Unidad 1

Fundamentos de Programación Competitiva

Logro de sesión

Al finalizar la sesión, el estudiante comprenderá conceptos de la programación competitiva y características del lenguaje de Programación C++.

Semana 1

Introducción a la Programación Competitiva

Contenido:

- Conceptos de Programación competitiva
- Características C/C++: entrada y salidas, tipos de datos.

Introducción

La directriz central en “Programación competitiva”

"¡Dados los problemas conocidos de Ciencias de la Computación (CS), resuélvalos lo más rápido posible!".

Programación Competitiva

- Resolver los problemas implica que debemos llevar nuestro conocimiento de Ciencias de la Computación a un cierto nivel requerido para que podamos producir un código de trabajo que también pueda resolver estos problemas,

- La necesidad de resolver el problema "lo más rápido posible" es donde radica el elemento competitivo: la velocidad es una meta muy natural en el comportamiento humano.

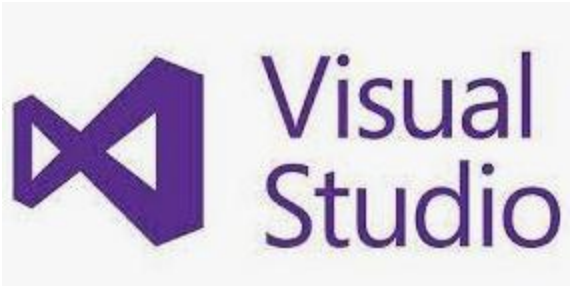
Que conocimientos necesitamos

- Matemáticas
- Algoritmos
- Conocimiento de lenguajes de programación
- Estructuras de datos

Lenguaje de Programación

- Se utilizará C++, es un lenguaje de programación, creado a mediados de 1979 por Bjarne Strousstrup, como extensión del lenguaje C. Este lenguaje abarca tres paradigmas de la programación:
  - Programación Estructurada
  - Programación Genérica
  - Programación Orientada a Objetos

Zona de trabajo



OnLine:

- https://repl.it/languages/cpp
- http://codepad.org/

Otras aplicaciones:

- DEV C++ :
- Visual Code:

Importantes torneos

- ACM – ICPC: Competencia de Programación más importante a nivel mundial. Organizada por la ACM.
  https://icpc.global/

- IEEExtrem: Competencia organizada por la IEEE una duración de 24 horas.
  http://ieeextreme.org/

Características de C++

- C++ hereda la sintaxis de C estándar.
- El punto y coma “;”, se usa para indicar el final de una línea de instrucción, además también se usar para separar contadores, condicionales e incrementadores.
- Comentarios, cuenta con 2 tipos:
  - // Comenta solo una línea de código.
  - /*Instrucción*/ Comenta párrafos

Características de C++

❑ Tipo de datos

TIPO DE DATO	CONCEPTO	TAMAÑO	RANGO DE VALORES
Char	Caracter o entero	1 byte	-128 a 127
Short	Entero corto	2 bytes	-32768 a 32767
Int	Entero	4 bytes	-2147483648 a 2147483647
Bool	Booleano	1 byte	True o false
Long long	Entero largo	8 bytes	-92233720368547758 08 a 92233720368547758 07
Float	Flotante	4 bytes	3.4e +/- 38 (7 dígitos)
double	Flotante con doble precisión	8 bytes	1.7e +/- 308 (15 dígitos)

Características de C++

- Variables, es un elemento que puede cambiar durante el proceso de ejecución.
  - int numero;
  - char c;
  - double num;
- Constante, cuyo valor no puede ser alterado durante el proceso de compilación.
  - const char letra;
  - const int n;

Características C++

```
#include<iostream>
using namespace std;
int main()
{
int n1, n2, suma;
Cin>>a;
Cin>>b;
suma = n1 + n2;
cout<<"\nLa suma es: "<<suma;
return 0
}
```

Librería

Funciones

Función principal

Declaración de variables

Entrada de datos

Operación

Salida de datos

Si retorna 0, no tiene errores

Características de C++

- C++ cuenta con diferentes operadores:

OPERADORES ARITMÉTICOS	
+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo de la división
++	Suma un valor de la variable
--	Resta un valor de la variable

