Compilando Conocimiento

Introducción a la Programación Competitiva

Club de Algoritmia ESCOM

Rosas Hernandez Oscar Andrés

Enero 2018

Índice general

I Introducción a la Programación Competitiva **6**

1. ¿Qué es la Programación Competitiva? **7**

1.1. Introducción 8

1.1.1. Ejemplos 8

1.2. Propiedades de un problema 9

II Un tour de C++ y lo que deberías saber de el **10**

2. ¿Porqué C++ ? **11**

2.1. Ventajas de C++ 12

2.2. vs C 14

2.3. vs Java 14

2.4. vs Python 14

3. Tipos de datos númericos **15**

3.1. Integers: Enteros 16

3.1.1. char 16

3.1.2. int 17

3.1.3. long long 17

3.1.4. unsigned 17

3.1.5. short 18

3.1.6. size_t 18

3.2. Floating Points: Puntos Flotantes 19

3.3. Complex: Números Complejos 19

4. Bases del Lenguaje 20

4.1. Condicionales 20

4.2. Ciclos 20

4.3. Funciones 20

4.3.1. Recursion: Recursividad 20

4.4. References and value types: Referencias o Valores 20

4.4.1. Value types: Variables de valor 20

4.4.2. Pointers: Apuntadores 20

4.4.3. References: Referencias 20

4.4.4. Move: Semánticas de Movimiento 20

5. Containers: Contenedores de la STD 21

5.1. std::vector 21

5.1.1. std::array 21

5.2. std::string 21

5.3. std::map 21

5.4. std::set 21

6. Cosas cool la sintaxis 22

6.1. auto 22

6.2. for (auto x : container) 22

III Ideas de las Ciencias de la Computación 23

7. La Complejidad 24

7.1. Cotas y Notaciones 24

7.1.1. Big O Notation: Notación de O grande 24

8. Optimización 25

8.1. Límites de Tiempo de Ejecución 25

8.2. Límites de Memoria 25

9. Problemas NP **26**

IV Estructuras de Datos 27

10. Arrays **28**

11. Stacks LIFO: Pilas **29**

12. Queue FIFO: Colas **30**

13. Linked Lists: Listas Enlazadas **31**

14. Binary Trees: Arboles Binarios **32**

14.1. BTS: Arboles de Búsqueda 32

14.2. AVL - RedBlackTree: Arboles Autobalanceables 32

14.3. Trie 32

15. Heaps **33**

16. Hash Tables **34**

V Algoritmos Generales 35

17. Search: Búsquedas **36**

17.1. Linear Search: Búsqueda Lineal 36

17.2. Binary Search: Búsqueda Binaria 36

17.3. Ternary Search: Búsqueda Ternaria 36

17.4. Upper Bound 36

17.5. Lower Bound 36

18. Sorting: Ordenamiento por Comparaciones **37**

18.1. Bubble Sort 37

18.2. Selection Sort 37
18.3. Merge Sort 37
18.4. Quick Sort 37

19. Sorting: Ordenamiento NO por Comparaciones 38
19.1. Bucket Sort 38

VI Programación es solo matemáticas aplicadas 39

20. Binary: Explotando el Binario 40
20.1. Bits 40
20.1.1. Manejo de Bits 40
20.1.2. Operaciones con Bits 40
20.2. Conversiones entre Sistemas 40
20.3. Binary Exponentiation: Exponenciacion Binaria 40
20.4. Binary Multiplication: Multiplicación Binaria 40

21. Roots: Encontrar Raíces de ecuaciones 41
21.1. Newton - Ranphson 41

22. Teoría de Números 42
22.1. Divisibilidad 42
22.1.1. Euclides 42
22.2. Modulos 42
22.3. Fibonacci 42
22.4. Números de Catalán 42
22.5. Primos y Factores 42
22.5.1. Eratosthenes Sieve: Criba de Eratóstenes 42
22.5.2. Prime Factorization: Factorización 42
22.5.3. Divisores 42
22.5.4. Euler Totient: La Phi de Euler 42

