Universidade Federal de Viçosa
Campus Florestal
CCF 330 - PROJETO DE ANÁLISE DE ALGORITMO

TRABALHO PRÁTICO 1
Exploração Espacial com Backtracking: Um Estudo de Busca em Mapas

Fabiano Lara Leroy - 4227
Gabriel Magalhães - 5560
Heron Fillipe Silveira Santos - 4211

Sumário

Resumo

Este trabalho apresenta o desenvolvimento de um algoritmo baseado na técnica de backtracking para solucionar o problema proposto no contexto do “Expresso Interestelar”. O objetivo é determinar um percurso viável em um mapa representando uma rodovia estelar, permitindo que a tripulação alcance seu destino antes que a durabilidade da nave se esgote. O algoritmo é capaz de ler dados de entrada a partir de um arquivo texto, processar as informações do mapa e calcular, recursivamente, um caminho possível entre o ponto inicial e o destino final, levando em conta a coleta de peças que restauram a durabilidade do veículo. Foram implementadas estruturas dinâmicas para armazenar o mapa e contabilizar as chamadas recursivas e o nível máximo de profundidade atingido. Os resultados obtidos comprovam a eficiência da abordagem e a correta aplicação do paradigma de backtracking.

Introdução

O presente trabalho foi desenvolvido no âmbito da disciplina Projeto e Análise de Algoritmos (PAA), ministrada no curso de Ciência da Computação da Universidade Federal de Viçosa – Campus Florestal. O desafio consiste em criar um algoritmo recursivo capaz de encontrar um caminho possível dentro de um mapa simbólico, representando o percurso do Expresso Interestelar até seu destino. O problema envolve conceitos fundamentais de recursão, busca em profundidade e retrocesso de estado (backtracking), exigindo que o programa explore caminhos possíveis, retorne quando necessário e contabilize estatísticas de execução. Além disso, a implementação contempla o uso de alocação dinâmica de memória, uma vez que o tamanho das estruturas é conhecido apenas após a leitura dos arquivos de entrada, e também a inclusão de um modo de análise para avaliar o desempenho do algoritmo.

Organização

A estrutura do projeto foi organizada de forma modular, visando clareza, reutilização de código e facilidade de manutenção. Na pasta principal do trabalho encontra-se o arquivo main.c, responsável pelo controle geral da execução e integração entre os módulos. O diretório src abriga três subpastas que separam as principais funcionalidades do programa: map, que contém as funções relacionadas à representação e manipulação do mapa (map.c e map.h); readFile, responsável pela leitura e interpretação do arquivo de entrada (readFile.c e readFile.h); e backtracking, que implementa o algoritmo de busca e controle recursivo (pathfinder.c e pathfinder.h). A pasta Files\In armazena os arquivos de entrada utilizados nos testes, como entrada.txt, contendo os dados do mapa e dos parâmetros do problema. Além disso, há arquivos auxiliares de configuração, como .gitignore e PAA-TP1.code-workspace, e o executável gerado após a compilação (pathfinder.exe). Essa organização reflete uma divisão lógica das responsabilidades do código, tornando o

desenvolvimento mais estruturado e permitindo a compreensão isolada de cada componente do sistema.

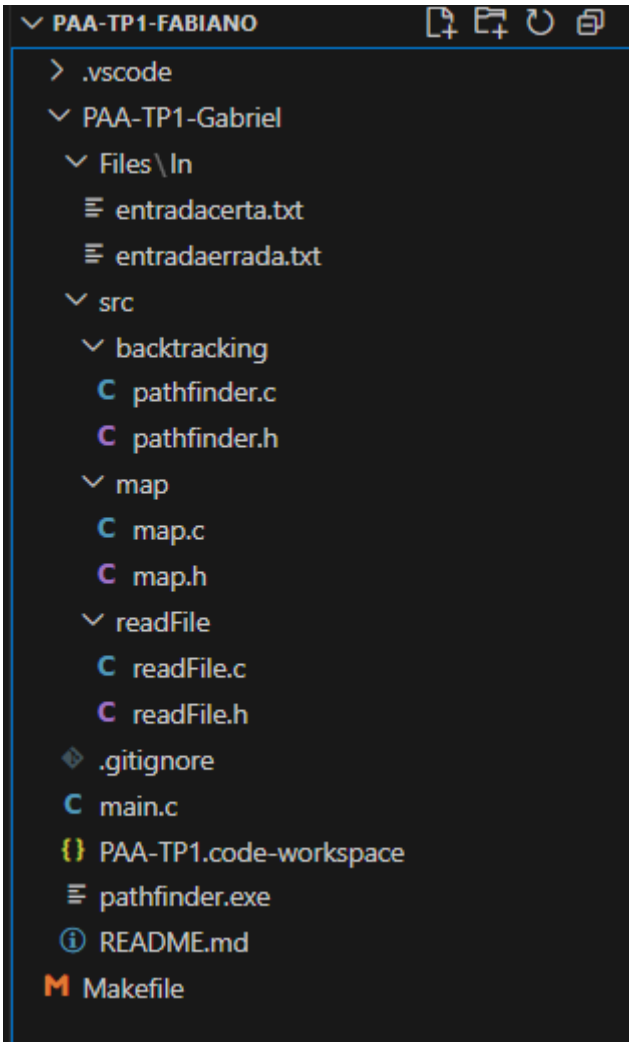


Imagem 1 - Organização do Projeto

Desenvolvimento

Makefile

Makefile

```
CC = gcc
SRC_DIR = src
MAP_DIR = $(SRC_DIR)/map
READ_DIR = $(SRC_DIR)/readFile
PATH_DIR = $(SRC_DIR)/backtracking
SRC = main.c \
      $(MAP_DIR)/map.c \
      $(READ_DIR)/readFile.c \
      $(PATH_DIR)/pathfinder.c
INCLUDES = -I$(MAP_DIR) -I$(READ_DIR) -I$(PATH_DIR)
CFLAGS = -Wall -Wextra -std=c11 $(INCLUDES)
TARGET = pathfinder

all: $(TARGET)

$(TARGET): $(SRC)
    @echo "Compilando Expresso Interestelar..."
    $(CC) $(CFLAGS) $(SRC) -o $(TARGET)
    @echo "Compilação concluída com sucesso!"

analise:
    @echo "Compilando com MODO DE ANÁLISE ativado..."
    $(CC) $(CFLAGS) -DANALISE $(SRC) -o $(TARGET)
    @echo "Compilação concluída (modo análise)."

run:
    @echo "Executando Expresso Interestelar..."
    ./$(TARGET)

run-analise:
    @echo "Executando Expresso Interestelar (modo análise)..."
    ./$(TARGET)

clean:
    @echo "Removendo arquivos compilados..."
    rm -f $(TARGET)
    @echo "Limpeza concluída!"
```

Imagem 2 - Makefile

O Makefile é um arquivo de configuração que automatiza o processo de compilação do projeto "Expresso Interestelar".

- **Compilação Padrão**: A regra principal (all) usa o compilador gcc para compilar e vincular todos os módulos (main.c, map.c, readfile.c, pathfinder.c), gerando um único executável chamado pathfinder. Ele gerencia automaticamente os caminhos de inclusão (INCLUDES) e as flags de compilação (-Wall, -Wextra).
- **Modo de Análise**: Ele implementa a regra analyse, que é crucial para os requisitos do trabalho. Ao executar make analyse, o programa é compilado com a flag adicional -DANALISE. Esta flag ativa o código condicional (#ifdef ANALISE) para contabilizar chamadas recursivas e profundidade.

- **Utilitários**: Também fornece regras de conveniência como run (para executar o programa), run-analyse (para executar a versão de análise) e clean (para remover o executável compilado).

