



UNIVERSIDADE FEDERAL DE VIÇOSA - CAMPUS FLORESTAL

Décio Cançado Barcelos

"TRABALHO PRÁTICO - 1"

Trabalho Prático de Projeto e Análise de algoritmos

2025

Florestal - MG

2025

"A Jornada do expresso interestelar"

Décio Cançado Barcelos[5782]

Davi de souza [5368]

Henrique campos[4673]

Guilherme Gustavo[5378]

FLORESTAL

Sumário

1. Introdução

2. Desenvolvimento

2.1 Visão Geral do Problema

2.2 Estratégia de Solução

2.3 Estruturas de Dados

2.3.1 Estrutura Mapa

2.3.2 Estrutura Analise

2.3.3 Justificativa para Alocação Dinâmica

2.4 Implementação do Algoritmo

2.4.1 Condições de Parada (Casos Base)

2.4.2 Processamento da Célula Atual

2.4.3 Cálculo da Durabilidade

2.4.4 Movimentos Baseados no Tipo de Célula

2.4.5 Mecanismo de Backtracking

2.5 Modo de Análise

3. Compilação e Execução

3.1 Comandos de Compilação

3.2 Comandos de Execução

3.3 Outros Comandos Úteis

3.4 Estrutura de Diretórios

4. Testes e Resultados

5. Decisões de Implementação

6. Conclusão

6.1 Resultados Alcançados

6.2 Possíveis Melhorias Futuras

7. Referências

1. Introdução

Este trabalho implementa uma solução computacional para o problema do “Expresso Interestelar”, no qual uma tripulação precisa navegar por uma rodovia estelar representada por um mapa bidimensional. O desafio consiste

em encontrar um caminho viável do ponto inicial (X) até o destino final (F), gerenciando a durabilidade limitada da nave.

O projeto foi desenvolvido em linguagem C, utilizando obrigatoriamente a técnica de **backtracking** para explorar todas as rotas possíveis de forma recursiva. A implementação inclui gerenciamento de memória dinâmica, leitura de arquivos de configuração e um modo especial de análise para contabilizar métricas de desempenho do algoritmo recursivo.

2. Desenvolvimento

2.1 Visão Geral do Problema

A tripulação parte de uma posição inicial X em um mapa que contém diferentes tipos de células:

X: ponto de partida

F: destino final

P: peças de reparo (4 no total)

-: caminho horizontal

|: caminho vertical

+: cruzamento (permite movimento em qualquer direção)

.: obstáculo intransponível

A nave possui uma durabilidade D que diminui em D' unidades a cada movimento. Ao coletar uma peça P , a durabilidade aumenta em A unidades. Quando as 4 peças são coletadas, a durabilidade para de diminuir, garantindo chegada ao destino.

A “missão” é encontrar um caminho que permita à tripulação chegar ao destino F antes que a durabilidade chegue a zero, considerando as restrições de movimento de cada tipo de célula.

2.2 Estratégia de Solução

A solução foi implementada utilizando o **paradigma de backtracking**, que explora recursivamente todas as possibilidades de caminho até encontrar uma solução válida ou determinar que não existe.

O algoritmo funciona da seguinte forma:

1. **Inicialização:** começa na posição X com durabilidade inicial D e zero peças coletadas.
2. **Exploração:** em cada célula, tenta mover em todas as direções permitidas pelo tipo da célula atual.
3. **Validação:** verifica se o movimento é válido (dentro dos limites, com durabilidade suficiente e célula não visitada).
4. **Marcação:** marca a célula como visitada para evitar ciclos.

5. **Recursão:** chama a função recursiva para as próximas posições.
6. **Backtracking:** se um caminho não leva à solução, desmarca a célula e tenta outra direção.
7. **Sucesso:** retorna verdadeiro quando encontra o destino **F**.

Este método garante que **todas as possibilidades sejam exploradas sistematicamente**, encontrando uma solução se ela existir.

2.3 Estruturas de Dados

Para implementar a solução, foram criadas três estruturas principais.

