Algoritmos e estruturas de dados em C

Objetivo geral

- Pode-se dizer, de modo geral, que estrutura de dados visam recuperar/organizar informações de modo especifico, e da melhor forma possível.

Busca simples

- Basicamente, o metodo mais simples para se encontra um valor em uma lista. Passando por todos os itens da lista até encontra-lo.
- Esse algoritmo possui como velocidade O(n).
- Exemplo de algoritmo:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

bool buscaSimples(int * vetor, int valor_procurado, int tamanho_vetor){
bool valor_foi_encontrado = false;
for(int x = 0; x < tamanho_vetor; x++){
if(vetor[x] == valor_procurado){
valor_foi_encontrado = true;
printf("Valor [%d] encontrado na posição %d", valor_procurado, x);
}
}
if(valor_foi_encontrado == false)
printf("Valor não encontrado!");
return valor_foi_encontrado;
}

void main(){
bool foi_encontrado;
int tamanho_vetor, valor_procurado;
int * vetor;
printf("Digite o tamanho do vetor: ");
scanf("%d", &tamanho_vetor);
vetor = (int *) malloc(tamanho_vetor * sizeof(int));
for(int x = 0; x < tamanho_vetor; x++){
printf("Digite o valor na posição %d: ", x);
scanf("%d", &vetor[x]);
}
printf("Digite o valor procurado: ");
scanf("%d", &valor_procurado);
foi_encontrado = buscaSimples(vetor, valor_procurado, tamanho_vetor);
free(vetor);
}

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

bool buscaSimples(int * vetor, int valor_procurado, int tamanho_vetor){

bool valor_foi_encontrado = false;

for(int x = 0; x < tamanho_vetor; x++){

if(vetor[x] == valor_procurado){

valor_foi_encontrado = true;

printf("Valor [%d] encontrado na posição %d", valor_procurado, x);
```

```
}

}

if(valor_foi_encontrado == false)

printf("Valor não encontrado!");

return valor_foi_encontrado;

}

void main(){

bool foi_encontrado;

int tamanho_vetor, valor_procurado;

int * vetor;

printf("Digite o tamanho do vetor: ");

scanf("%d", &tamanho_vetor);

vetor = (int *) malloc(tamanho_vetor * sizeof(int));

for(int x = 0; x < tamanho_vetor; x++){

printf("Digite o valor na posição %d: ", x);

scanf("%d", &vetor[x]);

}

printf("Digite o valor procurado: ");

scanf("%d", &valor_procurado);

foi_encontrado = buscaSimples(vetor, valor_procurado, tamanho_vetor);

free(vetor);