23.Probabilidad **43**

23.1. Inclusión Exclusión 43

24.Geometría **44**

VII Técnicas de Solución **45**

25.Ad-Hoc **46**

26.Recursividad y BackTracking **47**

27.Divide and Conquer: Divide y Vencerás **48**

28.Greedy **49**

29.Programación Dinámica **50**

VIII Grafos y Flujos **51**

30.Grafos y Gráficas **52**

30.1. Representaciones 52

30.2. BFS: Breadth-first Search 52

30.3. DFS: Depth-first Search 52

30.4. Dijsktra: Camino más cercano 52

Parte I

Introducción a la Programación Competitiva

Capítulo 1

¿Qué es la Programación Competitiva?

Figura 1.1: Imágen por Sharaft Siddiqui Reheb

1.1. Introducción

‘‘ Given well-known CS (computer science) problems,
solve them as quickly as possible! ’’

‘‘ Dados problemas famosos de ciencias de la computación,
¡resuelvelos tan rápido como puedas! ’’

- Competitive Programming 3 [1]

La programación competitiva es la actividad de resolver *problemas bastante conocidos de ciencias de la computación* mediante la creación de *programas* que obtengan la respuesta dentro de un cierto *límite*.

- **¿Problemas conocidos de ciencias de la computación?**

  Los problemas que vamos a resolver están bien definidos, en los que para cualquier entrada tu tendrías que ser capaz de resolverlo por tu cuenta.

  Además estarás informado de todas las restricciones del problema y todas las supociones que puedes tomar.

- **Tendrás que programar.**

  Si, pero no como en tú día a día, no harás una aplicación web o con una interfaz super bonita, sino que son programas que toman sus datos de la entrada estándar y nos regresan la respuesta por la salida estándar, es decir, un programa de terminal.

- **Tendrás límites.**

  Tu programa tendrá que resolver el problema con unas restricciones en tiempo y memoria, es decir, por ejemplo, tu programa tendrá que dar la solución en menos de 300ms y ocupando menos de 300MB de memoria.

1.1.1. Ejemplos

- On ta el pinche fácil pa irme? - OmegaUp
- Factores comunes - OmegaUp
- Reactores - OmegaUp

1.2. Propiedades de un problema

- Son calificados por una máquina

  Generalmente usamos onlines judges (jueces en línea) para poder saber si hemos resuelto un problema, es decir, al final del día nuestro problema lo califica un programa.

  Así que no hay puntos medios o tu programa funciona siempre y como debe o el juez te dirá que está mal.

- Tienen historia.

  Muchas veces (casi siempre) los problemas están metidos dentro de una historia, está ayuda a que sea mucho más interesante y que te cueste más entender de que trata de verdad el problema.

  Además personalmente ayuda mucho a la hora de aprender, pues es mucho más fácil que recuerdes una técnica por un problema en especial que te gusto mucho a que solo así por así como robot.

- Te dan ejemplos.

  Incluso aunque el problema te dice que es exactamente lo que te está pidiendo es mucho más fácil para los humanos entender si nos dan ejemplos.

  Esto también ayuda a que no malinterpretemos el problema y estemos resolviendo algo que no.

