```
/*
Discente:
Sigilene Irene António Sendela
*/
#include <stdio.h>
#include <stdlib.h>
// Definindo a estrutura do nó da árvore
struct No {
    int valor;
    struct No* esquerda;
    struct No* direita;
};
// Função para criar um novo nó
struct No* criarNo(int valor) {
    // Aloca memória para um novo nó
    struct No* novoNo = (struct No*)malloc(sizeof(struct No));
    // Define o valor do nó como o valor fornecido
    novoNo->valor = valor;
    // Inicializa os ponteiros esquerdo e direito como NULL
    novoNo->esquerda = NULL;
    novoNo->direita = NULL;
    // Retorna o novo nó criado
    return novoNo;
```

```
}

// Função para inserir um nó na árvore
struct No* inserirNo(struct No* raiz, int valor) {
    // Se a raiz for NULL, cria um novo nó com o valor fornecido
    if (raiz == NULL) {
        return criarNo(valor);
    }
    // Caso contrário, insere o nó na subárvore esquerda ou direita
    if (valor < raiz->valor) {
        raiz->esquerda = inserirNo(raiz->esquerda, valor);
    } else {
        raiz->direita = inserirNo(raiz->direita, valor);
    }
    // Retorna a raiz atualizada da árvore
    return raiz;
}

// Função para buscar em profundidade (pré-ordem)
void buscarProfundidade(struct No* raiz) {
    // Se a raiz não for NULL
    if (raiz != NULL) {
        // Imprime o valor do nó atual
        printf("%d ", raiz->valor);
        // Chama a busca em profundidade para a subárvore esquerda
```

```
        buscarProfundidade(raiz->esquerda);
        // Chama a busca em profundidade para a subárvore direita
        buscarProfundidade(raiz->direita);
    }
}
// Função para buscar em largura (ordem de nível)
void buscarLargura(struct No* raiz) {
    // Se a raiz for NULL, não há nada para buscar
    if (raiz == NULL) {
        return;
    }
    // Cria uma fila para armazenar os nós a serem explorados
    struct No* fila[100];
    int frente = 0;
    int tras = 0;
    // Enfileira a raiz para começar a busca
    fila[tras++] = raiz;
    // Enquanto há elementos na fila
    while (frente < tras) {
        // Remove o primeiro elemento da fila (o nó atual)
        struct No* atual = fila[frente++];
        // Imprime o valor do nó atual
        printf("%d ", atual->valor);
```

```
        // Enfileira os filhos (se existirem)
        if (atual->esquerda != NULL) {
            fila[tras++] = atual->esquerda;
        }
        if (atual->direita != NULL) {
            fila[tras++] = atual->direita;
        }
    }
}
struct Pilha {
    struct No* elementos[100];
    int topo;
};
void inicializarPilha(struct Pilha* pilha) {
    // Inicializa o topo da pilha como -1
    pilha->topo = -1;
}
void empilhar(struct Pilha* pilha, struct No* elemento) {
    // Incrementa o topo e armazena o elemento na pilha
    pilha->topo++;
    pilha->elementos[pilha->topo] = elemento;
}
struct No* desempilhar(struct Pilha* pilha) {
```

```
    // Obtém o elemento do topo, decrementa o topo e retorna o elemento
    struct No* elemento = pilha->elementos[pilha->topo];
    pilha->topo--;
    return elemento;
}
int buscarEArmazenarCaminho(struct No* raiz, int alvo, struct Pilha* pilha) {
    // Se a raiz for NULL, não encontrou o alvo
    if (raiz == NULL) {
        return 0;
    }
    // Se o valor da raiz for o alvo, empilha o nó e retorna 1
    if (raiz->valor == alvo) {
        empilhar(pilha, raiz);
        return 1;
    }
    // Tenta buscar o alvo na subárvore esquerda ou direita
    // Se encontrado, empilha o nó e retorna 1
    if (buscarEArmazenarCaminho(raiz->esquerda, alvo, pilha) ||
        buscarEArmazenarCaminho(raiz->direita, alvo, pilha)) {
        empilhar(pilha, raiz);
        return 1;
    }
    // Alvo não encontrado nesta subárvore
```

```
    return 0;
}
// Função para buscar por profundidade um nó com valor x e empilhar elementos
// dos caminhos nos respectivos nós
int buscarProfundidadeX(struct No* raiz, int alvo, struct Pilha* pilha) {
    if (raiz == NULL) {
        return 0;
    }
    empilhar(pilha, raiz);
    if (raiz->valor == alvo) {
        printf("Caminho para encontrar %d: ", alvo);
        int i;
        for (i = 0; i <= pilha->topo; i++) {
            printf("%d ", pilha->elementos[i]->valor);
        }
        printf("\n");
        return 1; // Retorna 1 quando o nó alvo é encontrado
    }

    int encontrado = buscarProfundidadeX(raiz->esquerda, alvo, pilha) ||
                     buscarProfundidadeX(raiz->direita, alvo, pilha);

    desempilhar(pilha);
```

```
    return encontrado;
}

// Função para mostrar de forma ordenada os caminhos para encontrar
// os diversos nós x na árvore
void mostrarCaminhosOrdenados(struct No* raiz, int* alvos, int numAlvos) {
    struct Pilha caminhoPilha;
    inicializarPilha(&caminhoPilha);
    int i;
    for (i = 0; i < numAlvos; i++) {
        int encontrados = 0;
        printf("Caminhos ordenados para encontrar %d:\n", alvos[i]);

        // Busca e mostra pelo menos dois caminhos para cada nó alvo
        while (encontrados < 2) {
            if (buscarProfundidadeX(raiz, alvos[i], &caminhoPilha)) {
                encontrados++;
            } else {
                break; // Se não encontrar mais caminhos, para
            }
        }
    }
}
```

```
int main() {
    struct No* raiz = NULL;

    // Inserindo 20 nós na árvore
    int valores[] = {10, 5, 15, 3, 8, 12, 18, 2, 4, 7, 9, 11, 14, 17, 20, 1, 6, 13, 16, 19};
    int i=0;
    for (i; i < 20; i++) {
        raiz = inserirNo(raiz, valores[i]);
    }

    // Busca em profundidade
    printf("Busca por profundidade: ");
    buscarProfundidade(raiz);
    printf("\n");

    // Busca em largura
    printf("Busca em largura: ");
    buscarLargura(raiz);
    printf("\n");

    // Inicializa a pilha para armazenar o caminho
```

```
    struct Pilha caminhoPilha;
    inicializarPilha(&caminhoPilha);



    // Nós a serem buscados
    int alvos[] = {11};
    int numAlvos = sizeof(alvos) / sizeof(alvos[0]);


    // Mostra os caminhos ordenados para os nós alvos
    mostrarCaminhosOrdenados(raiz, alvos, numAlvos);


    return 0;
}
```

```
Busca por profundidade: 10 5 3 2 1 4 8 7 6 9 15 12 11 14 13 18 17 16 20 19
Busca em largura: 10 5 15 3 8 12 18 2 4 7 9 11 14 17 20 1 6 13 16 19
Caminhos ordenados para encontrar 11:
Caminho para encontrar 11: 10 15 12 11
Caminho para encontrar 11: 10 10 15 12 11
```