Módulo Mapa

Este módulo descreve os arquivos responsáveis por gerenciar os dados do mapa, as regras de movimento e o gerenciamento de memória.

map.h

Este arquivo define a interface pública do módulo do mapa. Ele declara as variáveis globais e os protótipos das funções que estarão disponíveis para outros arquivos do projeto.

```c
#ifndef MAP_H
#define MAP_H
#include <stdbool.h>

extern char **mapa;
extern bool **visitado;
extern bool **peca_coletada;
extern int R, C;
extern int D_init, D_dec, A_add;

bool dentro(int r, int c);
bool permite_direcao(char ch, int dir);
bool ha_conexao(int r, int c, int nr, int nc);
void libera_estruturas(void);
void debug_mapa(int sr, int sc);
void debug_conexoes(int sr, int sc);

#endif

```

Imagem 3 - map.h

Variáveis Globais (extern):

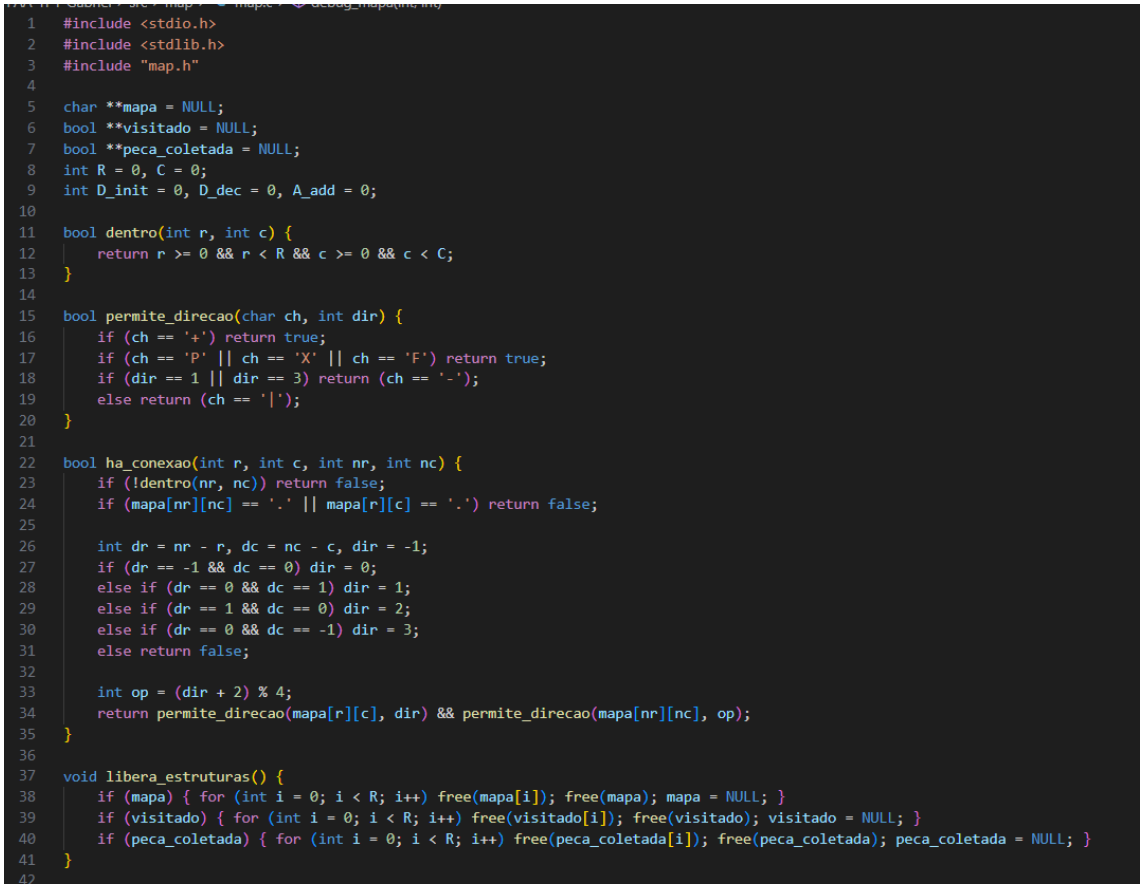
- **char **mapa**: Matriz que armazena o mapa (paredes, caminhos, peças, etc.).
- **bool **visitado**: Matriz para marcar posições visitadas.
- **bool **peca_coletada**: Matriz para rastrear peças coletadas.
- **int R, C**: Dimensões (altura e largura) do mapa.
- **int D_init, D_dec, A_add:** Parâmetros do jogo (Durabilidade inicial, decremento e bônus de peça).

Protótipos de Funções:

- **bool dentro(int r, int c)**: Verifica se a posição está dentro dos limites do mapa.
- **bool permite_direcao(char ch, int dir)**: Checa se um tipo de célula ('-', '|', '+') permite passagem em uma direção.
- **bool ha_conexao(int r, int c, int nr, int nc)**: Determina se duas células vizinhas estão conectadas de forma válida.
- **void libera_estruturas(void)**: Libera toda a memória alocada para as matrizes.
- **void debug_mapa(int sr, int sc)**: Exibe o mapa e os parâmetros de jogo para depuração.
- **void debug_conexoes(int sr, int sc)**: Mostra as conexões válidas a partir de uma célula para depuração.

map.c

Este arquivo contém a implementação (a lógica) das funções declaradas em map.h.



```c
#include <stdio.h>
#include <stdlib.h>
#include "map.h"

char **mapa = NULL;
bool **visitado = NULL;
bool **peca_coletada = NULL;
int R = 0, C = 0;
int D_init = 0, D_dec = 0, A_add = 0;

bool dentro(int r, int c) {
    return r >= 0 && r < R && c >= 0 && c < C;
}

bool permite_direcao(char ch, int dir) {
    if (ch == '+') return true;
    if (ch == 'P' || ch == 'X' || ch == 'F') return true;
    if (dir == 1 || dir == 3) return (ch == '-');
    else return (ch == '|');
}

bool ha_conexao(int r, int c, int nr, int nc) {
    if (!dentro(nr, nc)) return false;
    if (mapa[nr][nc] == '.' || mapa[r][c] == '.') return false;

    int dr = nr - r, dc = nc - c, dir = -1;
    if (dr == -1 && dc == 0) dir = 0;
    else if (dr == 0 && dc == 1) dir = 1;
    else if (dr == 1 && dc == 0) dir = 2;
    else if (dr == 0 && dc == -1) dir = 3;
    else return false;

    int op = (dir + 2) % 4;
    return permite_direcao(mapa[r][c], dir) && permite_direcao(mapa[nr][nc], op);
}

void libera_estruturas() {
    if (mapa) { for (int i = 0; i < R; i++) free(mapa[i]); free(mapa); mapa = NULL; }
    if (visitado) { for (int i = 0; i < R; i++) free(visitado[i]); free(visitado); visitado = NULL; }
    if (peca_coletada) { for (int i = 0; i < R; i++) free(peca_coletada[i]); free(peca_coletada); peca_coletada = NULL; }
}
```

Imagem 4 - Map.c

```c
void debug_mapa(int sr, int sc) {
    printf("=== DEBUG MAPA ===\n");
    printf("Dimensoes: %d x %d\n", R, C);
    printf("Start: (%d,%d) - '%c'\n", sr, sc, mapa[sr][sc]);
    printf("Destino F: ");
    for (int i = 0; i < R; i++)
        for (int j = 0; j < C; j++)
            if (mapa[i][j] == 'F') printf("(%d,%d) ", i, j);
    printf("\nD_init: %d, D_dec: %d, A_add: %d\nMapa:\n", D_init, D_dec, A_add);
    for (int i = 0; i < R; i++) printf("%s\n", mapa[i]);
    printf("==================\n\n");
}

void debug_conexoes(int sr, int sc) {
    printf("=== DEBUG CONEXOES from (%d,%d)='%c' ===\n", sr, sc, mapa[sr][sc]);
    const int drs[4] = {-1, 0, 1, 0};
    const int dcs[4] = {0, 1, 0, -1};
    const char *dirs[4] = {"UP", "RIGHT", "DOWN", "LEFT"};
    for (int k = 0; k < 4; k++) {
        int nr = sr + drs[k], nc = sc + dcs[k];
        if (dentro(nr, nc))
            printf("  %s: to (%d,%d)='%c' -> %s\n", dirs[k], nr, nc, mapa[nr][nc],
                   ha_conexao(sr, sc, nr, nc) ? "CONECTADO" : "BLOQUEADO");
        else
            printf("  %s: FORA DO MAPA\n", dirs[k]);
    }
    printf("================================\n\n");
}
```

Imagem 5 - map.c

Funções:

- **libera_estruturas()** Percorre as matrizes mapa, visitado e peca_coletada, liberando a memória de cada linha e, em seguida, a memória da própria matriz, evitando vazamentos de memória.
- **dentro(int r, int c)** Retorna true apenas se r (linha) estiver entre 0 e R-1 e c (coluna) estiver entre 0 e C-1.
- **permite_direcao(char ch, int dir)** Implementa as regras de movimento dos ladrilhos. Verifica se o caractere ch (ex: '-', '|', '+', 'X', 'F', 'P') permite movimento na direção dir (0=UP, 1=RIGHT, 2=DOWN, 3=LEFT).
- **ha_conexao(int r, int c, int nr, int nc)** Esta é a função mais importante para a lógica de movimento. Ela verifica se um movimento da célula (r, c) para a vizinha (nr, nc) é válido, checando múltiplos critérios:
    - Se a célula de destino (nr, nc) está dentro do mapa (usando dentro).
    - Se nenhuma das células é uma parede (.).
    - Usando permite_direcao, ela realiza uma verificação dupla: se a célula de origem (r, c) permite sair na direção do movimento E se a célula de destino (nr, nc) permite entrar pela direção oposta.