}
```

Busca binaria

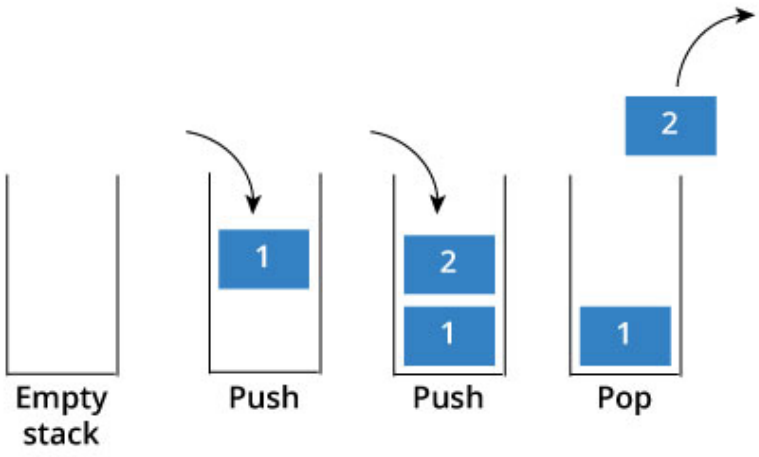
- Como a busca simples, a busca binaria visa encontrar um determinado valor na lista, contudo sendo esta lista obrigatoriamente ordenada (de modo crescente). No caso, tal metodo de busca segue o principio de "dividir para conquistar", no caso, o algoritmo começa a buscar pelo meio da lista ("pivô"), e vai "fatiando" entre os centros da lista, até encontrar o número.
- Para facilitar o entendimento, imagine inicialmente uma lista, e se questione, como poderiamos saber a posição do meio (inicialmente)? Bem, traçando dois limites, um na esquerda (que começa com zero) e um na direita (que corresponde ao tamanho do vetor menos um, para pegar a posição do ultimo elemento), e depois somando ambos os limites e dividindo por dois. Certo, temos a posição do meio, contudo, como poderiamos fazer para continuar fatiando ao meio a lista? Bem, caso o valor procurado fosse maior poderiamos tornar a esquerda igual o valor do meio mais um, e caso o valor fosse menor poderiamos tornar a direita igual ao meio menos um. Sendo o final do codigo, quando o limite da esquerda for maior que o limite da direita.
- Este algoritmo possui como velocidade O(log n).
- Codigo exemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

bool buscaBinaria(int * vetor_inteiros, int tamanho_vetor, int valor_procurado){
bool foi_encontrado = false;
int esquerda = 0, direita = tamanho_vetor - 1, pivo;
while(esquerda <= direita){
pivo = (esquerda + direita)/2;
```

```
if(vetor_inteiros[pivo] == valor_procurado){
printf("Valor[%d] encontrado na posição: %d", valor_procurado, pivo);
foi_encontrado = true;
return foi_encontrado;
}
if(vetor_inteiros[pivo] < valor_procurado){
esquerda = pivo + 1;
}
else{
direita = pivo - 1; //Interessante notar que para fazer as aleterações tanto a esquerda, como a direita, sempre se baseiam no valor do
//Digo isto pois errei, inicialmente, colocando aqui o valor da esquerda - 1.
//O que, caso o algoritmo iniciasse com o pivo sendo menor que o valor solicitado, ele iria pegar o valor da esquerda (0) - 1, ou seja
//Isso acaba por resultar em uma situação onde, no começo do algoritmo, o valor da esquerda torna-se maior que o da direita.
//Assim, terminando a busca.
}
}
if(foi_encontrado == false)
printf("Valor %d não foi encontrado!", valor_procurado);
return foi_encontrado;
}

void main(){
bool foi_encontrado;
int * vetor_inteiros;
int tamanho_vetor, valor_procurado, posicao_valor;
printf("Digite o tamanho do vetor: ");
scanf("%d", &tamanho_vetor);
vetor_inteiros = (int *) malloc(tamanho_vetor * sizeof(int));
for(int x = 0; x < tamanho_vetor; x++){
printf("Digite o valor na posição[%d]: ", x);
scanf("%d", &vetor_inteiros[x]);
}
printf("Digite o valor procurado: ");
scanf("%d", &valor_procurado);
foi_encontrado = buscaBinaria(vetor_inteiros, tamanho_vetor, valor_procurado);
}
```

Pilha

- Pense literalmente em uma pilha, onde o ultimo elemento a entrar sera o primeiro a sair.



