Campus Santa Fe

Programación de estructuras de datos y algoritmos fundamentales

Integrantes del equipo:

Andres Tarazona Solloa - A010233332

Profesor:

Dr. Esteban Castillo Juarez

Fecha:

3 de octubre de 2022

Para añadir elementos desde el inicio de la lista, es necesario tomar la inserción en cuenta como si fuera una lista vacía, la diferencia estando en que el primer índice no está vacío necesariamente:

```
void pushInicio(int numero) {
  struct nodo *nuevo = malloc(sizeof(struct nodo));
  if (nuevo == NULL) {
    printf("No hay memoria disponible");
  } else {
    nuevo->valor = numero;
    nuevo->siguiente = primero;
    primero = nuevo;
  }
}
```

Tomamos el nuevo nodo, le damos el valor que nos place, luego igualamos el siguiente al que era el primero antes de esta inserción, y pasamos el pointer de primero a nuestro nuevo valor, así volviendolo el primer valor de la lista ligada.

Para buscar el número de repeticiones de cualquier número deseado, se debe de recorrer la lista (algo como cuando insertamos un número al final de la lista), y de ahí se implementa un if que checa si el valor del apuntador temporal es igual al número deseado. Si así es, entonces se incrementa el número total de repeticiones.

```
void repeticion(int numero) {
  struct nodo *temp;
  int i = 0;
  int pos = 0;
  temp = primero;
  while (temp != NULL) {
    if (temp->valor == numero) {
      i++;
      printf("Posicion: %d \n", pos);
    }
    temp = temp->siguiente;
    pos++;
  }
  printf("El numero %d se repite %d veces. \n", numero, i);
}
```

Para eliminar los números pares de la lista ligada, debemos igualmente recorrerla para que se puedan checar todos los valores que esta posee. Una diferencia entre esta eliminación, y la eliminación que se impartió en clase, es que es necesario checar si el numero esta al principio, en medio, o al final de la lista, de la siguiente manera:

```c
void eliminarPares() {
  struct nodo *actual = primero;
  struct nodo *anterior = NULL;
  while (actual != NULL) {
    struct nodo *temp = actual;
    if ((actual->valor % 2) == 0) {
      if (anterior == NULL) {
        primero = actual->siguiente;
        free(temp);
        actual = primero;
      } else if (anterior != NULL && actual->siguiente != NULL) {
        anterior->siguiente = actual->siguiente;
        free(temp);
        actual = anterior->siguiente;
      } else if (anterior != NULL && actual->siguiente == NULL) {
        anterior->siguiente = NULL;
        free(temp);
        actual = NULL;
      }
    } else {
      anterior = actual;
      actual = actual->siguiente;
    }
  }
}
```

Igualmente, en este caso ya que existe la posibilidad de que nuestro apuntador temporal se libere, debemos volverlo a declarar al principio de la loop while, sino se llega a un segmentation fault.

Otra vez recorremos la lista ligada, y al haber creado un valor de suma, y uno de índice, podemos agregar la suma total, y el número total de valores presentes en la lista, así entonces sacando su valor promedio. Usamos floats para tener acceso a números decimales.

```c
void promedio() {
  struct nodo *temp;
  temp = primero;
  float i = 0;
  float suma = 0;
  while (temp != NULL) {
    suma += temp->valor;
    i++;
    temp = temp->siguiente;
  }
  printf("El promedio de la lista es: %.2f \n", suma / i);
}
```

Finalmente, para eliminar ocurrencias de un mismo número dentro de la lista ligada, debemos implementar una situación sumamente similar a aquella cuando eliminamos los números pares, solo que cambia la condición dentro del if.

```c
void eliminarOcurrencias(int numero) {
  struct nodo *actual = primero;
  struct nodo *anterior = NULL;
  while (actual != NULL) {
    struct nodo *temp = actual;
    if (actual->valor == numero) {
      if (anterior == NULL) {
        primero = actual->siguiente;
        free(temp);
        actual = primero;
      } else if (anterior != NULL && actual->siguiente != NULL) {
        anterior->siguiente = actual->siguiente;
        free(temp);
        actual = anterior->siguiente;
      } else if (anterior != NULL && actual->siguiente == NULL) {
        anterior->siguiente = NULL;
        free(temp);
        actual = NULL;
      }
    } else {
      anterior = actual;
      actual = actual->siguiente;
    }
  }
}
```

Entonces finalmente, en el main, implementamos las inserciones usando rand().

```c
int main() {
    srand(time(NULL));
  printf("Uso de una lista ligada: \n");
  printf("Inserciones \n");
  for (int i = 0; i < 10; i++) {
    pushFinal(rand() % 10);
  }
  printf("Inserciones terminadas \n");
  imprimirLista();
  printf("Longitud: \n");
  longitudLista();
  printf("Repetición: \n");
  repeticion(3);
  imprimirLista();
  promedio();
  imprimirLista();
  printf("\n");
  printf("Eliminacion de un nodo \n");
  eliminarLista(3);
  eliminarLista(4);
  eliminarLista(1);
  imprimirLista();
  printf("Insercion al inicio \n");
  for (int i = 0; i < 10; i++) {
    pushInicio(rand() % 10);
  }
  imprimirLista();
  printf("Eliminacion de pares \n");
  eliminarPares();
  imprimirLista();
  printf("Eliminacion de 5 \n");
  eliminarOcurrencias(5);
  imprimirLista();
}
```