Funções de Depuração

- **debug_mapa** imprime o mapa formatado e os valores de D, D' e A.
- **debug_conexoes** testa as 4 direções a partir de um ponto (sr, sc) e informa, usando ha_conexao, quais estão "CONECTADO" ou "BLOQUEADO".d

Módulo de Leitura

Este módulo descreve os arquivos responsáveis por ler um arquivo de mapa, alocar a memória necessária e inicializar os parâmetros do problema.

readfile.h

Este arquivo define a interface pública do módulo de leitura. Sua única responsabilidade é declarar a função principal de leitura para que outros módulos (como o main.c) possam usá-la.

```c
#ifndef READFILE_H
#define READFILE_H
#include <stdbool.h>

bool ler_arquivo(const char *fname, int *sr, int *sc);

#endif

```

Imagem 6 - readfile.h

bool ler_arquivo(const char *fname, int *sr, int *sc);: Declara a função que lê o arquivo. Ela recebe o nome do arquivo (fname) e dois ponteiros (sr, sc) que serão usados para "retornar" as coordenadas da posição inicial ('X').

readfile.c

Este arquivo contém a implementação da função ler_arquivo. Ele é responsável por todo o processo de parsing do arquivo de entrada e pela alocação dinâmica das estruturas de dados globais (definidas em map.h).

PAA-TP1-Gabriel > src > readFile > C readFile.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "../map/map.h"

bool ler_arquivo(const char *fname, int *sr, int *sc) {
    FILE *f = fopen(fname, "r");
    if (!f) { printf("Erro: Nao foi possivel abrir '%s'\n", fname); return false; }

    char line[2048];
    if (!fgets(line, sizeof(line), f) || sscanf(line, "%d %d %d", &D_init, &D_dec, &A_add) != 3)
        return fclose(f), printf("Erro na primeira linha\n"), false;

    if (!fgets(line, sizeof(line), f) || sscanf(line, "%d %d", &R, &C) != 2)
        return fclose(f), printf("Erro nas dimensoes\n"), false;

    mapa = malloc(sizeof(char*) * R);
    visitado = malloc(sizeof(bool*) * R);
    peca_coletada = malloc(sizeof(bool*) * R);
    for (int i = 0; i < R; i++) {
        mapa[i] = malloc(C + 1);
        visitado[i] = calloc(C, sizeof(bool));
        peca_coletada[i] = calloc(C, sizeof(bool));
    }

    int start_r = -1, start_c = -1;
    for (int i = 0; i < R; i++) {
        fgets(line, sizeof(line), f);
        int len = strlen(line);
        while (len > 0 && (line[len-1] == '\n' || line[len-1] == '\r')) line[--len] = '\0';
        for (int j = 0; j < C; j++) {
            mapa[i][j] = (j < len) ? line[j] : '.';
            if (mapa[i][j] == 'X') { start_r = i; start_c = j; }
        }
        mapa[i][C] = '\0';
    }
    fclose(f);
    if (start_r == -1) return printf("Erro: 'X' nao encontrado\n"), false;
    *sr = start_r; *sc = start_c; return true;
}

```

Imagem 7 - readfile.c

Função:

bool ler_arquivo(const char *fname, int *sr, int *sc) Esta função executa as seguintes etapas:

1. Abertura: Tenta abrir o arquivo fname em modo de leitura ("r"). Se falhar, imprime um erro e retorna false.
2. Leitura de Parâmetros: Lê a primeira linha do arquivo e a interpreta para preencher as variáveis globais D_init, D_dec, e A_add. (Ex: "20 5 10").
3. Leitura de Dimensões: Lê a segunda linha para preencher as variáveis globais R (linhas) e C (colunas). (Ex: "7 10").
4. Alocação de Memória: Com R e C definidos, ele aloca dinamicamente a memória para as matrizes globais:
    - mapa = malloc(sizeof(char*) * R);
    - visitado = malloc(sizeof(bool*) * R);
    - peca_coletada = malloc(sizeof(bool*) * R);
    - Em seguida, aloca as colunas para cada linha (ex: mapa[i] = malloc(C + 1);).

5. Leitura do Mapa: Lê o restante do arquivo (as R linhas seguintes) linha por linha.
6. Preenchimento e Busca: Para cada linha lida, ele a copia para mapa[i] e, simultaneamente, procura pelo caractere 'X'.
7. Armazenamento do Início: Ao encontrar o 'X', armazena suas coordenadas (i, j) nas variáveis locais start_r e start_c.
8. Validação e Retorno: Após ler todo o mapa, ele fecha o arquivo. Se o 'X' não for encontrado (start_r == -1), imprime um erro e retorna false. Caso contrário, atribui os valores de início aos ponteiros de saída (*sr = start_r, *sc = start_c) e retorna true.

Módulo de Busca

Este módulo implementa o núcleo do algoritmo de backtracking, responsável por encontrar um caminho válido do início ao fim do mapa.

pathfinder.h

Executa a busca recursiva (backtracking) que tenta levar a nave do ponto X até F, respeitando as regras de durabilidade e coleta de peças.

```
1  #ifndef PATHFINDER_H
2  #define PATHFINDER_H
3  #include <stdbool.h>
4
5  void movimentar(int r, int c, int dur, int pecas, int depth);
6  void imprimir_solucao(void);
7
8  #endif
9
```

Imagem 8 - pathfinder.h

Este arquivo define a interface pública do módulo de busca (pathfinder).

- void movimentar(int r, int c, int dur, int pecas, int depth); Esta é a declaração da principal função recursiva de backtracking. Ela explora o caminho a partir da posição (r, c) com um estado atual de durabilidade (dur), peças coletadas (pecas) e profundidade de recursão (depth).
- void imprimir_solucao(void); Declara a função que será chamada após a conclusão da busca para imprimir o resultado final (o caminho encontrado ou uma mensagem de falha).

pathfinder.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "../map/map.h"

#define REQUIRED_PIECES 4

#ifdef ANALISE
long long rec_calls = 0;
long long max_rec_depth = 0;
#endif

typedef struct {
    int linha, coluna;
    int durabilidade;
    int pecas_restantes;
} Step;

static Step *path_buf = NULL;
static Step *sol_path = NULL;
static int path_len = 0, sol_len = 0;
static bool solution_found = false;

static void push_step(int r, int c, int D, int faltam) {
    if (!path_buf) path_buf = malloc(sizeof(Step) * R * C * 10);
    path_buf[path_len++] = (Step){r + 1, c + 1, D, faltam};
}

static void pop_step(void) { if (path_len > 0) path_len--; }

static void salvar_solucao(void) {
    sol_len = path_len;
    sol_path = realloc(sol_path, sizeof(Step) * sol_len);
    for (int i = 0; i < sol_len; i++) sol_path[i] = path_buf[i];
    solution_found = true;
    printf("\n*** SOLUCAO ENCONTRADA! Comprimento: %d ***\n", sol_len);
}

```

Imagem 9 - pathfinder.c

Esta seção do arquivo define as variáveis estáticas (globais ao arquivo) e as funções auxiliares que gerenciam o estado da busca e o armazenamento do caminho.

- **Variáveis Estáticas e Estruturas**:
    - REQUIRED_PIECES 4: Define o número total de peças necessárias para o reparo completo (conforme especificado no PDF).
    - rec_calls, max_rec_depth: Variáveis (compiladas condicionalmente com #ifdef ANALISE) para rastrear o número de chamadas recursivas e a profundidade máxima atingida, para o "modo de análise".
    - typedef struct Step: Define uma estrutura que armazena o estado de um único passo no caminho: a linha, coluna, durabilidade atual e o número de peças restantes.
    - path_buf, sol_path: Ponteiros do tipo Step que armazenam, respectivamente, o caminho atual sendo explorado e o caminho da solução final encontrada.
    - path_len, sol_len: O comprimento atual do path_buf e sol_path.
    - solution_found: Flag booleana que indica se uma solução já foi encontrada, usada para parar a busca recursiva.
- **void push_step(int r, int c, int d, int faltam)** Função auxiliar que adiciona o estado atual ao path_buf (pilha). Ela aloca/realoca dinamicamente o buffer conforme necessário e incrementa path_len.