- Codigo exemplo:

```
#include <stdlib.h>
#include <stdio.h>

void exibirPilha(int * pilha, int tamanho){
for(int x = tamanho - 1; x >= 0; x--){
printf("%d\n", pilha[x]);
}
}
```

```
void pushPilha(int ** pilha, int * topo, int tamanho, int valor_a_ser_inserido){
if(*topo < tamanho - 1){
*topo += 1;  //Também não consegui descobrir exatamente o porquê de *topo++ neste caso retornar 10
(*pilha)[*topo] = valor_a_ser_inserido; //Leia-se: Conteudo apontado por ponteiro de ponteiro (ou seja, o nosso ponteiro original)
//Esse é um comportamento curioso em C, por algum motivo, para alterar o conteudo de um array por meio de uma função, é necessario
}                                           //Não consegui descobrir exatamente o porquê disto acontecer.
else{
puts("Pilha cheia!");
}
}

void popPilha(int ** pilha, int * topo){
if((**pilha)[*topo] == -1){
printf("A pilha já esta vazia!");
}
else{
printf("Valor removido: %d\n", (** pilha)[* *topo]);
(**pilha)[*topo] = 0;
*topo -= 1;
}
}

void main(){
int * pilha;
int topo = -1, tamanho, valor_a_ser_inserido;
printf("Insira o tamanho da pilha: ");
scanf("%d", &tamanho);
pilha = (int *) malloc(tamanho * sizeof(int));
for(int x = 0; x < tamanho; x++){
printf("Insira o valor a ser inserido na pilha: ");
scanf("%d", &valor_a_ser_inserido);
pushPilha(&pilha, &topo, tamanho, valor_a_ser_inserido);
}
exibirPilha(pilha, tamanho);
popPilha(&pilha, &topo);
exibirPilha(pilha, tamanho);
pushPilha(&pilha, &topo, tamanho, 16);
exibirPilha(pilha, tamanho);
}
```

Fila

- Segue o principio de "primeiro a entrar, primeiro a sair" (First in, first out, ou FIFO)
- Codigo exemplo:

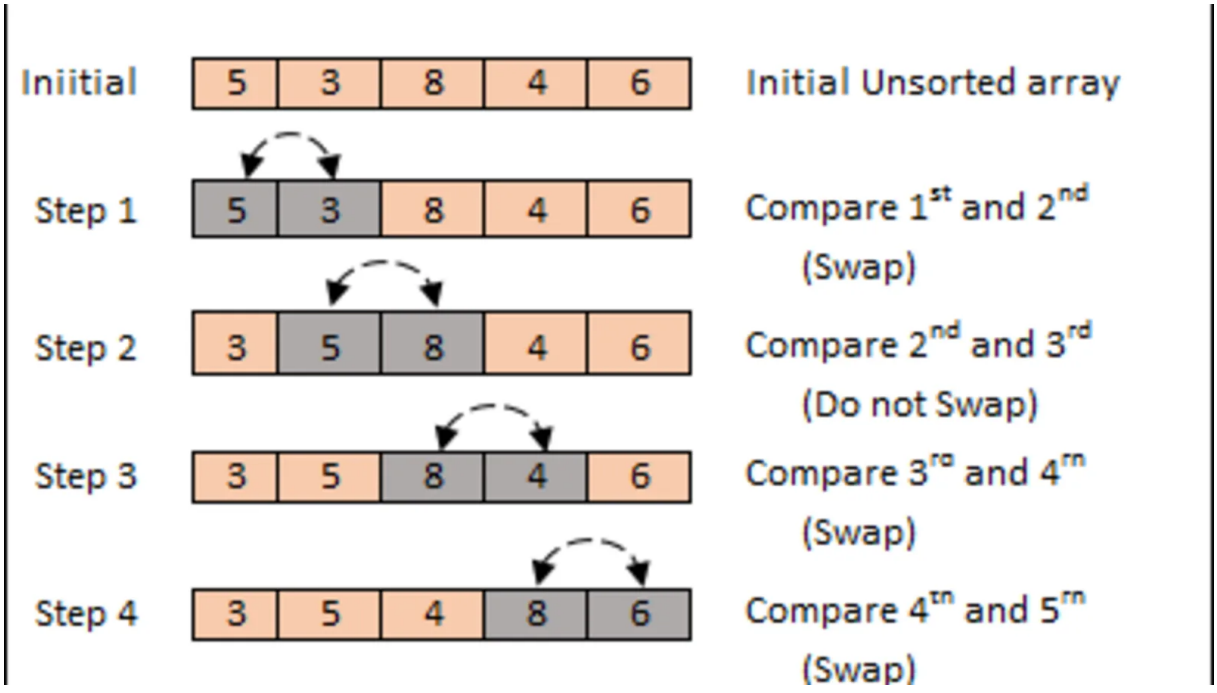
```
#include <stdio.h>
#include <stdlib.h>

void exibir_valores(int * fila, int tamanho){
for(int x = 0; x < tamanho; x++){
if(x < tamanho - 1){
printf("%d - ", fila[x]);
}
else{
printf("%d", fila[x]);
}
}
puts("");
}

void enfileirar(int **fila, int * tras, int valor, int tamanho_fila){
if(*tras  < tamanho_fila - 1){
*tras += 1;
(*fila)[*tras] = valor;
}
else{
printf("Fila cheia!\n");
}
}

void desinfilerar(int ** fila, int * tras, int * frente, int tamanho_fila){
if(*frente > *tras){
printf("Fila vazia!\n");
*tras = -1;  // Essa foi a melhor forma que eu consegui pensar para resetar a posição do elemento a ser inserido, e assim, poder ir
```

```
// O problema é que, em tese, para conseguir resetar a fila é necessario tentar desinfilerar ela enquanto vazia.
*frente = 0;
}
else{
printf("Valor removido: %d\n", (*fila)[*frente]);
(*fila)[*frente] = 0;
*frente += 1;
}
}

void main(){
int tamanho_fila, tras = -1, frente = 0, valor_a_ser_inserido, qtd_valores_a_inserir;
int * fila;
printf("Insira o tamanho da fila: ");
scanf("%d", &tamanho_fila);
fila = (int * ) malloc(tamanho_fila * sizeof(int));
printf("Deseja inserir quantos valores?: ");
scanf("%d", &qtd_valores_a_inserir);
for(int x = 0; x < qtd_valores_a_inserir; x++){
printf("Digite o valor a ser inserido: ");
scanf("%d", &valor_a_ser_inserido);
enfileirar(&fila, &tras, valor_a_ser_inserido, tamanho_fila);
}
exibir_valores(fila, tamanho_fila);
  }
```

