No	No	6
11	No	No	6
12	Yes	No	6

The GCD of 12 and 18 is 6.

We got the correct answer, but it took us 26 operations! (1 each time it divided 12, 1 each time it divided 18, and 1 each time the GCD was set)

Euler’s algorithm uses recursion to find the correct answer in much faster time! (This assumes a >= b.)

The pseudocode for the algorithm is as follows:

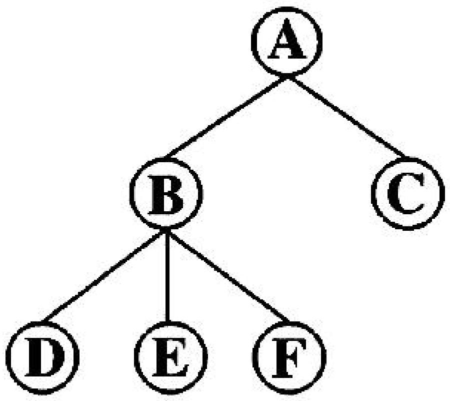
```
def gcd(int a, int b):
        if(a%b == 0)
          return b;
        return gcd(b, a-b);
```

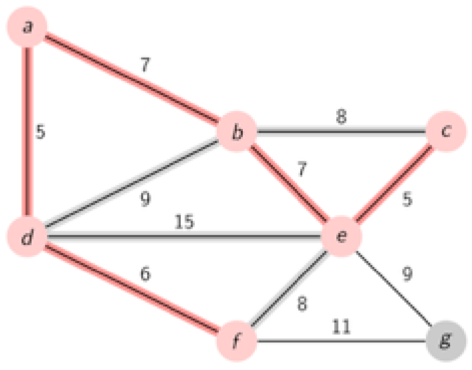
Graph Theory and Dijkstra’s Minimum Distance Algorithm

Graphs

What is a graph? (And no not the type of graph you see in math.) A graph is a network of nodes (vertices) and edges. Each node connects to other nodes through edges. Edges can be directional (meaning that you can go from Node A to B but not Node B to A) or bidirectional/ undirected (meaning that you can go from Node A to B and Node B to A). Sometimes each path may have a cost associated with it, called a weight (eg the distance from A to B).

Trees are the most common type of graph and are used in many data structues (eg a heap). The first node in a tree is called the root node. All nodes that have a path to other nodes are called parents and the “other nodes” are called children.





The diagram on the left is an example of a tree and the diagram on the right is an example of a bidirectional weighted graph.

<u>BFS (Breadth-First Search) vs DFS (Depth-First Search)</u>

BFS and DFS are both methods for traversing a graph.

In DFS, each parent node goes to its child's node children before going to the parent's other children nodes. For example the DFS order for the tree above is A, B, D, E, F, C.

In BFS, each parent node does all of its children nodes before its children nodes do their children nodes. For example the BFS order for the tree above is A, B, C, D, E, F.

<u>Dijkstra's Algorithm</u>

What is the minimum distance between nodes A and G in the second diagram. The answer is 22 (A to D to F to G). Any ideas of how to code this? Luckily for us Dutch computer-scientist Edsger Dijkstra figured it out. Basically, we start off by setting the minimum distance from the root node to the other nodes as ∞. We then set the distance from the root node to itself as 0. Then, by using a breadth-first search, we go to all of the children nodes and set their distance to be the distance of the parent plus the weight of the edge between them. The pseduocode for this is as follows.

```
def dijkstra():
    for Node n in nodes
            n.distance = Integer.MAX_VALUE;
    Queue pq = new PriorityQueue();
    pq.add(start);
    while pq.isNotEmpty
            Node n = pq.poll();
                for Node adj in n.adjacentNodes
                    adj.distance = Math.min(n.distance+n.distanceFrom(adj), adj.distance);
                    pq.add(adj);
```

Stacks vs Queues

What is a stack or queue?

When you made a recursive algorithm, have you ever had a “Stack Overflow Error?” This just means that you recurred past the point that your programming language could handle. A good way to think about stacks is like a pile of plates. When we add a plate to the stack we put it on top. Likewise, when we remove a plate from the stack we also take it from the top. This is called a LIFO (last-in, first-out) data structure.

The second kind of data structure is called a queue. Converse with a stack, a queue is a FIFO (first-in first-out) data structure. An analogy for a stack is like waiting in line for ice cream. When you enter the line, you enter from the back and must wait for everyone else to order before you can.

Floodfill

Floodfill problems are common on bronze-level USACO problems. Typically they involve counting how many lakes or asteroids there are. For the lake example, you would find a point at which there is water, then set itself and all water next to it (and all water next to that and so forth) to land and increase your count by 1. After checking each point, the number of lakes would be your count. There are two standard methods for implementing a floodfill: Breadth-First Search (BFS) and Depth-First Search (DFS) by using queues and stacks (or recursion!) respectively.