- **void pop_step(void)** Função auxiliar que "remove" o último estado do path_buf (pilha), simplesmente decrementando path_len. Este é o passo central do "backtrack".
- **void salvar_solucao(void)** Chamada quando a função movimentar encontra o destino ('F'). Ela copia o caminho do path_buf para o sol_path (alocando a memória exata) e define solution_found = true para sinalizar a todas as outras chamadas recursivas que a busca deve parar.

```
39 void movimentar(int r, int c, int dur, int pecas, int depth) {
40 #ifdef ANALISE
41     rec_calls++;
42     if (depth > max_rec_depth) max_rec_depth = depth;
43 #endif
44     if (solution_found || dur < 0 || visitado[r][c]) return;
45
46     visitado[r][c] = true;
47     int faltam = REQUIRED_PIECES - pecas;
48     push_step(r, c, dur, faltam);
49
50     if (mapa[r][c] == 'F') { salvar_solucao(); pop_step(); visitado[r][c] = false; return; }
51
52     const int drs[4] = {-1, 0, 1, 0};
53     const int dcs[4] = {0, 1, 0, -1};
54
55     for (int k = 0; k < 4 && !solution_found; k++) {
56         int nr = r + drs[k], nc = c + dcs[k];
57         if (!dentro(nr, nc) || visitado[nr][nc] || !ha_conexao(r, c, nr, nc)) continue;
58
59         int newD = dur - D_dec;
60         if (newD < 0) continue;
61
62         bool pegou = false;
63         if (mapa[nr][nc] == 'P' && !peca_coletada[nr][nc]) {
64             newD += A_add;
65             pegou = true;
66             peca_coletada[nr][nc] = true;
67         }
68
69         int newP = pecas + (pegou ? 1 : 0);
70         if (newP > REQUIRED_PIECES) newP = REQUIRED_PIECES;
71
72         movimentar(nr, nc, newD, newP, depth + 1);
73         if (pegou) peca_coletada[nr][nc] = false;
74     }
75
76     pop_step();
77     visitado[r][c] = false;
78 }
79
```

Imagem 10 - pathfinder.c

Esta seção contém a implementação da função movimentar, o coração do algoritmo de backtracking.

- **void movimentar(int r, int c, int dur, int pecas, int depth)** Esta função implementa a lógica recursiva para explorar o mapa.
    1. **Modo Análise**: Se ANALISE estiver definido, incrementa rec_calls e atualiza max_rec_depth.
    2. **Condições de Parada (Poda)**: A função retorna (pára de explorar este galho) se:
        - Uma solução já foi encontrada (solution_found).

- A durabilidade (dur) chegou a 0 ou menos.
- A posição (r, c) já foi visitada (visitado[r][c]) no caminho atual.

3. **Visita**: Marca a célula atual como visitada (visitado[r][c] = true) e a empilha no caminho (push_step).
4. **Objetivo**: Verifica se a posição atual é o destino (mapa[r][c] == 'F'). Se for, salva a solução (salvar_solucao), desfaz a visita (pop_step, visitado[r][c] = false) e retorna.
5. **Exploração (Recursão)**: Itera pelas 4 direções vizinhas (k = 0 a 3).
6. **Validação do Vizinho**: Para cada vizinho (nr, nc), ele verifica se é uma jogada válida: se está dentro do mapa (dentro), se não foi visitado (!visitado) e se há uma conexão válida (ha_conexao). Se não for válido, pula para a próxima direção.
7. **Cálculo de Estado**: Calcula a nova durabilidade (newD). Se todas as peças já foram coletadas, o custo de movimento (D_dec) é ignorado.
8. **Coleta de Peça**: Verifica se o vizinho (nr, nc) é uma peça ('P') e se ela ainda não foi coletada (!peca_coletada[nr][nc]).
9. **Bônus**: Se uma peça for coletada, a durabilidade é aumentada (newD += A_add), o contador de peças é incrementado (newP) e a peça é marcada como coletada (peca_coletada[nr][nc] = true).
10. **Chamada Recursiva**: Chama movimentar para o vizinho (nr, nc) com os novos valores de estado.
11. **Backtracking (Peça)**: Após o retorno da chamada recursiva, se uma peça foi coletada neste passo, ela é "descoletada" (peca_coletada[nr][nc] = false). Isso é crucial para que outros caminhos alternativos possam tentar coletar essa mesma peça.
12. **Backtracking (Posição)**: Após testar todas as 4 direções, a função "desempilha" o passo atual (pop_step) e desmarca a visita (visitado[r][c] = false), permitindo que outros ramos da recursão visitem esta célula.

```c
void imprimir_solucao(void) {
    if (!solution_found) {
        printf("\nApesar da bravura a tripulacao falhou em sua jornada\n");
#ifdef ANALISE
        printf("\n--- MODO ANALISE ---\nChamadas recursivas: %lld\nMaior profundidade: %lld\n",
               rec_calls, max_rec_depth);
#endif
        return;
    }
    printf("\n=== PERCURSO ENCONTRADO ===\n");
    for (int i = 0; i < sol_len; i++)
        printf("Linha: %d, Coluna: %d; D: %d, pecas restantes: %d\n",
               sol_path[i].linha, sol_path[i].coluna, sol_path[i].durabilidade, sol_path[i].pecas_restantes);

    Step last = sol_path[sol_len - 1];
    printf("\n#MENSAGEM FINAL\n\n");
    if (last.pecas_restantes == 0)
        printf("A jornada sera finalizada sem mais desafios\n");
    else
        printf("A tripulacao finalizou sua jornada\n");

#ifdef ANALISE
    printf("\n--- MODO ANALISE ---\nChamadas recursivas: %lld\nMaior profundidade: %lld\n",
           rec_calls, max_rec_depth);
#endif
}
```

Imagem 11 - pathfinder.c

Esta seção implementa a função final, responsável por exibir os resultados da busca.

- **void imprimir_solucao(void)**
    1. **Sem Solução**: Se solution_found for false, imprime a mensagem de falha ("Apesar da bravura...").
    2. **Solução Encontrada**: Se solution_found for true:
        - Imprime o cabeçalho "PERCURSO ENCONTRADO".
        - Itera pelo sol_path (de i = 0 até sol_len - 1), imprimindo cada passo no formato "Linha: ..., Coluna: ...; D: ..., pecas restantes: ...".
        - Mensagem Final: Verifica o último passo da solução. Se pecas_restantes == 0, imprime a mensagem de sucesso total ("A jornada sera finalizada sem mais desafios"). Caso contrário, imprime a mensagem de chegada padrão ("A tripulacao finalizou sua jornada").
    3. **Modo Análise**: Se ANALISE estiver definido, imprime as estatísticas rec_calls e max_rec_depth, independentemente do resultado.

Módulo Principal

Main

O arquivo main.c é o ponto de entrada do programa "Expresso Interestelar". Ele não contém a lógica de backtracking ou de leitura de arquivos, mas atua como o orquestrador central, inicializando os módulos e executando as etapas principais na ordem correta.

```c
#include <stdio.h>
#include "src/readFile/readFile.h"
#include "src/backtracking/pathfinder.h"

int main(int argc, char *argv[]) {
    const char *fname = (argc > 1) ? argv[1] : "entrada.txt";  // usa argumento se houver
    int sr = -1, sc = -1;

    printf("=== EXPRESSO INTERESTELAR ===\n");
    if (!ler_arquivo(fname, &sr, &sc)) return 1;

    debug_mapa(sr, sc);
    debug_conexoes(sr, sc);

    printf("Buscando solucao...\n");
    movimentar(sr, sc, D_init, 0, 1);
    imprimir_solucao();
    libera_estruturas();
    return 0;
}
```

Imagem 12 - main.c

Leitura de Argumentos

const char *fname = (argc > 1) ? argv[1] : "entrada.txt"; Define o arquivo de entrada. O programa verifica se um argumento de linha de comando (argc > 1) foi fornecido. Se sim, argv[1] é usado como o nome do arquivo; caso contrário, ele utiliza o valor padrão "entrada.txt". Isso cumpre o requisito de aceitar arquivos de texto como entrada.

Leitura do Mapa

if (!ler_arquivo(fname, &sr, &sc)) return 1; Invoca a função ler_arquivo (do módulo readFile) para ler e processar o arquivo fname. Esta função aloca dinamicamente todas as estruturas de dados globais (como mapa, visitado, peca_coletada) e determina a posição inicial ('X'), que é retornada para as variáveis sr (linha) e sc (coluna). Se a leitura falhar (retornar false), o programa é encerrado.

Depuração Opcional

debug_mapa(sr, sc); debug_conexoes(sr, sc); Chama as funções de depuração (do módulo map). debug_mapa imprime o mapa lido e os parâmetros de simulação (D, D', A). debug_conexoes verifica e exibe as direções de movimento válidas (CONECTADO/BLOQUEADO) a partir da posição inicial (sr, sc).