OPERADORES LÓGICOS	
&&	And
||	OR
!	Negación
&	Dirección de memoria
*	Puntero

Características de C++

- Estructuras de Control Selectivas
  - if: condición

```
if (n == 5)
cout<<"\nEl valor de n es 5";
```

  - switch: condición múltiple

```
switch (opción)
{ case 1: cout<<"\nRojo"; break;
  case 2: cout<<"\nVerde"; break;
  default: cout<<"\nError";
}
```

Características de C++

- Estructuras de Control Repetitivas
  - for: (para) bucle

```
for(int i=0;i<10;i++)
    { cout<<i; }
```

  - while: (mientras) bucle

```
while (numero<=5)
{ cout<<"\nVerde";
  numero++;
}
```

Características de C++

- Estructuras de Control Repetitivas

  - do - while: (hacer - mientras) bucle

```
do
{
     cout<<"\nNumero: ";
     cin>>n;
}while(n < 1);
```

Ejercicios

- Para entender los conceptos, se usará como referencia el siguiente problema.

Primer Problema – online judge

Jueces online

HackerRank

UVa Online Judge

CODEFORCES β
Sponsored by Telegram

LeetCode

Envío de código

1. Subir el código fuente al juez online.
2. Esperar, el juez probará tu programa con un input secreto.
3. Te dará uno de los siguiente veredictos.

AC (accepted) – Tu solución es correcta.
WA (wrong answer) – Tu programa da una respuesta incorrecta.
TLE (time limit exceeded) – Tu programa tarda mucho tiempo.
RE (runtime error) – Tu programa se cae durante su ejecución.
CE (compilation error) – Tu programa no compila.

Técnicas

En programación competitiva ¿Cuáles son las técnicas para resolver algoritmos más eficientes?

Técnicas

- Algunas técnicas

Técnica	Complejidad	Uso
STL (`map` , `set` , `priority_queue`)	O(1) - O(log N)	Buscar, ordenar, contar
Búsqueda Binaria	O(log N)	Buscar en datos ordenados
Dos punteros	O(N)	Suma de subconjuntos, ventanas deslizantes
Programación Dinámica	O(N)	Problemas de optimización
BFS/DFS en grafos	O(N + M)	Recorrer grafos
Dijkstra	O((N + M) log N)	Camino más corto en grafos

Técnicas

Técnica	Descripción	Complejidad típica	Ejemplos de uso
Complejidad algorítmica	Analiza la eficiencia del algoritmo usando notación Big-O.	Depende del algoritmo	Elección de estructuras de datos eficientes.
Divide y vencerás	Divide el problema en subproblemas, resuélvelos y combina las soluciones.	$O(n \log n)$ (común)	Merge Sort, Búsqueda Binaria.
Programación dinámica (DP)	Almacena resultados de subproblemas para evitar recalcularlos.	$O(n)$ o $O(n^2)$ (común)	Fibonacci, Problema de la mochila.
Algoritmos voraces (Greedy)	Toma decisiones locales óptimas en cada paso para alcanzar una solución global.	$O(n \log n)$ (común)	Dijkstra, Problema de la mochila fraccional.
Backtracking con poda	Explora todas las soluciones posibles, eliminando ramas no óptimas.	$O(2^n)$ (mejora con poda)	Problema de las N reinas.

Técnicas

Técnica	Descripción	Complejidad típica	Ejemplos de uso
Grafos y árboles	Aplica algoritmos clásicos para recorrer o analizar grafos y árboles.	$O(V + E)$ (común)	BFS, DFS, Dijkstra, Kruskal.
Manipulación de bits	Usa operaciones a nivel de bits para optimizar cálculos.	$O(1)$ por operación	Verificar potencias de 2, contar bits.
Matemáticas y teoría de números	Aprovecha propiedades matemáticas para optimizar algoritmos.	$O(\log n)$ (común)	Criba de Eratóstenes, GCD, exponenciación rápida.
Técnicas de optimización	Usa métodos como memorización, sliding window o two pointers.	$O(n)$ (común)	Problemas de subarreglos, búsqueda eficiente.
Bibliotecas estándar	Aprovecha funciones y estructuras optimizadas de tu lenguaje.	Depende de la función	STL en C++, `collections` en Python.

Muchas Gracias!!!