Below is the pseudocode for a BFS floodfill

```
def fill(Coordinate start, int fillTo):
    PriorityQueue<Coordinate> pq = new PriorityQueue<Coordinate>();
    pq.add(start)
    int temp = matrix.get(start)
    while pq has Elements
        Coordinate coor = pq.poll();
        if coor is in Bounds and matrix.get(coor) == temp
            pq.add(new Coordinate(coor.x-1, coor.y))
            pq.add(new Coordinate(coor.x+1, coor.y))
            pq.add(new Coordinate(coor.x, coor.y-1))
            pq.add(new Coordinate(coor.x, coor.y+1))
            matrix.set(coor, fillTo)
```

Important Data Structures

Array

This is the most common data structure that you would use. It is plain and simple with no fancy methods. However, it is also the fastest data structure. When you don’t need an outlandish way to store your data just use a plain and simply array. (In Python there aren’t arrays; instead there are Lists.)

ArrayList/Vector

ArrayLists and Vectors (yes there are some differences such as thread-safe and resizing but in this they will be treated as the same) are basically re-sizable arrays. Think of them like a rubber band that will stretch to hold your data. They are useful when you don't know how many answers you will get and like arrays have O(1) look-up time. Depending on the circumstance they could have O(n) deletion and insertion. Therefore if you're going to be constantly adding and removing data and you need the indefinite length, go for a heap or LinkedList.

Heaps/General Tree Data Structures

A heap is a type of data structure based on a binary tree. (This just means that each parent node can have up to 2 children). They are extremely useful when you need to have your data in a special order and from this technique created the Heap Sort. When sorted they have O(log n) look-up time. In a maximum heap (where the parent node is greater its children) each node has two children nodes generally called right and left. The left node is always greater than or equal to the right node (or less in a minimum heap).

PriorityQueues

A PriorityQueue, like a regular queue is a FIFO data structure. However, since PriorityQueues are very nice to use since the sort your data for you, instead of the "first element" being what was inserted into the Queue first, the first element is the smallest according to the natural ordering of the elements (eg the compareTo() method). (Cool fact: In java and many other programming languages PriorityQueues are implemented with heaps for speed.)

LinkedLists

LinkedLists is a string of nodes hooked-up to each other where each node contains a piece of data. LinkedLists are awesome for inserting data because all you have to do is add another node at the end (or front). However, they have terrible look-up time if you need to access something that isn't the front or back since you have to go to each node in between. In some problems such as the Josephus problem where you need to remove an element after a set number of spaces, it is incredibly useful, but keep in mind this happens only in certain situations.

TreeSet

Like PriorityQueues, TreeSets are nice and keep your data sorted. However, all "Sets" contain only one copy of each element so if you were to add the number 2 twice there would only be one number 2 in the set.

HashMaps

HashMaps are one of the coolest data structures I've seen. Think about it like an array or arraylist, but instead of integer indices, you give it an object (called a key) as an index to access another object. The following hashmap is an example of where we have People as keys and Books as the values.

```
HashMap<People, Book> hm = new HashMap<People, Book>();
hm.put(bessie, hitchHikersGuidToTheGalaxy);
Book awesomeBook = hm.get(bessie);
```

Stack

Again these are re-sizable and have a LIFO structure. These are what are used to create recursion in several programming languages -- hence to name Stack Overflow Error.

Use these tips to make sure you pick the right data structure for your program to function at top speed and efficiency. As a final note when you create these data structures using generics, you have to use the wrapper class for primitive types. (Eg. LinkedList<Integer> linked = new LinkedList<Integer>();). [If you didn't know, the <> are part of Generics which allow these data structures to hold any class. Simply put the class name of what you want to hold inside. Eg to hold Crêpes we would say ArrayList<Crêpe> al = new ArrayList<Crêpe>();].

Dynamic Programming (DP)

Bessie the cow (and yes if you did not know it yet all Comp-sci problems somehow involve Bessie the cow or Farmer Joe) is going on a trip to Disney and wants to bring back lots of cool souvenirs, but her suitcase can only hold 47 ft^3. If she can buy souvenirs having sizes of 2, 5, 7, and 12 cubic feet, what is the maximum number of souvenirs she can bring home assuming she can't break a souvenir into separate pieces.

The naive approach is to brute force this until you've exhausted all of the possible combinations. Unfortunately since Bessie doesn't have 3 months to wait for your program to run we'll have to come up with a faster way. This way is called dynamic programming and is very useful in a variety of USACO problems.

Dynamic programming is simply solving a problem by solving several smaller problems. In our case what if instead of a maximum size of 47, we solved for say a maximum size of 10 or even 2.