2.3.1 Estrutura Mapa

Armazena os dados do jogo, incluindo dimensões do mapa, parâmetros de durabilidade e duas matrizes alocadas dinamicamente.

```
typedef struct{
    int altura;
    int largura;
    int durabilidadeInicial; //D
    int custoMovimento; //D'
    int bonusPeca; //A
    int posInicial_coluna;
    int posInicial_linha;
    char **matrizMapa; //matriz para o mapa
    bool **localVisitado; // matriz para marcar os lugares que ja foram visitados
} Mapa;
```

2.3.2 Estrutura Analise

Utilizada exclusivamente no modo de análise para contabilizar métricas de desempenho:

```
typedef struct{  
    long long chamadaRecursiva;  
    int profundidadeMaxima;  
} Analise;
```

2.3.3 Justificativa para Alocação Dinâmica

Como as dimensões do mapa são conhecidas apenas após a leitura do arquivo de entrada, foi necessário utilizar **alocação dinâmica de memória**.

Isso permite flexibilidade para trabalhar com mapas de tamanhos variados **sem desperdício de memória** ou **limitações arbitrárias de tamanho**.

2.4 Implementação do Algoritmo

O coração do algoritmo está na função `movimentar()` em `backtracking.c`.

2.4.1 Condições de Parada (Casos Base)

Verificações que impedem o movimento:

```

//Casos de falha
//Sair dos limites
if(linha < 0 || linha >= mapa->altura || coluna < 0 || coluna >= mapa->largura){
    return false;
}
//Durabilidade acabar
if(durabilidadeAtual <= 0){
    return false;
}
//Local onde nao pode passar "."
if(mapa->matrizMapa[linha][coluna] == '.'){
    return false;
}
//Local ja foi visitado
if(mapa->localVisitado[linha][coluna] == true){
    return false;
}

```

2.4.2 Processamento da Célula Atual

```

//Local ja foi visitado

if(mapa->localVisitado[linha][coluna] == true){
    return false;
}

//Processar a célula atual

mapa->localVisitado[linha][coluna] = true;

char celulaAtual = mapa->matrizMapa[linha][coluna];
if(celulaAtual == 'P'){
    pecasColetadas++;
    durabilidadeAtual += mapa->bonusPeca;
}

```

2.4.3 Cálculo da Durabilidade

```
//Passo recursivo
int proxDurabilidade = durabilidadeAtual;
if(pecasColetadas < 4){
    proxDurabilidade -= mapa->custoMovimento;
}
```

Após coletar as 4 peças, a durabilidade **não diminui mais**.

2.4.4 Movimentos Baseados no Tipo de Célula

O algoritmo respeita as restrições de movimento de cada célula:

```
//Movimentacoes Verticais
if(celulaAtual == '|' || celulaAtual == '+' || celulaAtual == 'X' || celulaAtual == 'P'){
    if(movimentar(mapa,analise,linha + 1,coluna,proxDurabilidade,pecasColetadas,profundidadeAtual+1)){
        return true;
    }
    if(movimentar(mapa,analise,linha - 1,coluna,proxDurabilidade,pecasColetadas,profundidadeAtual+1)){
        return true;
    }
}

//Movimentacoes Horizontais
if(celulaAtual == '-' || celulaAtual == '+' || celulaAtual == 'X' || celulaAtual == 'P'){
    if(movimentar(mapa,analise,linha,coluna + 1,proxDurabilidade,pecasColetadas,profundidadeAtual + 1)){
        return true;
    }
    if(movimentar(mapa,analise,linha,coluna - 1,proxDurabilidade,pecasColetadas,profundidadeAtual + 1)){
        return true;
    }
}
```

2.4.5 Mecanismo de Backtracking

```
mapa->localVisitado[linha][coluna] = false;  
return false;
```

Permite que a célula seja visitada novamente por outros caminhos.

2.5 Modo de Análise

O modo de análise é ativado com a flag **-DMODO_ANALISE**, permitindo coletar métricas sem afetar o desempenho normal:

Vantagens:

- Zero overhead no modo normal
- Fácil manutenção
- Controle simples via **Makefile**

3. Como Compilar e Executar

Este projeto utiliza um Makefile para automatizar o processo de compilação. Todos os comandos a seguir devem ser executados a partir do terminal, na pasta raiz do projeto.

3.1 Limpando o Projeto (Opcional)

Antes de compilar, é uma boa prática limpar quaisquer arquivos de compilação anteriores.

```
make clean
```

Este comando remove as pastas bin/ e obj/, garantindo uma compilação limpa.