Algoritmos de ordenação

Bubble sort

- Resumidamente, ele compara em pares os elementos presentes em um array e vai os trocando, como nessa imagem:



- Melhor caso: Quando o array já esta completamente ordenado.
- Pior caso: Quando o array esta em ordem decrescente.
- Logíca basica: É preciso saber quais as duas posições que estão sendo analizadas (x,y), sendo a segunda posição (y) igual a x + 1.
- Exemplo:

```
#include <stdio.h>
#include <stdlib.h>

int * bubbleSort(int * array, int tamanho){
int aux, * novo_array;
novo_array = (int * ) malloc(tamanho * sizeof(int));
for(int x = 0; x < tamanho; x++){
novo_array[x] = array[x];
}
for(int x = 0; x < tamanho; x++){
for(int y = x + 1; y < tamanho; y++){
if(novo_array[x] > novo_array[y]){
aux = novo_array[x];
novo_array[x] = novo_array[y];
novo_array[y] = aux;
}
}
}
return novo_array;
}

void main(){
int * array, * lista_ordenada, tamanho, valor_a_ser_inserido;
printf("Digite o tamanho do vetor: ");
scanf("%d", &tamanho);
array = (int * ) malloc(tamanho * sizeof(int));
for(int x = 0; x < tamanho; x++){
printf("Digite o valor na posição %d: ", x);
scanf("%d", &valor_a_ser_inserido);
array[x] = valor_a_ser_inserido;
}
lista_ordenada = bubbleSort(array, tamanho);
printf("Lista original: ");
for(int x = 0; x < tamanho; x++){
printf("%d ", array[x]);
}
puts("");
printf("Lista ordenada: ");
for(int x = 0; x < tamanho; x++){
printf("%d ", lista_ordenada[x]);
}
}
```