The easiest method for solving DP problems like this is to create an array of the answers to the smaller problems. In this case the index would represent the size of the suitcase Bessie and the value represents the maximum number of souvenirs. Of course in creating our array we would want it to have a length of 48 (0 to 47 inclusive).

```
int[] dp = new int[48];
```

First we would solve for index 0 (or Bessie having a suitcase size of 0). Of course this would be 0 since you can't put anything inside.

```
dp[0] = 0;
```

For each of the next indices, (dp[1] ... dp[47]) the value would be the maximum of the previous size and the suitcase with size of the leftover space of each of the souvenir plus one.

```
for(int i = 1; i<dp.length; i++)
        int max = dp[i-1];
        for(int a: possibilities)
                if(a>=i)
                        max = Math.max(max, dp[i-a]+1);
        dp[i] = max;
```

Once we've reached the end of the array (index 47) we have our answer!

Sample Problems

1. (Traditional) Given two integers, A and B (1 <= A, B <= 1,000,000,000,000), find their greatest common factor.

2. (Alex Chen, 2011) Bessie wants to buy toys from Farmer John, but Farmer John demands that Bessie pay using milk. In order to buy the toy dinosaur, Farmer John demands M (1 <= M <= 10,000) units of milk. Bessie can only produce one of K (1 <= K <= 1,000) amounts, numbered 1..K, of milk on any given day. The i-th amount is equal to P_i (1 <= P_i <= M) units of milk. On any day, Bessie can choose any of the K amounts of milk and produce that any units. After producing milk once during a day she must wait until the next day to produce more milk. Help Bessie find the minimum possible number of days she must take to produce enough milk to buy the dinosaur toy.

3. (Sherry Wu, 2011) Bessie and Canmuu found a sack of N(1 ≤ N ≤ 250) gold coins of values Vi(1 ≤ Vi ≤ 2000) that they wish to divide as evenly as possible, although that is not always possible. What is the smallest difference between the values of the two piles, and how many ways are there to split the piles with this difference?

4. (Alex Chen, 2011) You are given a set of locations, lengths of roads connecting them, and an ordered list of package dropoff locations. Find the length of the shortest route that visits each of the package dropoff locations in order.

5. (Brian Dean, 2011) Hearing that the latest fashion trend was cows with two spots on their hides, Farmer John has purchased an entire herd of two-spot cows. Unfortunately, fashion trends tend to change quickly, and the most popular current fashion is cows with only one spot! FJ wants to make his herd more fashionable by painting each of his cows in such a way that merges their two spots into one. Please help FJ determine the minimum amount must paint in order to merge two spots into one large spot.

6. (Alex Chen, 2011) Bessie likes to go to the Cownty Fair, which offers free food. Specifically, there are N (1 <= N <= 100,000) booths, numbered 1..N, each of which offers free food. Booth i offers free food at time T_i (and only at time T_i) (0 <= T_i <= 100,000). The food from booth i takes L_i (1 <= L_i <= 100,000) time to eat. When Bessie gets free food from a booth, she must eat all of the food before proceeding to get more free food. Bessie, being a hungry cow, wants to eat free food from as many different booths as possible. Help Bessie determine the maximum number of booths she can eat free food from. Assume that Bessie takes 0 units of time to travel from one booth to the next.

BONUS PROBLEMS UNRELATED TO THE LECTURE :D

7. (Alex Chen, 2011) Farmer John's cows are trying to solve a "word search" puzzle. Cow word searches are sort of strange. Instead of receiving a grid of characters, they only use a very long string, of length N (1 <= N <=100,000). Bessie wants to find the number of times her nickname appears within the very long string. Her nickname has length M (1 <= M<= N). All strings only use lower-case letters ('a'..'z'). CHECKING EVERY SUBSTRING NORMALLY WILL **NOT** RUN IN TIME.

8. (Brian Dean, Andre Kessler, 2009) Farmer John is going to the market to buy hay for his C (1 <= C <= 100) cows. Ever conscious for the comfort of his cows, Farmer John doesn't care about the price of the hay, but about its quality. The market sells hay as one extremely long barrel consisting of N (C <= N <= 100,000) equally-sized sections, each with quality value Q_i (0 <= Q_i <= 1,000,000). The hay-sellers will make at most two cuts, each cut splitting the hay barrel into pieces that are an integer number of sections in length. Farmer John will be allowed to buy exactly one of the lengths of hay that result. He needs at least C sections of hay (at least one for each of his cows), but beyond that, what he really wants to do is maximize the overall quality of the hay. Determine the maximum average quality of a section of hay that Farmer John can buy to within 0.001 (that satisfies all of his cows), and find this section. Break ties by taking the shortest section closest to the start of the barrel of hay.

