ALGORITMOS Y ESTRUCTURAS DE DATOS
2023/2

Proyecto 3

El presente documento entrega la descripción del escenario relacionado con el problema a resolver de la Unidad 3.

Objetivos

- Realizar la solución utilizando el lenguaje de programación C++.
- Realizar el análisis de un problema y aplicar el diseño de una solución mediante la implementación de algoritmos de ordenamiento.
- Realizar el análisis y comparación de los algoritmos implementados.
- Realizar una correcta declaración de variables según corresponda.
- Realizar una correcta implementación de algoritmos que involucren decisiones y bucles.
- Realizar una correcta implementación de entrega de resultados por la salida estándar.

Requerimientos de entrega

El Software desarrollado debe cumplir con los siguientes requerimientos:

- El programa debe estar completamente funcional, con ausencia de errores y alertas (warnings).
- El programa debe estar bien documentado para lo cual se necesita:
    - La inclusión de un archivo README donde se explica la solución desarrollada considerando el formato recomendado para este tipo de archivos. [1]
    - Presencia de comentarios que expliquen el código, considerando la descripción de estructura, clases, métodos y funciones.
- La inclusión de un archivo INSTALL donde se explica el procedimiento para instalar el programa (compilar).
- El programa debe encontrarse en un repositorio en su cuenta Github.

El programa debe ser implementado a través del uso del lenguaje de programación C++ y menús interactivos en un contexto práctico relacionado con el problema entregado.

[1] Formato README

The guardians library-basics

Lircay Hub está interesado en comenzar a trabajar en las versiones beta de los juegos “The clash of the guardians” y “The guardians battles” para lo cual es fundamenten sumar algunas funcionales fundamental a ser utilizada en estos y posiblemente en otros juegos de la empresa.

Para esto es necesario realizar la búsqueda de los mejores algoritmos que permitan generar las librerías con el propósito de optimizar la lógica de implementación. Estos algoritmos en una primera instancia sería utilizados en:

- **Colas de espera:** La idea es poder ordenar a los jugadores que se encuentran en la búsqueda de partidas dentro de los juegos. Aquí es necesario poder contar con un algoritmo que pueda realizar cálculos en tiempo real para obtener los jugadores con mayor prioridad en base a ciertas características.
- **Trazabilidad de objetos:** La idea es poder ordenar la lista de objetivos según su prioridad además del estatus que tiene dentro del juego. Otra característica para considerar en el orden son la posibilidad de unirse para formar otros objetos. Aquí es necesario poder contar con un algoritmo que permita ordenar los objetos según su prioridad y orden de progreso dentro del juego.
- **Eventos de cada escenario:** La idea es poder ordenar los eventos y las interacciones que puede tener los jugadores en distintas etapas o niveles del juego. Los eventos también tiene posibles respuestas de los jugadores que aumentan según su evolución en niveles e ítems disponibles. Aquí es necesario un algoritmo que permita ordenar los eventos para generar distintos tipos de dificultades para mantener a los jugadores con un nivel alto de incertidumbre y entusiasmo.

Para cada una de las áreas de uso se tiene la estimación de la siguiente cantidad de elementos:

Colas de espera: Se espera tener hasta un rango de **100.000 a 110.000** de jugadores en la primera versión.

Trazabilidad de objetos: Aquí se considera mantener una variación de objetos totales disponibles entre **1000 y 1500** por cada categoría. Al día de hoy se cuenta con **15** categorías en total que conforman la vestimenta completa de los guardianes.

Eventos de cada escenario: Al ser los juegos abiertos a que puedan presentar distintas alternativas de interacciones se considera dentro de las combinaciones posible tener entre **60.000 a 80.000** posibles combinaciones de eventos.

Especificaciones de las pruebas

El objetivo es el desarrollo de un programa en C + + que permita realizar una comparación de distintos algoritmos, en base a **distintas entradas de datos**, con el propósito de determinar, para cada uno de los casos, el algoritmo que puede ser incluido dentro de las librerías.

Para trabajar con **distintas entradas de datos,** debe tener las siguiente consideraciones para **generar las entradas de datos**:

- Aleatorio: Un arreglo de datos de orden aleatorio sin la repetición de elementos.
- Aleatorio con duplicados: Un arreglo de datos de orden aleatorio con la posibilidad de tener elementos duplicados.
- Ordenado: Un arreglo de datos de orden según lo establecido.
- Inversamente ordenado: Un arreglo de datos de orden inverso según lo establecido.