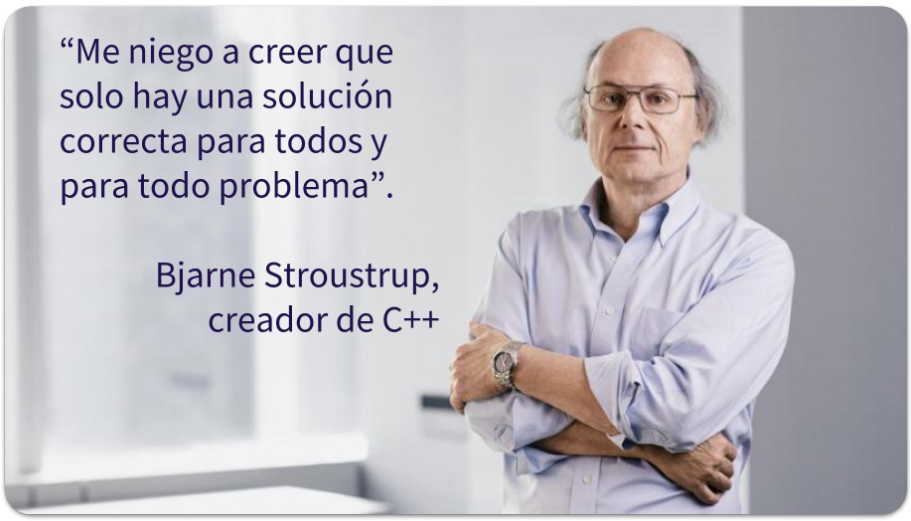
- El corazón del problema está relacionado siempre con matemáticas, lógica o ciencias de la computación.

Parte II

Un tour de C++ y lo que deberías saber de el

Capítulo 2

¿Porqué C++ ?



2.1. Ventajas de C++

Hay muchas razones por las cuales C++ es uno de los lenguajes más usados en la modernidad en la programación competitiva (si no es que el más).

Puedes escuchar un video muy bonito para mi sobre porque elegir este lenguaje:

Bjarne Stroustrup: Why I Created C++

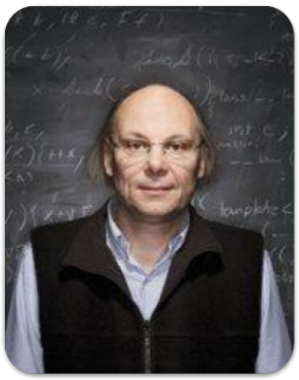


Figura 2.1: Este es el creador del lenguaje (reto: decir su nombre en voz alta) Bjarne Stroustrup

Las principales razones por C++ sin un orden en particular son:

- **Te da abstracciones sin costo extra.**

  O como lo diría su creador te da el poder de usar abstracciones de alto nivel pero estando muy cerca del hardware, o como lo diría en inglés:

  > " C++ has the zero-overhead principle: `what you don't use, you don't pay for`, that is, with every new feature added to the language, you get at least as good performance as if that feature will not be included.
  >
  > Also there are the principle: `What you do use cost either only as much as what you'd implement yourself, or it cost less`, meaning that a new feature can either mantain or improve the performance. "

  Personalmente creo que esa es mayor ventaja que tiene sobre todos los demás, creo que esa frase resume perfectamente todo el objetivo de C++ .

  Lo que nos dice esto es que podremos representar grafos, matrices, operaciones, clases en general, algoritmos, etc... en nuestros programas con gran facilidad, cosa que no podemos hacer fácilmente por ejemplo en lenguajes como C o la familia de los ensambladores.

  Y podrías decir, pero en Java o en Python podemos representar de una manera igual de fácil (siendo muy exactos no creo que sea el caso, pero esa es otra historia amigos) así que ... ¿porqué usar C++ ?

Pues porque en todos los demás lenguajes estas pagando un costo (a veces muy grande de varios cientos o miles de veces) por poder representar ideas o conceptos abstractos en un programa, en C++ esta prácticamente garantizado que usar una clase por ejemplo o que usar un arreglo que se auto dimensiona no será más costoso que si lo hubieras hecho tu desde ensamblador.

- **Tienes a la STD a tu lado**

  Esta es otra gran ventaja, ya que siguiendo con la mentalidad de abstracciones sin costo extra tenemos gracias a la gran libreria estándar un montón de cosas que no tenemos que hacer desde cero, desde arreglos que cambian de tamaño, arreglos asociativos, pilas, colas, algoritmos de ordenamiento, de búsqueda, algoritmos para acumular, para hacer particiones o permutaciones, etc...