Busca Recursiva

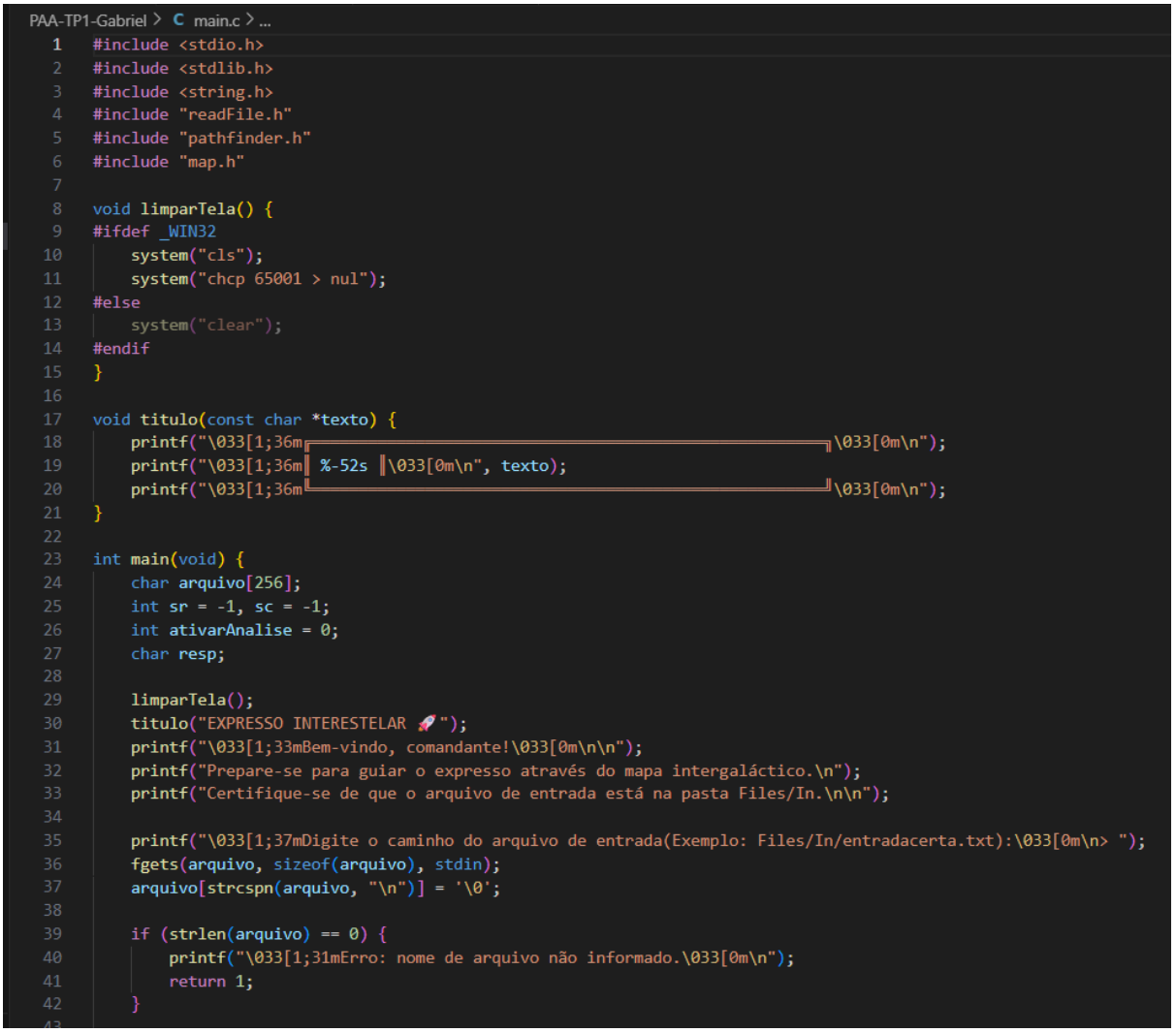
movimentar(sr, sc, D_init, 0, 1); Esta é a única chamada inicial para a função de backtracking movimentar (do módulo pathfinder), conforme exigido pela especificação do trabalho. A busca é iniciada na posição (sr, sc), com a durabilidade inicial (D_init), 0 peças coletadas e na profundidade de recursão 1.

Resultados e Finalização

imprimir_solucao(); libera_estruturas(); Após o término da função movimentar, imprimir_solucao (do módulo pathfinder) é chamada para exibir o resultado final na tela — seja o percurso encontrado ou a mensagem de falha. Por fim, libera_estruturas (do módulo map) é invocada para desalocar com segurança toda a memória dinâmica alocada para as matrizes, evitando vazamentos de memória.

Interface Aprimorada (Tarefa Extra)

Este arquivo é o ponto de entrada do programa e foi modificado para prover uma interface de usuário interativa, indo além do main.c básico anterior.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "readFile.h"
#include "pathfinder.h"
#include "map.h"

void limparTela() {
#ifdef _WIN32
    system("cls");
    system("chcp 65001 > nul");
#else
    system("clear");
#endif
}

void titulo(const char *texto) {
    printf("\033[1;36m╔══════════════════════════════════════════════════╗\033[0m\n");
    printf("\033[1;36m║ %-52s ║\033[0m\n", texto);
    printf("\033[1;36m╚══════════════════════════════════════════════════╝\033[0m\n");
}

int main(void) {
    char arquivo[256];
    int sr = -1, sc = -1;
    int ativarAnalise = 0;
    char resp;

    limparTela();
    titulo("EXPRESSO INTERESTELAR 🚀");
    printf("\033[1;33mBem-vindo, comandante!\033[0m\n\n");
    printf("Prepare-se para guiar o expresso através do mapa intergaláctico.\n");
    printf("Certifique-se de que o arquivo de entrada está na pasta Files/In.\n\n");

    printf("\033[1;37mDigite o caminho do arquivo de entrada(Exemplo: Files/In/entradacerta.txt):\033[0m\n> ");
    fgets(arquivo, sizeof(arquivo), stdin);
    arquivo[strcspn(arquivo, "\n")] = '\0';

    if (strlen(arquivo) == 0) {
        printf("\033[1;31mErro: nome de arquivo não informado.\033[0m\n");
        return 1;
    }
```

Imagem 13 - main.c

Esta seção inicializa a interface, define funções auxiliares de UI (User Interface) e solicita a entrada do usuário de forma interativa.

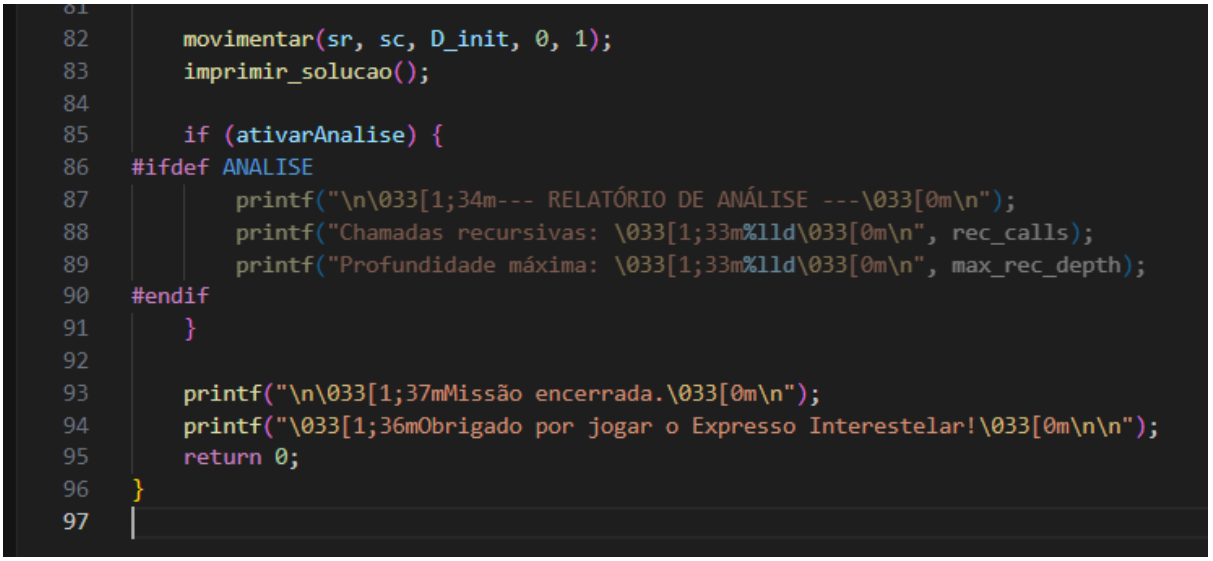
- **limparTela()** Função de conveniência que limpa o terminal. Ela usa compilação condicional (#ifdef WIN32) para ser compatível com Windows (cls) e sistemas POSIX (clear).
- **titulo(const char *texto)** Uma função de aprimoramento visual (extra). Ela utiliza códigos de escape ANSI (\033[...]) para imprimir um texto como um título formatado, centralizado e colorido (ciano).
- **Início da main()** - Entrada Interativa Ao contrário do main.c anterior que usava argumentos de linha de comando (argv) ou um valor padrão, esta versão é totalmente interativa. Ela limpa a tela, exibe um título de boas-vindas ("EXPRESSO INTERESTELAR") e solicita ativamente que o usuário "Digite o caminho do arquivo de entrada", lendo o valor com fgets.