Para realizar la comparación de los distintos algoritmos se realizan carreras. Las carreras se realizan de la siguiente manera:

- Todos los algoritmos deben tratar de ordenar la misma entrada de datos.
- Los algoritmos se ejecutan por separado y para cada uno se debe almacenar el tiempo que demoró en realizar el ordenamiento de la entrada de datos.
- Las carreras están asociadas a las áreas en las cuales serán utilizados, en este caso existen tres; Colas de espera, Trazabilidad de objetos y Eventos de cada escenario.
- Se debe hacer una carrera por cada área y aplicando la combinación de todas las entradas de datos.

A continuación se detalla un ejemplo, para el caso de las “Cola de espera” y para la entrada de datos “Ordenado” (solo se utilizaron 3 algoritmos de ejemplo).

```
Carrera Cola de espera, Modo Ordenado
1. NombreAlg1, t
2. NombreAlg3, t
3. NombreAlg2, t

El ganador es:NombreAlg1 un tiempo de t segundos
```

El resultado del programa es la salida del resultado de las carreras (ejemplo anterior) relacionadas con las tres áreas y los 4 modos de entradas de datos.

Desarrollo de un producto de Software

El programa debe contener:

- La inclusión de comentarios en la estructura(s) y funciones.
- Desarrollar un menú que permite navegar por las distintas opciones que tiene el programa. Dentro de las opciones deben estar la opción de seleccionar una de las tres carreras además de salir.
- Desarrollar la funcionalidad que determina de manera aleatoria el total de elementos para cada carrera según el rango entregado dentro del problema.
- Desarrollar la funcionalidad que permite implementar los elementos a través del modo ordenado e inversamente ordenado dentro del set de datos.
- Desarrollar la funcionalidad que permite implementar los elementos aleatorios duplicados dentro del set de datos
- Desarrollar la funcionalidad que permite implementar los elementos aleatorios únicos dentro del set de datos.
- Desarrollar la implementación del algoritmo selection sort
- Desarrollar la implementación del algoritmo bubble sort
- Desarrollar la implementación del algoritmo insertion sort
- Desarrollar la implementación del algoritmo shell sort
- Desarrollar la implementación del algoritmo merge sort
- Desarrollar la implementación del algoritmo quick sort
- Desarrollar la implementación del algoritmo heap sort
- Desarrollar la funcionalidad que permite determinar el tiempo de ejecución de cada algoritmo.
- Desarrollar la estructura de datos para almacenar los resultados de los algoritmos para entregar el resumen de la carrera e implementar la interfaz solicitada.
- Desarrollar la funcionalidad que permite determinar el ganador de cada carrera y presentar el resultado según lo solicitado.

Presentación de la solución

La entrega del proyecto debe ser acompañada por un video, de no más de 7 minutos en los que se debe:

- Desarrollar un material de apoyo (presentación) que sirva como guía para el video.
- El video debe estar enfocado en presentar los aspectos que diferencia a los algoritmos entre sí, los resultados obtenidos además de señalar para cada uno la notación en el peor y mejor caso
- Temas a incluir en el video respecto a los algoritmos:
    - Breve descripción de cada algoritmo.
    - Notación asintótica del algoritmo para el peor y mejor caso.
- Temas a incluir en el video respecto al programa desarrollado:
    - Exponer los aspectos claves de su solución y ejemplo de los resultados. Aquí puede realizar cortes en la grabación para que pueda presentar todo en los 7 minutos señalados. El video será revisado en clases y se realizarán consultas.