  Así podemos dejar los algoritmos básicos al lenguaje y enfocarnos en las cosas que son de verdad interesantes.

- **Es compilado, el compilador es tu amigo**

  Otra ventaja más, podemos siempre confiar en el compilador, en que si nuestro programa compila muy probablemente está haciendo lo que debe hacer (cosa que no podemos esperar con python por ejemplo), el compilador es tu amigo, te dira en donde te equivocaste, en donde puede que hayas querido decir otra cosa y muchas veces te dará consejos, además, tras bambalinas está transformando tu código en algo que la computadora puede de verdad entender y además usará toda la información que le diste para muchas veces incluso mejorar tu código en vez de solo "traducirlo" y optimizarlo de maneras que me sorprenden personalmente.

- **Es prácticamente un "super set" de C**

  Es decir, que cualquier código válido de C es válido en C++ , esto es de gran ayuda pues C es uno de los lenguajes más conocidos por lo que puede que la sintaxis de C++ sea más fácil que entender la sintaxis de Haskell, por ejemplo.

  Otra ventaja es que al estar basados en C conserva muchas de las ventajas de C como su portabilidad, su velocidad de ejecución y la capacidad de tener un gran control de todos los recursos del sistema (memoria y tiempo de vida de un objecto, cough cough Java y su recolector de basura)

- **Tienes un gran control de todos los recursos del sistema**

  Esto es también es muy importante, pues nos dice que en C++ podemos controlar con gran lujo de detalle cuando un pedazo de memoria ya no es usado y deberíamos liberarlo ayudando con esto a aumentar la velocidad de nuestro programa o cuando queremos usar una variable local y cuando queremos usar memoria del heap, tenemos el control de decidir si queremos pasar las cosas por referencia o por valor, si deseamos mover un objecto o si una referencia no podrá ser modificada.

2.2. vs C

El gran problema con C es que es un lenguaje muy pequeño en el sentido en que todo lo tienes que hacer tu, si quieres hacer un problema que involucra cosas medio complejas todas las estructuras las tienes que codear al momento, y en un deporte de tiempo, cada segundo cuenta, así que en resumén, lo que "mata" a C es la falta de algo parecido a la std de C++ .

Aunque para problemas sencillos C también puede ser una opción, (pero ya que C++ es casi casi un superset de C podrías entonces igualde fácil hacerlo en C++).

2.3. vs Java

Con toda honestidad hay un porcentaje de la comunidad de programación competitiva que usan Java, así que si que es una opción viable, sobretodo por su gran libreria estándar y también porque en C++ no hay algo parecido a `BigInteger` y `BigDecimal` y suelen ser muchos los problemas que lo requieran, así que si bien C++ podría ser tu lenguaje por defecto es importante que también conozcas lo básico de Java (O Kotlin si quieres ser feliz).

2.4. vs Python

Python es un gran lenguaje pero tiene todas las de perder en programción competitiva pues a ser interpretado y debilmente tipado, sus programas acaban siendo muy lentos incluso usando el algoritmo correcto, eso si, hay varias aplicaciones útiles de Python, como que todos los enteros tienen infinita precisión por defecto (aka `BigInteger` como en Java).

Así que tampoco es una mala idea aprenderlo por si se necesita un día, pero definitivamente no es la mejor idea para ser tu lenguaje por defecto en programción competitiva.

Capítulo 3

Tipos de datos númericos

3.1. Integers: Enteros

Recuerda que esta sección del texto no es una introducción a la programación para alguien que nunca ha programado nada en su vida, sino solo para alguien que no sabe C++ .

C++ (y C) maneja varios tamaños estándares de enteros, el más clásico es `int`, pero no es el único, los demás solo cambian en tamaño (y por los tanto los números que podemos almacenar) y estos son, de menor a mayor: `char, short, int, long, long long`.