Insertion sort

- Basicamente, ele analise cada item do vetor, e vê se ele é maior ou menor que os outros, e então, insere/"joga" todos os elementos maiores que ele para frente, como neste GIF:

- Melhor caso: Quando a lista já esta ordenada.
- Pior caso: Quando a lista está em ordem inversa.
- Logíca basica: Basicamente, iremos precisar de uma variavel que representa o elemento analisado, e uma que representa o seu anterior. No caso, o seu anterior varia conforme o elemento analisado, sendo assim, fazemos um laço de repetição com base no elemento analisado (inicializando com 1), e dentro deste laço, temos que o elemento anterior recebe o elemento analizado menos um. Então, enquanto o elemento anterior foi maior ou igual a zero e enquanto o elemento analisado for menor que o elemento anterior, o vetor na posição do elemento anterior mais um recebe o elemento anterior, e anda mais um para trás. Então, apos isso, o vetor na posição do elemento anterior mais um recebe o elemento atual.
- Codigo exemplo:

```
#include <stdio.h>
#include <stdlib.h>

int * insertionSort(int * array, int quantidade_elementos){
int * ordenado = (int *) malloc(quantidade_elementos * sizeof(int)), elemento_analisado, anterior;
for(int x = 0; x < quantidade_elementos; x++){
ordenado[x] = array[x];
}
for(int x = 1; x < quantidade_elementos; x++){
elemento_analisado = ordenado[x];
anterior = x - 1;
while(anterior >= 0 && elemento_analisado < ordenado[anterior]){
ordenado[anterior + 1] = ordenado[anterior];
anterior--;
}
ordenado[anterior + 1] = elemento_analisado;
}
return ordenado;
}

void main(){
int * array, * novo_array, quantidade_elementos, numero_a_ser_inserido;
printf("Digite a quantidade de elementos no vetor: ");
scanf("%d", &quantidade_elementos);
array = (int *) malloc(quantidade_elementos * sizeof(int));
for(int x = 0; x < quantidade_elementos; x++){
printf("Digite o valor a ser colocado na posição %d: ", x);
scanf("%d", &numero_a_ser_inserido);
array[x] = numero_a_ser_inserido;
}
novo_array = insertionSort(array, quantidade_elementos);
printf("Array original: ");
for(int x = 0; x < quantidade_elementos; x++){
printf("%d ", array[x]);
}
puts("");
printf("Array ordenado: ");
for(int x = 0; x < quantidade_elementos; x++){
printf("%d ", novo_array[x]);
}
}
```

Selection sort

- Basicamente, ele percorre cada elemento da lista, e, percorre para cada elemento todos os outros, e assim, verifica se há algum elemento percorrido menor que o elemento da lista, e caso haja, troca os dois de posição.

- Codigo exemplo:

```
#include <stdio.h>
#include <stdlib.h>

int * selectionSort(int * array, int tamanho){
int * ordenado = (int *) malloc(tamanho * sizeof(int)), posicaoMenorValor, aux;
for(int x = 0; x < tamanho; x++)
ordenado[x] = array[x];
for(int x = 0; x < tamanho; x++){
posicaoMenorValor = x;
for(int y = x + 1; y < tamanho; y++){
if(ordenado[y] < ordenado[x]){
posicaoMenorValor = y;
}
}
if(posicaoMenorValor != x){
aux = ordenado[x];
ordenado[x] = ordenado[posicaoMenorValor];
ordenado[posicaoMenorValor] = aux;
}
}
return ordenado;
}

void main(){
int * array, * ordenado, tamanho, valor_a_ser_inserido;
printf("Digite o tamanho do array: ");
scanf("%d", &tamanho);
array = (int *) malloc(tamanho * sizeof(int));
for(int x = 0; x < tamanho; x++){
printf("Digite o valor a ser inserido na posição %d: ", x);
scanf("%d", &valor_a_ser_inserido);
array[x] = valor_a_ser_inserido;
}
ordenado = selectionSort(array, tamanho);
printf("Vetor original: ");
for(int x = 0; x < tamanho; x++){
printf("%d ", array[x]);
}
puts("");
printf("Vetor ordenado: ");
for(int x = 0; x < tamanho; x++){
printf("%d ", ordenado[x]);
}
}
```