9. (Brian Dean) Farmer John has hired a professional photographer to take a picture of some of his cows. Since FJ's cows represent a variety of different breeds, he would like the photo to contain at least one cow from each distinct breed present in his herd. FJ's N cows are all standing at various positions along a line, each described by an integer position (i.e., its x coordinate) as well as an integer breed ID. FJ plans to take a photograph of a contiguous range of cows along the line. The cost of this photograph is equal its size -- that is, the difference between the maximum and minimum x coordinates of the cows in the range of the photograph. Please help FJ by computing the minimum cost of a photograph in which there is at least one cow of each distinct breed appearing in FJ's herd.

What language is right for this problem?

This depends on your personal coding process and preferences. C and C++ are easy to use for simple problems, while Java requires a lot of overhead code. Java is also much less efficient but usually a handicap is given to try to make up for it. There's also a handicap for Python but unless you physically can't code anything else it'll be too slow. Finally, for the love of God don't use BrainF*ck, Malbolge or WhiteSpace. Hello world is implemented below for example.

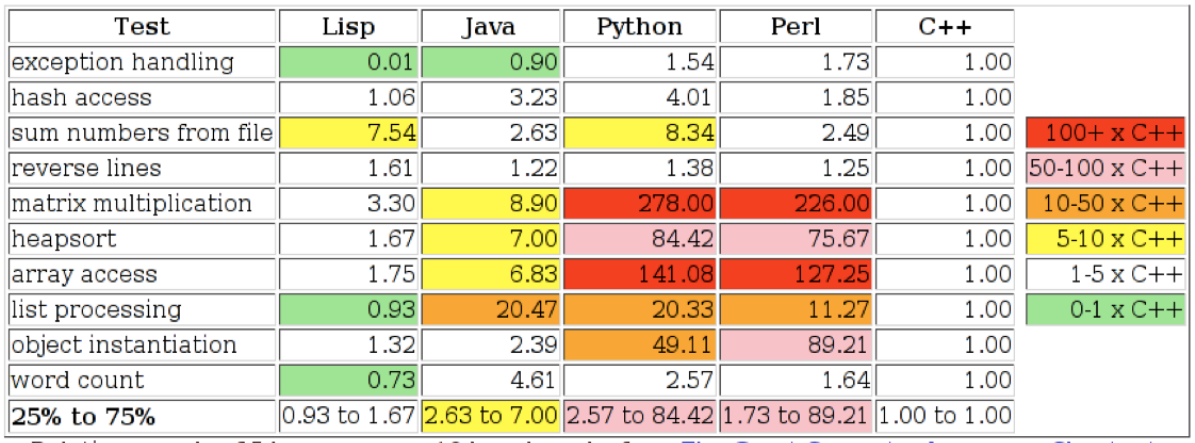
BrainF*ck:

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]
>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```

Malbolge:

```
('&%:9]!~}|z2Vxwv-,POqponl$Hjig%eB@@>}=<M:9wv6WsU2T|nm-,jcL(I&%$#"
 `CB]V?Tx<uVtT`Rpo3NlF.Jh++FdbCBA@?]!~|4XzyTT43Qsqq(Lnmkj"Fhg${z@>
```

WhiteSpace:



Test	Lisp	Java	Python	Perl	C++	
exception handling	0.01	0.90	1.54	1.73	1.00	
hash access	1.06	3.23	4.01	1.85	1.00	
sum numbers from file	7.54	2.63	8.34	2.49	1.00	100+ x C++
reverse lines	1.61	1.22	1.38	1.25	1.00	50-100 x C++
matrix multiplication	3.30	8.90	278.00	226.00	1.00	10-50 x C++
heapsort	1.67	7.00	84.42	75.67	1.00	5-10 x C++
array access	1.75	6.83	141.08	127.25	1.00	1-5 x C++
list processing	0.93	20.47	20.33	11.27	1.00	0-1 x C++
object instantiation	1.32	2.39	49.11	89.21	1.00	
word count	0.73	4.61	2.57	1.64	1.00	
25% to 75%	0.93 to 1.67	2.63 to 7.00	2.57 to 84.42	1.73 to 89.21	1.00 to 1.00	

Relative speeds of 5 languages on 10 benchmarks from The Great Computer Language Shootout. Speeds are normalized so the g++ compiler for C++ is 1.00, so 2.00 means twice as slow; 0.01 means 100 times faster. Background colors are coded according to legend on right. The last line estimates the 25% to 75% quartiles by throwing out the bottom two and top two scores for each language.

A big thanks to: Aaron Zhao, Abi Gopal, Matt Levonian, Will Bradbury, Dan Stiffler, Ellen Mulé, Quinnlan Sweeny, Bridget Andersen, and Maddie Guyant for helping proof this and add meaningful suggestions.