Ahora podemos aplicarles a estos tipos un modificador, `unsigned`, que nos permite que no puedan representar los negativos, a cambio nos da el doble de espacio, por lo que podemos representar un entero hasta el doble de grande.

Así que con esto conocido, veamos las características más detalladamente de cada tipo: Pero si quieres un resumen, C++ garantiza que:

```
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <=
  sizeof(long long);
```

3.1.1. char

A ver, técnicamente este tipo de dato como su nombre lo dice esta diseñado para almacenar un carácter, lo que pasa es que en computación (no nos compliquemos la vida con UFT ahora) un carácter como `'a'` ó `'p'` se representa internamente usando el código ASCII (deberías buscar más de esto cuando tengas tiempo).

Total, que en resumen este tipo de dato nació para almacenar un carácter de ASCII, por lo tanto es un tipo de datos numérico que va de 0 a 255.

No te preocupes si no entiendes las primeras líneas, pero si lo haces, muy bien por ti.

```
auto character {'a'};                          // character is a char!
char number {23};                              // number is a char!
('a' == 97) and ('z' == 122);                  // true: ASCII is numbers
('A' == 65) and ('Z' == 90);                   // true: ASCII is numbers
('A' + 1 == 'B') and ('Z' - 1 == 'Y');         // You can do arithmetic

auto size {"char is 1 byte"};
char minValue {-128};
char maxValue {127};
```

3.1.2. int

Es el tipo de dato numérico estándar en C++ , así que si declaras un número con auto entonces lo más probable es que sea int.

Ahora, pasa algo raro con este tipo de dato, que dependiendo de la máquina en la que compiles entonces puede tener 2 o 4 bytes de tamaño (usa el que sea más eficiente para el sistema), aun así, casi nunca compilaras en algún sistema que te diga que un int es de 2 bytes, así que para este texto usaremos que int es de 4 bytes.

```
auto number {23};                                   // number is an int!

auto size {"int is 4 bytes"};
int minValue {-2,147,483,648};
int maxValue {2,147,483,647};
```

3.1.3. long long

Este no es un tipo de dato, per se, sino que es un modificador que le puedes aplicar a int y con esto lo que logras es duplicar el tamaño de un int (suponiendo que int tenga 4 bytes de tamaño, y creeme, seguramente es así).

```
int normalVariable {};                              // It takes 4 bytes
long long int normalVariable {};                    // It takes 8 bytes
auto number {1,000,000,000,000,000,000}             // number is long long

auto size {"long long int is 8 bytes"};
int minValue {-9,223,372,036,854,775,808};
int maxValue {9,223,372,036,854,775,807};
```

3.1.4. unsigned

Lo que hace este modificador (exacto, esto tampoco es un tipo de dato) es eliminar el signo de los tipos números, es decir el entero mas bajo que vas a poder guardar va a ser el 0, pero con ello vas a lograr duplicar el máximo entero que puedes almacenar y no aumentas para nada el espacio necesario :o

```
char maxValueChar {127};
unsigned char maxValueIntUnsigned {255};

int maxValueInt {2,147,483,647};
unsigned int maxValueIntUnsigned {4,294,967,295};

long long maxValueLL {9,223,372,036,854,775,807};
unsigned long long maxValueLLUnsigned {18,446,744,073,709,551,615};
```

3.1.5. short

Hace lo inverso que `long`, en vez que duplicar el tamaño lo parte a la mitad, y ya, solo eso :v

```
auto size {"short is 2 bytes"};
short int minValue {-32,768};
short int maxValue {32,767};
```

3.1.6. size_t

Este es especial y muchas veces lo usaré como estandar de tipo numérico y es que usamos muchas veces los enteros como índice de un contenedor.

Bien pues `size_t` es un tipo de dato especial que nos da `C++` que nos asegura que será tan grande como necesitemos para usarlo como índice de cualquier contenedor.

Nota que como lo usamos para índice, este tipo de dato no tiene signo.