Operações e buscas com listas encadeadas

- Conceito: É um método para organizar dados onde tais dados (chamados de nós) são alocados em diferentes regiões da memoria HEAP ("memoria dispersa/geral do computador"), no caso, temos o(s) dado(s) (podendo ser um inteiro, string [char *], char, float, bool), e um ponteiro que aponta pra o próximo nó (mesmo que em linguagens mais "high-level" o detalhe do ponteiro seja escondido por meio da alocação do nó por múltiplas instancias de uma classe).
- A vantagem de uma lista encadeada/ligada consiste no fato de a remoção de um elemento da lista não necessitar na alteração dos outros elementos na lista (diferente de um array que, devido ao fato dos elementos dele estarem em sequencia na memoria, necessita mover todos os elementos de lugar).
- Para alterar os elementos de uma lista ligada basta alterar os ponteiros dos nós.
- Para navegar em uma lista ligada, deve-se criar uma auxiliar dentro de uma função que ira recebendo o proximo nó, até chegar no nó desejado.
- Codigo exemplo

```
#include <stdio.h>
#include <stdlib.h>

struct ListaLigadaNo{
    void * data;
    int tipo_data; /* 1 - Int | 2 - Float */
    struct ListaLigada * proximo_no;
}typedef LISTA_LIGADA_NO;

void inicializarNo(LISTA_LIGADA_NO * no){
    no->data = NULL;
    no->proximo_no = NULL;
    no->tipo_data = NULL;
}

void inicializarLista(LISTA_LIGADA_NO * cabeca){
    int tipo_data;
    inicializarNo(cabeca);
    printf("Digite se quer inteiro(1) ou float(2): ");
    scanf("%d", &tipo_data);
    switch(tipo_data){
        case 1:
            cabeca->data = (int *) malloc(sizeof(int));
            printf("Digite o inteiro: ");
            scanf("%d", cabeca->data);
            cabeca->tipo_data = 1;
            break;
        case 2:
            cabeca->data = (float *) malloc(sizeof(float));
            printf("Digite o float: ");
            scanf("%f", cabeca->data);
            cabeca->tipo_data = 2;
            break;
        default:
            exit(1);
    }
}

void imprimeLista(LISTA_LIGADA_NO * cabeca){
    LISTA_LIGADA_NO * aux = cabeca;
    puts("-----LISTA-----");
    while(aux != NULL){
        if(aux->tipo_data == 1){
            printf("%d\n", *((int *) aux->data));
        }
        else{
            printf("%.2f\n", *((float *) aux->data));
        }
        aux = aux->proximo_no;
    }
    puts("-----FIM-----");
}

LISTA_LIGADA_NO * adicionarNo(LISTA_LIGADA_NO * cabeca, int posicao, int tipo_data){
    LISTA_LIGADA_NO * aux = cabeca, * novo_no = (LISTA_LIGADA_NO *) malloc(sizeof(LISTA_LIGADA_NO));
    int contador = 1;
    inicializarNo(novo_no);
```

```
    for(contador; contador < posicao; contador++){
        if(aux->proximo_no != NULL){
            aux = aux->proximo_no;
        }
        else{
            printf("Saindo do laço...\n");
            break;
        }
    }
    switch(tipo_data){
        case 1:
            novo_no->tipo_data = 1;
            novo_no->data = (int *) malloc(sizeof(int));
            printf("Digite o inteiro: ");
            scanf("%d", ((int *) novo_no->data));
            break;
        case 2:
            novo_no->tipo_data = 2;
            novo_no->data = (float *) malloc(sizeof(float));
            printf("Digite o float: ");
            scanf("%f", ((float *) novo_no->data));
            break;
        default:
            exit(1);
    }
    if(posicao == 0){
        printf("Colocando o no no começo da lista...\n");
        novo_no->proximo_no = cabeca;
        cabeca = novo_no;
        return cabeca;
    }
    else if(contador < posicao){
        printf("Colocando o no no final da lista...\n");
        aux->proximo_no = novo_no;
        return cabeca;
    }
    else{
        printf("Colocando o no na posicao desejada(%d)...\n", posicao);
        novo_no->proximo_no = aux->proximo_no;
        aux->proximo_no = novo_no;
        return cabeca;
    }
}

LISTA_LIGADA_NO * removerNo(LISTA_LIGADA_NO * cabeca, int posicao){
    LISTA_LIGADA_NO * aux = cabeca, * aux2;
    int contador = 1;
    for(contador; contador < posicao; contador++){
        if(aux->proximo_no != NULL){
            aux = aux->proximo_no;
        }
        else{
            printf("Saindo do laço...\n");
            break;
        }
    }
    if(contador == 1 && aux->proximo_no == NULL){
        free(cabeca);
        cabeca = (LISTA_LIGADA_NO *) malloc(sizeof(LISTA_LIGADA_NO));
        inicializarNo(cabeca);
    }
    else if(posicao == 0){
        cabeca = cabeca->proximo_no;
    }
    else if(contador <= posicao){
        aux2 = cabeca;
        while(aux2->proximo_no != aux){
            aux2 = aux2->proximo_no;
        }
        aux2->proximo_no = NULL;
    }
    else{
        aux2 = aux->proximo_no;
        aux2 = aux2->proximo_no;
        aux->proximo_no = aux2;
    }
    return cabeca;
}

void main(){
    LISTA_LIGADA_NO * lista_ligada = (LISTA_LIGADA_NO *) malloc(sizeof(LISTA_LIGADA_NO));
```

```
    int opcao = 0, posicao = 0;
    inicializarLista(lista_ligada);
    while(1){
        puts("----------");
        printf("1 - Inserir inteiro\n2 - Inserir float\n3 - Remover no\n-1 - Sair\n: ");
        scanf("%d", &opcao);
        switch(opcao){
            case 1:
                printf("Insira a posição onde deseja inserir o no: ");
                scanf("%d", &posicao);
                lista_ligada = adicionarNo(lista_ligada, posicao, 1);
                break;
            case 2:
                printf("Insira a posição onde deseja inserir o no: ");
                scanf("%d", &posicao);
                lista_ligada = adicionarNo(lista_ligada, posicao, 2);
                break;
            case 3:
                if(lista_ligada->proximo_no != NULL){
                    printf("Insira a posição onde deseja remover o no: ");
                    scanf("%d", &posicao);
                    lista_ligada = removerNo(lista_ligada, posicao);
                }
                else{
                    printf("Não foi possível rmeover o primeiro no\n");
                }
                break;
            case -1:
                exit(0);
                break;
        }
        imprimeLista(lista_ligada);
    }
}
```

Útil:

- https://www.hackerrank.com/
- https://leetcode.com/
- https://cses.fi/book/book.pdf → Competitive programming handbook
- https://www.youtube.com/watch?v=pg-HEGBpCQk → How To Convert Hexadecimal to Decimal
- https://www.youtube.com/watch?v=EPWmcbBdhzE → Aula Virtual 04 - USP - ICC1: Matrizes, Funções, Stack e Heap
- https://docs.google.com/spreadsheets/d/1A2PaQKcdwO_lwxz9bAnxXnIQayCouZP6d-ENrBz_NXc → Leetcode 75 Questions (NeetCode on yt) spreadsheet
- https://leetcode.com/list/xi4ci4ig/