Entregable

El entregable corresponde al link de acceso público al repositorio de la cuenta de github. **En el archivo README del repositorio debe incorporar el LINK para el video.**

Porcentajes y ponderación

#	Indicadores	Ponderación
1	Realiza la gestión del proyecto a través de un repositorio y presenta a lo menos cuatro registros de commits realizados durante el desarrollo.	2%
2	Realiza una correcta descripción del Software junto con la utilización del formato recomendado para la elaboración del archivo README.	2%
3	Realizar una correcta descripción de las instrucciones a realizar para una correcta instalación y/o compilación del producto de Software.	2%
4	Desarrolla una solución con ausencia de errores y alertas.	2%
5	Desarrolla de manera correcta comentarios en la estructura(s) y funciones.	2%
6	Desarrolla la funcionalidad que entrega la posibilidad de seleccionar las opciones a través de un menú (acceder a las distintas carreras)	2%
7	Desarrolla la funcionalidad que determina de manera aleatoria el total de elementos para cada carrera según el rango entregado dentro del problema.	2%
8	Desarrolla la funcionalidad que permite implementar los elementos a través del modo ordenado e inversamente ordenado dentro del set de datos	2%
9	Desarrolla la funcionalidad que permite implementar los elementos aleatorios duplicados dentro del set de datos	3%
10	Desarrolla la funcionalidad que permite implementar los elementos aleatorios únicos dentro del set de datos.	3%
11	Desarrolla la implementación del algoritmo selection sort	5%
12	Desarrolla la implementación del algoritmo bubble sort	5%
13	Desarrolla la implementación del algoritmo insertion sort	5%
14	Desarrolla la implementación del algoritmo shell sort	6%
15	Desarrolla la implementación del algoritmo merge sort	6%
16	Desarrolla la implementación del algoritmo quick sort	6%
17	Desarrolla la implementación del algoritmo heap sort	6%
18	Desarrolla la funcionalidad que permite determinar el tiempo de ejecución de cada algoritmo.	5%
19	Desarrolla la estructura de datos para almacenar los resultados de los algoritmos para entregar el resumen de la carrera e implementa la interfaz solicitada.	5%
20	Desarrolla la funcionalidad que permite determinar el ganador de cada carrera y presentar el resultado según lo solicitado.	4%

21	Realizar la construcción de material multimedia (video) para ser entregado según los requerimientos solicitados (video y publicación).	5%
22	Desarrolla material de apoyo para ser expuesto en el video según lo solicitado en relación a los algoritmos, notación y el programa desarrollado.	10%
23	Desarrolla una exposición exposición de la solución desarrollada utilizando un lenguaje técnico y formal	10%
	TOTAL	**100%**

Ayuda

```
#ifdef _WIN32
#include <Windows.h>
#else
#include <unistd.h>
#endif
#include <cstdlib>
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include <random>

using namespace std;
//https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/

void SelectionSort(vector<int>& arr) {
    int n = arr.size();
    for (int i = 0; i < n ; ++i) {
        for (int j = i; j < n ; ++j) {
            if (arr[j] < arr[i]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

double getResultFromAlg(vector<int>& arr,int option) {
    time_t start, end;
    double time_taken;
    time(&start);
    ios_base::sync_with_stdio(false);
```

```
    SelectionSort(arr);
    time(&end);
    time_taken = double(end - start);
    return time_taken;
}

int main(int argc, char* argv[]) {

    cout << "Generando set de datos: " << endl;

    for (int i = 0; i < amount ; ++i) {
        arrSorted.push_back(i+1);
        arrReverse.push_back(amount-i);
        if ( i == 0 ) {
            random_value = 1 + rand() % (amount);
            arr.push_back(random_value);
        }
        else {
            random_value = 1 + rand() % (amount);
            arr.push_back(random_value);
        }
    }

    unordered_map<string, double> results;
    results["SeleciontSort"] = getResultFromAlg(arr);

    vector<int> arr1,arr2;
    arr1.assign(arr.begin(), arr.end());
    arr2.assign(arr.begin(), arr.end());
    int id = 1;
    for (const auto& pair : results) {
        const string& key = pair.first;
        double value = pair.second;
        cout << id << ". " << key << ", " << fixed << value << setprecision(5)
<< endl;
        id++;

    }
    return 0;
}
```

```cpp
#include <iostream>
#include <chrono>

using std::cout;
using std::endl;
using std::chrono::high_resolution_clock;
using std::chrono::duration;
using std::chrono::duration_cast;

auto myFunction() {
    // Perform some computation
    auto start = high_resolution_clock::now();
    for (int i = 0; i < 100000; ++i) {
        // Do something
    }
    auto end = high_resolution_clock::now();
    return duration_cast<duration<double>>(end - start);
}

int main() {
    auto time_taken = myFunction();
    cout << "Execution time: " << time_taken.count() << " seconds" << endl;
    return 0;
}
```