```c
    limparTela();
    titulo("CARREGANDO MAPA...");
    printf("Tentando abrir arquivo: \033[1;34m%s\033[0m\n", arquivo);

    if (!ler_arquivo(arquivo, &sr, &sc)) {
        printf("\n\033[1;31mFalha ao carregar o mapa. Encerrando.\033[0m\n");
        return 1;
    }

    printf("\n\033[1;32mMapa carregado com sucesso!\033[0m\n\n");
    printf("Dimensões: \033[1;36m%d x %d\033[0m\n", R, C);
    printf("Durabilidade inicial: \033[1;33m%d\033[0m | Perda: \033[1;31m%d\033[0m | Ganho: \033[1;32m%d\033[0m\n",
           D_init, D_dec, A_add);
    printf("Ponto inicial: (\033[1;36m%d,%d\033[0m)\n", sr + 1, sc + 1);

    printf("\nDeseja visualizar o mapa completo? (\033[1;32ms\033[0m/\033[1;31mn\033[0m): ");
    scanf(" %c", &resp);
    if (resp == 's' || resp == 'S') {
        limparTela();
        titulo("VISUALIZAÇÃO DO MAPA");
        debug_mapa(sr, sc);
    }

    printf("\nAtivar modo de análise? (\033[1;32ms\033[0m/\033[1;31mn\033[0m): ");
    scanf(" %c", &resp);
    if (resp == 's' || resp == 'S') {
        ativarAnalise = 1;
    }

    limparTela();
    titulo("INICIANDO A MISSÃO 🚀");
    printf("\033[1;37mCalculando rota e verificando conexões...\033[0m\n\n");

#ifdef ANALISE
    rec_calls = 0;
    max_rec_depth = 0;
#endif
```

Imagem 14 - main.c

Esta seção é o núcleo da interatividade pré-execução. Ela carrega o mapa e oferece opções em tempo de real ao usuário.

- **Carregamento e Feedback:** Chama ler_arquivo. Se for bem-sucedido, ele exibe ao usuário os parâmetros que foram carregados (Dimensões, Durabilidade, Ponto Inicial).
- **Visualização Opcional (Extra):** O programa pergunta interativamente se o usuário "Deseja visualizar o mapa completo? (s/n)". Se a resposta for 's', ele chama a função debug_mapa.
- **Modo de Análise em Tempo de Execução (Extra):** Esta é uma implementação aprimorada do "modo de análise". Em vez de ser apenas uma flag de compilação, o programa pergunta ao usuário "Deseja ativar modo de analise? (s/n)". A resposta é armazenada na variável ativarAnálise. Isso permite ao usuário decidir se quer ver as estatísticas de recursão a cada execução.



```
82      movimentar(sr, sc, D_init, 0, 1);
83      imprimir_solucao();
84
85      if (ativarAnalise) {
86  #ifdef ANALISE
87          printf("\n\033[1;34m--- RELATÓRIO DE ANÁLISE ---\033[0m\n");
88          printf("Chamadas recursivas: \033[1;33m%lld\033[0m\n", rec_calls);
89          printf("Profundidade máxima: \033[1;33m%lld\033[0m\n", max_rec_depth);
90  #endif
91      }
92
93      printf("\n\033[1;37mMissão encerrada.\033[0m\n");
94      printf("\033[1;36mObrigado por jogar o Expresso Interestelar!\033[0m\n\n");
95      return 0;
96  }
97
```

Imagem 15 - main.c

Esta seção final executa a busca e exibe os resultados, incluindo o relatório de análise condicional.

- **Execução da Busca:** Inicia a busca (movimentar) e exibe os resultados (imprimir_solucao), de forma similar ao main anterior.
- **Relatório de Análise Condicional:** O "RELATÓRIO DE ANÁLISE" (que exibe Chamadas recursivas e Profundidade máxima) só é impresso se a variável ativarAnálise for verdadeira (definida pelo usuário na etapa anterior) e se o programa foi compilado com a flag ANALISE.
- **Finalização (Extra):** O programa termina com mensagens de encerramento amigáveis ("Missão encerrada.", "Obrigado por jogar o Expresso Interestelar!"), concluindo a experiência de interface aprimorada.

Resultados

Para compilar o projeto, basta utilizar o Makefile fornecido. Conforme especificado nos requisitos do trabalho, foram criadas regras distintas para a compilação padrão e para o "Modo de Análise":

Compilação Padrão:

- Comando: make all
- Execução: make run ou ./pathfinder

Modo de Análise:

- Comando: make analyse
- Execução: make run-analyse

Também pode usar o comando:

```
None
gcc -Wall -Wextra -std=c11 main.c src/map/map.c src/readFile/readFile.c
src/backtracking/pathfinder.c -Isrc/map -Isrc/readFile -Isrc/backtracking -o
pathfinder
```

Para avaliar o funcionamento do algoritmo desenvolvido, foram realizados dois testes distintos utilizando o mesmo mapa-base. Em ambos os casos, o arquivo de entrada seguiu o formato especificado no enunciado, contendo as informações de durabilidade inicial, perda por movimento, ganho por peça e a matriz que representa o cenário do problema. O objetivo foi observar o comportamento do programa diante de uma situação solucionável e outra inviável, demonstrando sua capacidade de identificar corretamente a existência ou não de um caminho possível entre o ponto inicial X e o destino final F.

Caso com solução

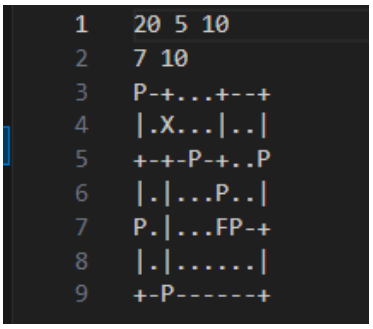


Imagem 16 - entrada1

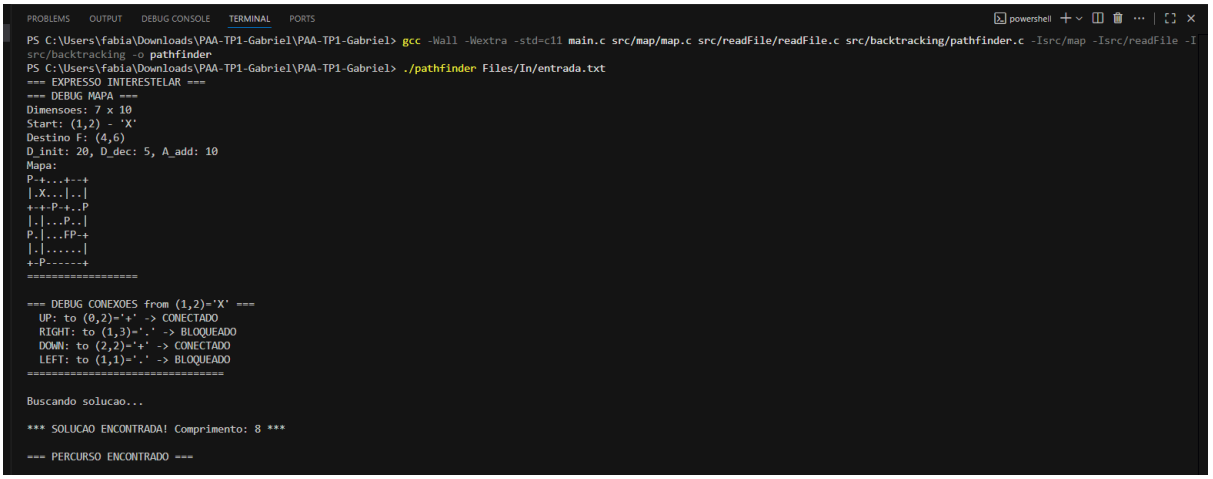


Imagem 18 - Saída

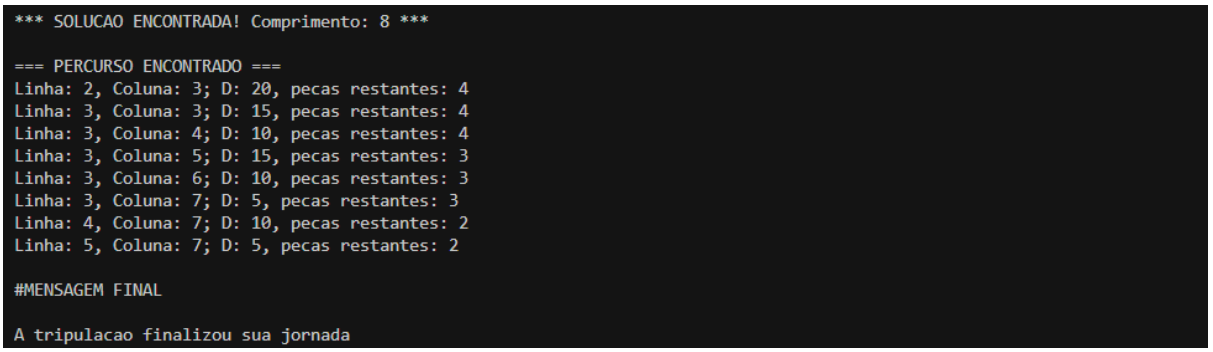


Imagem 19 - Saída

No primeiro teste, o arquivo de entrada (map1.txt) definiu uma durabilidade inicial de 20 unidades, perda de 5 por movimento e ganho de 10 ao coletar uma peça. O mapa continha sete linhas e dez colunas, com quatro peças P distribuídas em diferentes regiões.

Ao executar o programa, o algoritmo iniciou a busca a partir do ponto X e, utilizando o método de backtracking, explorou os caminhos válidos conforme as conexões horizontais e verticais permitidas. Durante o percurso, uma peça foi coletada, o que restaurou parte da durabilidade e permitiu que o expresso alcançasse o destino F com cinco unidades restantes. A saída impressa mostrou o detalhamento de cada passo, com as posições visitadas, a durabilidade atual e o número de peças restantes. Ao final, foi exibida a mensagem “A tripulação finalizou sua jornada”, confirmando o sucesso da navegação e a correção do algoritmo na aplicação das regras de movimentação e coleta. Esse resultado demonstra que o programa conseguiu interpretar corretamente o mapa e ajustar os valores de durabilidade conforme os eventos do trajeto, encontrando uma rota viável até o destino.

Nova Interface

Agora vamos mostrar o mesmo teste com solução anterior, só que com a interface interativa criada , demonstrando o fluxo de interação com o usuário.

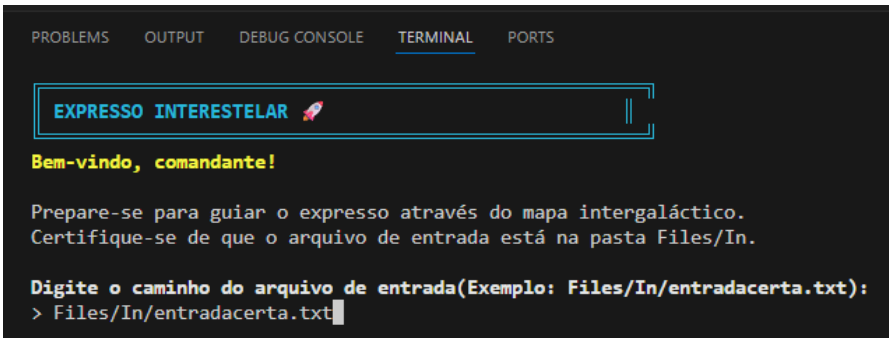


Imagem 20 - Saída

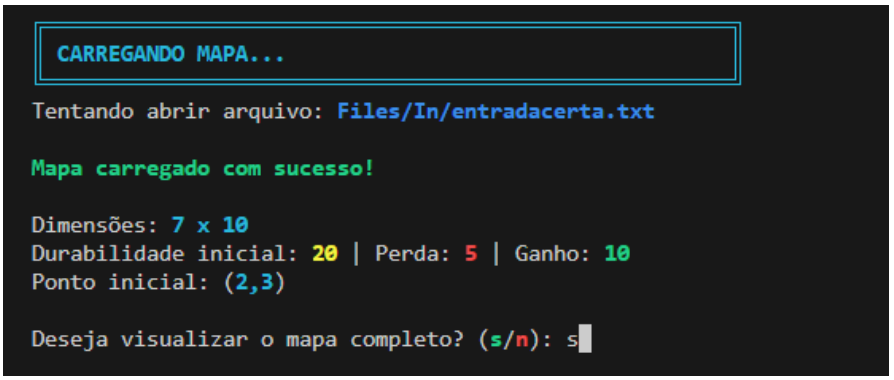


Imagem 21 - Saída

```
VISUALIZAÇÃO DO MAPA

=== DEBUG MAPA ===
Dimensoes: 7 x 10
Start: (1,2) - 'X'
Destino F: (4,6)
D_init: 20, D_dec: 5, A_add: 10
Mapa:
P-+...+--+
|.X...|..|
+-+-P-+..P
|.|...P..|
P.|...FP-+
|.|......|
+-P------+
==================


Ativar modo de análise? (s/n): s
```

Imagem 22 - Saída

```
INICIANDO A MISSÃO 🚀

Calculando rota e verificando conexões...


*** SOLUCAO ENCONTRADA! Comprimento: 8 ***

=== PERCURSO ENCONTRADO ===
Linha: 2, Coluna: 3; D: 20, pecas restantes: 4
Linha: 3, Coluna: 3; D: 15, pecas restantes: 4
Linha: 3, Coluna: 4; D: 10, pecas restantes: 4
Linha: 3, Coluna: 5; D: 15, pecas restantes: 3
Linha: 3, Coluna: 6; D: 10, pecas restantes: 3
Linha: 3, Coluna: 7; D: 5, pecas restantes: 3
Linha: 4, Coluna: 7; D: 10, pecas restantes: 2
Linha: 5, Coluna: 7; D: 5, pecas restantes: 2

#MENSAGEM FINAL

A tripulacao finalizou sua jornada

Missão encerrada.
Obrigado por jogar o Expresso Interestelar!

