Actividad Integradora 2

Estructuras de Datos

21 de abril del 2023

Grant Keegan | A01700753

Cinthya Villalobos | A01751678

Programación de estructuras de datos y algoritmos fundamentales (Gpo 850)

Profesor: Dr. Eduardo Arturo Rodríguez Tello

En esta segunda actividad integradora, vamos a escribir un código que utilice una mezcla de diferentes algoritmos y estructuras de datos que hemos aprendido durante el semestre. Principalmente un algoritmo de ordenamiento merge, y una lista doblemente ligada.

El objetivo de esta actividad es utilizar los datos dentro de un archivo .txt para agregarlas a una estructura de datos de una lista doblemente ligada. Una vez ahí, debemos de ordenarla utilizando un algoritmo de ordenamiento merge. La lista ordenada deberá de ser transferida a un nuevo archivo .txt. También será posible solamente agregar una serie de líneas que es ingresada por el usuario.

Parte 1: Escribiendo el algoritmo de ordenamiento

En esta actividad, debemos ordenar la lista por medio del algoritmo merge sort. Esto lo hicimos escribiendo el algoritmo paso por paso. Primeramente escribimos dos funciones principales, la de merge(), y la de mergeSort().

La función de un algoritmo de ordenamiento merge es dividir la lista en 2 mitades que ordenara antes de juntarlas a una lista más grande, y así sucesivamente hasta que el algoritmo haya ordenado todas las sublistas.

En la función merge(), nosotros escribimos código para las operaciones que deben de hacer para dividir la lista en 2, y así poder ordenarlas por medio de mergeSort().

```cpp
// Algoritmo Merge Sort | Divide en 2 y vuelve a juntarlos ordenados.
// Complejidad = O(nlog)
// log n) La funcion junta los arrays divididos.
void merge(int arr[], int left, int mid, int right) {
  int n1 = mid - left + 1;
  int n2 = right - left;
  int L[n1], R[n2];
  for (int l1 = 0; l1 < n1; l1++)
    L[l1] = arr[left + l1];
  for (int l2 = 0; l2 < n2; l2++)
    R[l2] = arr[mid + 1 + l2];
  // Contienen las listas de subarrays.
  int l1, l2, lm;
  l1 = 0;
  l2 = 0;
  lm = left;
  // Fusiona los arreglos de regreso a arr[].
  while (l1 < n1 && l2 < n2) {
    if (L[l1] <= R[l2]) {
      arr[lm] = L[l1];
      l1++;
    } else {
      arr[lm] = R[l2];
      l2++;
    }
    lm++;
  }
  // Loop del array izquierdo.
  while (l1 < n1) {
    arr[lm] = L[l1];
    l1++;
    l2++;
  }
  // Loop del array derecho.
  while (l2 < n2) {
    arr[lm] = R[l2];
    l2++;
    lm++;
  }
}
```

```cpp
// Divide el array en 2, los ordena y los junta por medio de merge().
void mergeSort(int arr[], int left, int right) {
  if (left < right) {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
  }
}

// Imprime el array ordenado.
void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++)
    cout << arr[i] << " ";
  cout << endl;
}
```

El código de la función mergeSort(), a diferencia se encarga de ordenar las listas e integrarlas a una lista ordenada más grande. Con esta parte del código, pudimos lograr ordenar distintas listas ordenadas numéricamente.

Parte 2: La lista doblemente ligada

Una lista doblemente ligada es una estructura de datos donde utilizaremos nodos que programamos así:

```cpp
// nodo de lista encadenada
class Node {
  public:
  int data;
  Node* next;
  Node* prev;
};
```

Tiene el objetivo de apuntar a nodos previos y subsecuentes en la lista encadenada. Lo que hace que pueda ir en varias direcciones.

También escribimos código para la estructura de la lista doblemente ligada. Sin embargo, no nos dio tiempo de poder implementarla con el resto del programa, por lo que quedó sin utilizar.

```cpp
// Codigo para Doubly Linked List.
void push(Node** head_ref, int new_data) {
  Node* new_node = new Node();
  new_node->data = new_data;
  new_node->next = (*head_ref);
  new_node->prev = NULL;
  if ((*head_ref) != NULL)
    (*head_ref)->prev = new_node;
  (*head_ref) = new_node;
}

// Esta función imprime el contenido de la lista
void printList(Node* node)
{
  Node* last;
  cout<<" \nImprime lista hacia al frente \n " ;
  while (node != NULL)
  {
    cout<<" "<<node->data<<" ";
    last = node;
    node = node->next;
  }
  cout<<"\nImprime lista hacia atras \n " ;
  while (last != NULL)
  {
    cout<<" "<<last->data<<" ";
    last = last->prev;
  }
}
```

Parte 3: Abriendo los contenidos del archivo

Para la última parte de nuestro proyecto debemos de abrir un archivo .txt para procesar sus contenidos en nuestro programa. Para esto utilizamos varias líneas de código en main que pueda abrir el archivo y procesarlo en una lista. Utilizamos el archivo llamado nums.txt. para probar y nos funcionó, pero no nos dio tiempo de implementarlo al archivo completo de bitácora.

```cpp
// Las siguientes lineas del codigo se encargan de abrir el archivo:
ifstream inputFile("nums.txt"); // Ingresa el archivo con el que trabajaremos.
const int max_lines = 20000; // El maximo numero de lineas.
string lines[max_lines];
if(inputFile.is_open()) {
  string line; int i = 0;
  while (getline(inputFile, line) && i < max_lines) { // Loop para guardar las lineas en un array.
    lines[i] = line;
    i++;
  }
  inputFile.close(); // cierra el archivo.
```

Reflexión Individual - Grant

En esta actividad utilicé de nuevo uno de los algoritmos de ordenamiento que usamos en actividades pasadas. Esto me permitió darme cuenta de la importancia

de saber para qué sirven y cómo están construidos, ya que en toda la programación, siempre me encontraré con listas.

El concepto de las estructuras de datos, empezando por las listas ligadas, se me hace un tema útil e interesante en el que me gustaría seguir mejorando.

Reflexión Individual - Cinthya

…