3.2. Compilação Padrão

Para compilar o programa no modo de execução normal, utilize o comando:

```
make
```

Isso irá compilar todos os arquivos-fonte e criar o executável final no diretório bin/, com o nome `expresso_interstellar`.

3.3. Compilação em Modo de Análise

O projeto suporta um modo de análise que, ao final da execução, exibe a quantidade total de chamadas recursivas e a profundidade máxima de recursão alcançada.

Para compilar o programa neste modo, utilize o comando:

```
make ANALISE=1
```

3.4. Executando o Programa

Após a compilação (seja ela padrão ou em modo de análise), o programa pode ser executado com o seguinte comando:

```
./bin/expresso_interstelar
```

Após a execução, o programa solicitará ao usuário que digite o caminho para o arquivo de mapa que deseja processar. Por exemplo:

Digite o nome do arquivo de entrada (ex: testes/mapa3.txt): testes/mapa_caso2.txt

O programa então executará o algoritmo de backtracking para o mapa fornecido e exibirá o rastro da jornada, seguido pela mensagem de resultado final.

4. Testes e Resultados

```
Davi@Davi MINGW64 /c/Users/david/TP1-PAA-Backtracking
$ ./bin/expresso_interestelar
```

```
=====
A Jornada do Expresso Interestelar
=====
```

```
Digite o nome do arquivo de entrada:
```

```
testes/mapa2.txt
```

```
-----Iniciando Jornada-----
```

```
Linha: 1, Coluna: 1, Durabilidade: 100, Pecas restantes: 4
Linha: 1, Coluna: 2, Durabilidade: 90, Pecas restantes: 4
Linha: 1, Coluna: 3, Durabilidade: 80, Pecas restantes: 4
Linha: 2, Coluna: 3, Durabilidade: 70, Pecas restantes: 4
Linha: 3, Coluna: 3, Durabilidade: 60, Pecas restantes: 4
Linha: 4, Coluna: 3, Durabilidade: 50, Pecas restantes: 4
Linha: 5, Coluna: 3, Durabilidade: 70, Pecas restantes: 3
Linha: 3, Coluna: 4, Durabilidade: 50, Pecas restantes: 4
Linha: 3, Coluna: 5, Durabilidade: 40, Pecas restantes: 4
```

```
#MENSAGEM FINAL
```

```
A tripulacao finalizou sua jornada
```

```
-----Fim da Jornada-----
```

```
Deseja executar com outro arquivo? (s/n): n
```

```
Encerrando o programa.
```

```
Davi@Davi MINGW64 /c/Users/david/TP1-PAA-Backtracking
```

```
$ make run_analise
```

```
makefile:119: warning: overriding recipe for target 'run'
```

```
makefile:101: warning: ignoring old recipe for target 'run'
```

```
makefile:124: warning: overriding recipe for target 'run_analise'
```

```
makefile:110: warning: ignoring old recipe for target 'run_analise'
```

```
\033[0;32m✓ Compilação concluída com sucesso (modo ANÁLISE)!\033[0m
```

```
\033[0;34mExecute com: ./bin/expresso_interestelar_analise\033[0m
```

```
\033[0;34m=== EXECUTANDO MODO ANÁLISE ===\033[0m
```

```
=====
A Jornada do Expresso Interestelar
=====
```

```
Digite o nome do arquivo de entrada:
```

```
testes/mapa2.txt
```

```
-----Iniciando Jornada-----
```

```
Linha: 1, Coluna: 1, Durabilidade: 100, Pecas restantes: 4
Linha: 1, Coluna: 2, Durabilidade: 90, Pecas restantes: 4
Linha: 1, Coluna: 3, Durabilidade: 80, Pecas restantes: 4
Linha: 2, Coluna: 3, Durabilidade: 70, Pecas restantes: 4
Linha: 3, Coluna: 3, Durabilidade: 60, Pecas restantes: 4
Linha: 4, Coluna: 3, Durabilidade: 50, Pecas restantes: 4
Linha: 5, Coluna: 3, Durabilidade: 70, Pecas restantes: 3
Linha: 3, Coluna: 4, Durabilidade: 50, Pecas restantes: 4
Linha: 3, Coluna: 5, Durabilidade: 40, Pecas restantes: 4
```

```
#MENSAGEM FINAL
```

```
A tripulacao finalizou sua jornada
```

```
-----Fim da Jornada-----
```

```
=====
| MODO ANÁLISE |
=====
```

```
Total de chamadas recursivas: 17
```

```
Profundidade maxima de recursao: 7
```

```
Deseja executar com outro arquivo? (s/n): n
```

```
Encerrando o programa.
```