PS C:\Users\fabia\Downloads\PAA-TP1-Fabiano\PAA-TP1-Gabriel>
```

Imagem 23 - Saída

Caso sem solução

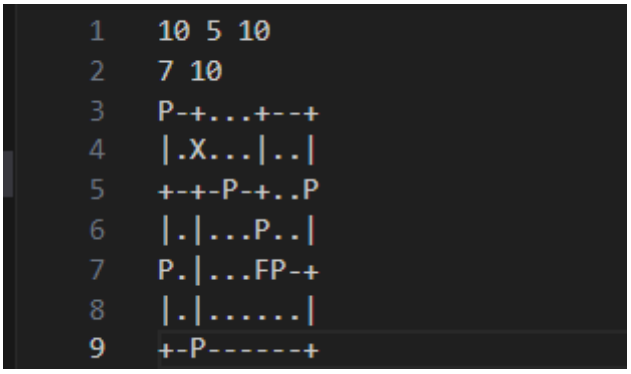


Imagem 23 - entrada

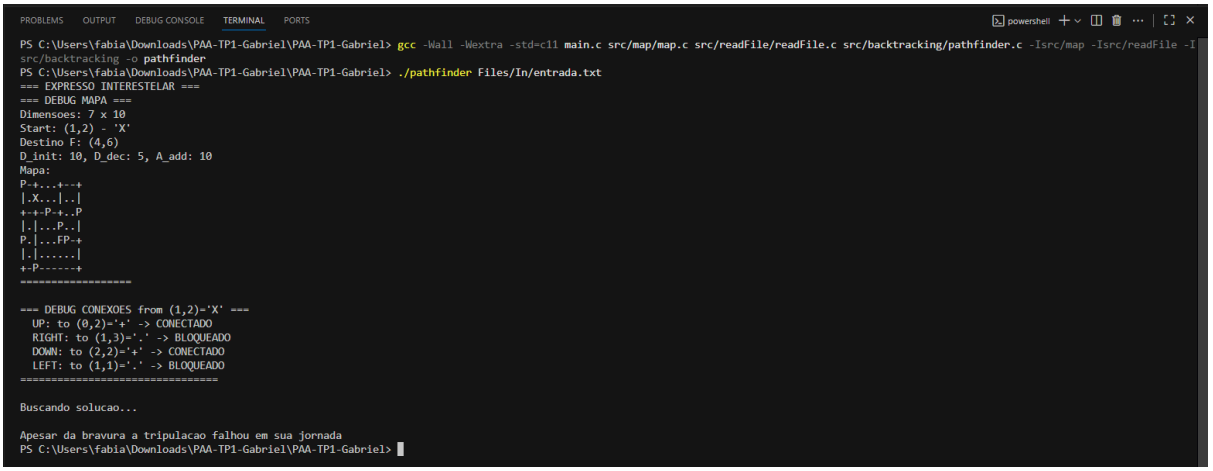


Imagem 24 - saída

No segundo teste, utilizou-se o mesmo mapa, mas com a durabilidade inicial reduzida para 10 unidades, mantendo os demais parâmetros iguais. Essa pequena alteração impossibilitou a chegada ao destino antes que a durabilidade fosse completamente consumida. Durante a execução, o algoritmo iniciou a exploração a partir do ponto X e percorreu algumas posições iniciais, porém a energia do expresso se esgotou antes de alcançar a primeira peça capaz de restaurá-la. Ao detectar que a durabilidade havia se tornado negativa, o programa interrompeu automaticamente a busca naquele ramo e começou o processo de retrocesso, até não restar mais caminhos viáveis a serem explorados. Como nenhuma rota levou ao destino F, o sistema exibiu a mensagem “Apesar da bravura a tripulação falhou em sua jornada”. Esse resultado evidencia o funcionamento correto dos critérios de parada e do controle de durabilidade: o algoritmo reconheceu a impossibilidade de prosseguir e finalizar de forma segura, sem entrar em loops ou acessar posições inválidas. O comportamento observado confirma a robustez da implementação na detecção de estados terminais e na prevenção de chamadas recursivas desnecessárias.

Nova Interface

Agora vamos mostrar o mesmo teste sem solução anterior, só que com a interface interativa criada, demonstrando o fluxo de interação com o usuário.

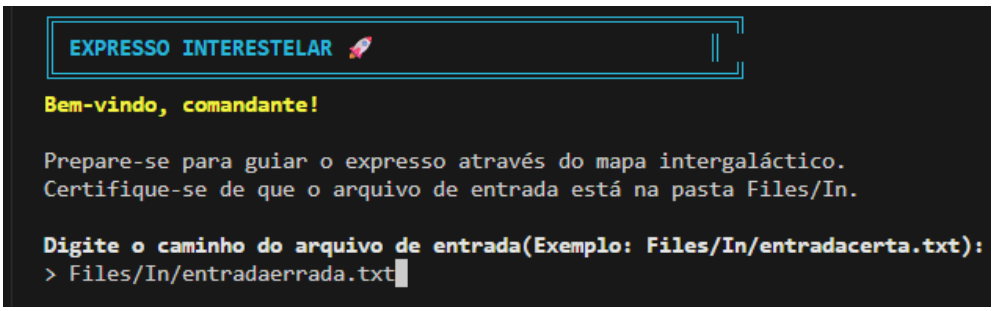


Imagem 25 - Saída

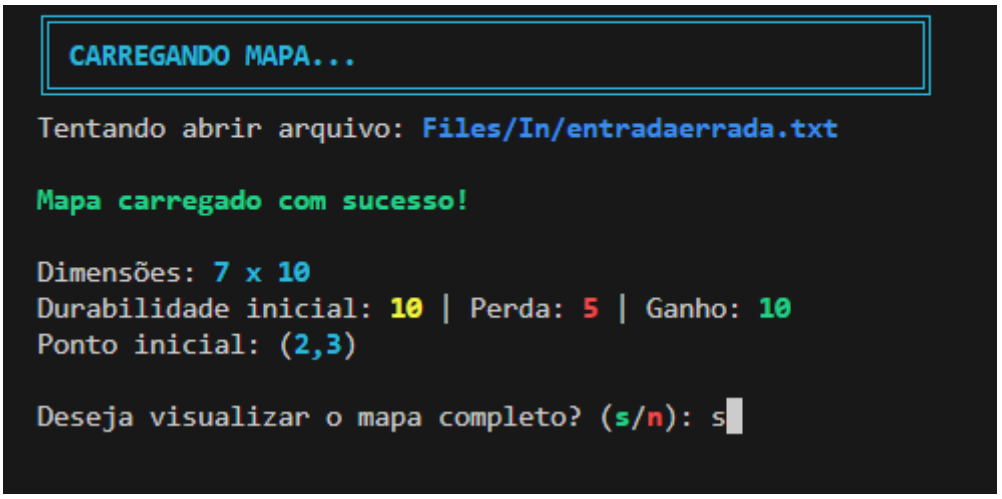


Imagem 26 - Saída

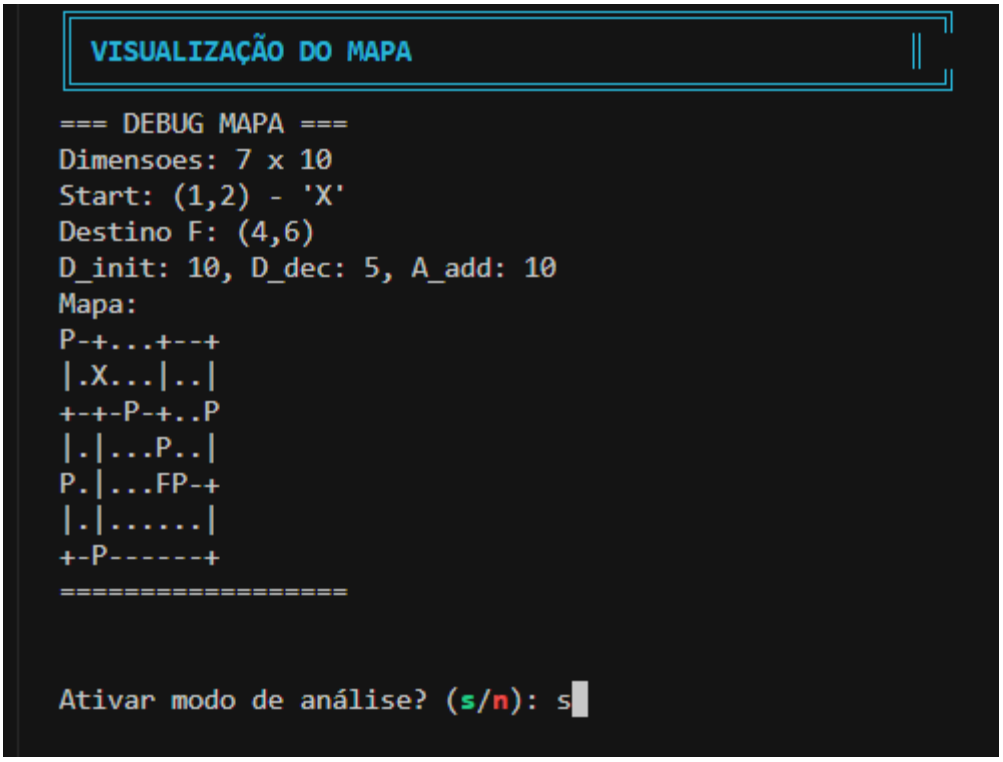


Imagem 27 - Saída

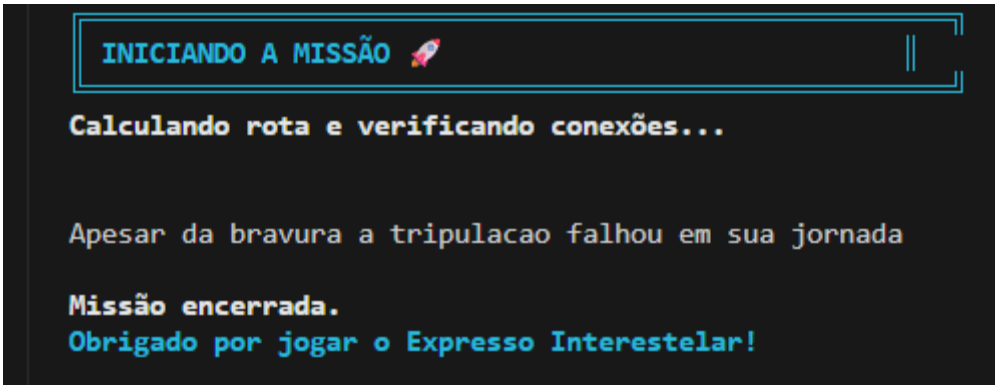


Imagem 28 - Saída

Conclusão

A implementação proposta atendeu integralmente aos requisitos definidos no enunciado do trabalho, demonstrando na prática o funcionamento do algoritmo de backtracking aplicado à resolução de problemas de busca e exploração de caminhos. O programa foi capaz de identificar rotas válidas, contabilizar chamadas recursivas e registrar o nível máximo de recursão atingido, evidenciando a eficiência da abordagem. Além disso, o uso de alocação dinâmica e a separação entre modo normal e modo de análise contribuíram para um código modular, eficiente e de fácil compreensão. O trabalho permitiu aprofundar o entendimento dos princípios de recursão e backtracking, destacando sua importância na solução de problemas combinatórios e de exploração de estados.

Referências

- BARBOSA, Daniel Mendes. *Trabalho Prático 1 – Projeto e Análise de Algoritmos*. Universidade Federal de Viçosa – Campus Florestal, 2025.

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.

- SEDGEWICK, R.; WAYNE, K. *Algorithms*. 4th ed. Addison-Wesley, 2011.

- Manual e especificações disponíveis em: https://docs.google.com/document/d/1D-N08cR_zp9CHSaYmw_h36BOp6lyca4aq0Qt6K2Oje8/edit