Por ejemplo, `std::vector, std::string, std::array` y más lo usan como índice.

3.2. Floating Points: Puntos Flotantes

3.3. Complex: Números Complejos

Capítulo 4

Bases del Lenguaje

4.1. Condicionales

4.2. Ciclos

4.3. Funciones

4.3.1. Recursion: Recursividad

4.4. References and value types: Referencias o Valores

4.4.1. Value types: Variables de valor

4.4.2. Pointers: Apuntadores

4.4.3. References: Referencias

4.4.4. Move: Semánticas de Movimiento

Capítulo 5

Containers: Contenedores de la STD

5.1. std::vector

5.1.1. std::array

5.2. std::string

5.3. std::map

5.4. std::set

Capítulo 6

Cosas cool la sintaxis

6.1. auto

6.2. for (auto x : container)

Parte III

Ideas de las Ciencias de la Computación

Capítulo 7

La Complejidad

7.1. Cotas y Notaciones

7.1.1. Big O Notation: Notación de O grande

Capítulo 8

Optimización

8.1. Límites de Tiempo de Ejecución

8.2. Límites de Memoria

Capítulo 9

Problemas NP

Parte IV

Estructuras de Datos

Capítulo 10

Arrays

Capítulo 11

Stacks LIFO: Pilas

Capítulo 12

Queue FIFO: Colas

Capítulo 13

Linked Lists: Listas Enlazadas

Capítulo 14

Binary Trees: Arboles Binarios

14.1. BTS: Arboles de Búsqueda

14.2. AVL - RedBlackTree: Arboles Autobalanceables

14.3. Trie

Capítulo 15

Heaps

Capítulo 16

Hash Tables

Parte V

Algoritmos Generales

Capítulo 17

Search: Búsquedas

17.1. Linear Search: Búsqueda Lineal

17.2. Binary Search: Búsqueda Binaria

17.3. Ternary Search: Búsqueda Ternaria

17.4. Upper Bound

17.5. Lower Bound

Capítulo 18

Sorting: Ordenamiento por Comparaciones

18.1. Bubble Sort

18.2. Selection Sort

18.3. Merge Sort

18.4. Quick Sort

Capítulo 19

Sorting: Ordenamiento NO por Comparaciones

19.1. Bucket Sort

Parte VI

Programación es solo matemáticas aplicadas

Capítulo 20

Binary: Explotando el Binario

20.1. Bits

20.1.1. Manejo de Bits

20.1.2. Operaciones con Bits

20.2. Conversiones entre Sistemas

20.3. Binary Exponentiation: Exponenciacion Binaria

20.4. Binary Multiplication: Multiplicación Binaria

Capítulo 21

Roots: Encontrar Raíces de ecuaciones

21.1. Newton - Ranphson

Capítulo 22

Teoría de Números

22.1. Divisibilidad

22.1.1. Euclides

22.2. Modulos

22.3. Fibonacci

22.4. Números de Catalán

22.5. Primos y Factores

22.5.1. Eratosthenes Sieve: Criba de Eratóstenes

22.5.2. Prime Factorization: Factorización

22.5.3. Divisores

22.5.4. Euler Totient: La Phi de Euler

Capítulo 23

Probabilidad

23.1. Inclusión Exclusión

Capítulo 24

Geometría

Parte VII

Técnicas de Solución

Capítulo 25

Ad-Hoc

Capítulo 26

Recursividad y BackTracking

Capítulo 27

Divide and Conquer: Divide y Vencerás

Capítulo 28

Greedy

Capítulo 29

Programación Dinámica

Parte VIII

Grafos y Flujos

Capítulo 30

Grafos y Gráficas

30.1. Representaciones

30.2. BFS: Breadth-first Search

30.3. DFS: Depth-first Search

30.4. Dijsktra: Camino más cercano

Bibliografía

[1] Competitive Programming 3, *Halim and Halim, 2013*.