4.1 Análise dos Resultados

Os testes demonstram que:

1. O algoritmo funciona corretamente para mapas **sem solução**, identificando quando a durabilidade é insuficiente.
2. Para mapas **com solução**, encontra um caminho válido respeitando todas as restrições.
3. O sistema de coleta de peças **aumenta a durabilidade corretamente**.
4. O comportamento da nave muda após coletar as 4 peças.
5. O modo de análise fornece **informações valiosas sobre o desempenho**.

A quantidade de chamadas recursivas varia conforme o tamanho e a complexidade do mapa, demonstrando a natureza **exploratória do backtracking**.

5. Decisões de Implementação

5.1 Linguagem e Estrutura do Código

O código foi modularizado em:

- `mapa.c/h` – gerenciamento do mapa e leitura de arquivos
- `backtracking.c/h` – implementação do algoritmo de busca
- `main.c` – interface e controle de execução

Essa divisão **facilita manutenção, teste e compreensão**.

5.2 Ordem de Exploração dos Movimentos

A ordem adotada foi:

1. Movimentos verticais: **baixo**, depois **cima**
2. Movimentos horizontais: **direita**, depois **esquerda**

Ordem arbitrária, mas consistente.

Ordens diferentes podem gerar caminhos diferentes (mas igualmente válidos).

5.3 Tratamento de Erros

Foram incluídas verificações robustas para:

- Abertura de arquivo
- Alocação de memória
- Formato de entrada
- Existência do ponto inicial **X**

Em caso de erro, o programa exibe uma mensagem e libera toda a memória antes de encerrar.

5.4 Gerenciamento de Memória

Todas as alocações dinâmicas são liberadas pela função `liberarMapa()`.

Ela é chamada inclusive em casos de erro, garantindo **ausência de vazamentos**.

5.5 Modo de Análise

Implementado via **compilação condicional** ao invés de opção em tempo de execução.

Vantagens:

- Elimina overhead desnecessário

- Simplifica a lógica
 - Controlado pelo **Makefile**
-

5.6 Limitações Conhecidas

Funcionalidades implementadas:

- Leitura de mapas de vários tamanhos
- Busca de caminhos com restrições
- Coleta de peças e controle de durabilidade
- Detecção de impossibilidade
- Modo de análise com métricas

Não implementado:

- Busca pelo caminho mais curto
- Interface gráfica

- Gerador automático de testes
 - Busca de múltiplas soluções
-

5.7 Decisão sobre Células Tipo P e X

As células **P** (peças) e **X** (início) aceitam movimentos em qualquer direção.

Isso aumenta a flexibilidade e reflete a natureza **especial** dessas posições.

6. Conclusão

Este trabalho proporcionou experiência prática valiosa na implementação e análise de **backtracking**.

Principais aprendizados:

1. Compreensão aprofundada do funcionamento do backtracking.
2. Uso correto de **alocação dinâmica e liberação de memória** em C.
3. Importância da **modularização** para clareza e manutenção.
4. Observação prática da **complexidade recursiva**.

5. Coordenação e **trabalho em equipe**.

7. Referências

1. CORMEN, Thomas H. et al. *Algoritmos: Teoria e Prática*. 3ª ed. Rio de Janeiro: Elsevier, 2012.
2. BARBOSA, Daniel Mendes. *Slides da disciplina Projeto e Análise de Algoritmos*. UFV, 2025.
3. *Documentação da Linguagem C*. cppreference.com
4. ZIVIANI, Nivio. *Projeto de Algoritmos com Implementações em Pascal e C*. 3ª ed. São Paulo: Cengage Learning, 2011.
5. *GNU Make Manual*. Disponível em:
<https://www.gnu.org/software/make/manual/>
6. SEDGEWICK, Robert; WAYNE, Kevin. *Algorithms*. 4ª ed. Addison-Wesley, 2011.

