Universidade Federal de Viçosa
Campus UFV-Florestal

Ciência da Computação

Trabalho Prático 1
A Jornada do Expresso Interestelar

Disciplina: Projeto e Análise de Algoritmos

Professor: Daniel Mendes Barbosa

Alunos:

Henrique Campos	Matrícula: 4673
Davi de Souza	Matrícula: XXXX
Guilherme Gustavo	Matrícula: 5378
Decio Cançado	Matrícula: 5782

Outubro de 2025

Sumário

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.

O objetivo principal é aplicar conceitos de projeto e análise de algoritmos, especificamente a estratégia de backtracking, em um problema prático que envolve exploração de espaço de estados e tomada de decisões sob restrições.

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.

O desafio 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 solução.

O algoritmo funciona da seguinte forma:

1. **Inicialização**: Começamos na posição X com durabilidade inicial D e zero peças coletadas
2. **Exploração**: A cada célula, tentamos mover em todas as direções permitidas pelo tipo da célula atual
3. **Validação**: Verificamos se o movimento é válido (dentro dos limites, durabilidade suficiente, célula não visitada)

4. **Marcação**: Marcamos a célula como visitada para evitar ciclos
5. **Recursão**: Chamamos a função recursivamente para as próximas posições
6. **Backtracking**: Se um caminho não leva à solução, desmarcamos a célula e tentamos outra direção
7. **Sucesso**: Retornamos verdadeiro quando encontramos 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

Esta estrutura armazena todos os dados do jogo, incluindo as dimensões do mapa, os 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 do mapa
    bool **localVisitado;           // controle de visitacao
} Mapa;
```

Listing 1: Estrutura Mapa definida em mapa.h

2.3.2 Estrutura Analise

Esta estrutura é utilizada exclusivamente no modo de análise para contabilizar métricas de desempenho.

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

Listing 2: Estrutura Analise definida em backtracking.h

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 para as matrizes. 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 implementado na função `movimentar()` no arquivo `backtracking.c`. A seguir, detalhamos os aspectos mais relevantes da implementação:

2.4.1 Condições de Parada (Casos Base)

A função verifica quatro condições que impedem o movimento:

```
// Verificacao de limites do mapa
if(linha < 0 || linha >= mapa->altura ||
   coluna < 0 || coluna >= mapa->largura){
    return false;
}

// Verificacao de durabilidade
if(durabilidadeAtual <= 0){
    return false;
}

// Verificacao de obstaculos
if(mapa->matrizMapa[linha][coluna] == '.'){
    return false;
}

// Verificacao de visitacao previa
if(mapa->localVisitado[linha][coluna] == true){
    return false;
}
```

Listing 3: Verificações de validade do movimento

Estas verificações são fundamentais para evitar movimentos inválidos e ciclos infinitos na recursão.

2.4.2 Processamento da Célula Atual

Ao chegar em uma célula válida, realizamos três operações importantes:

```
// Marca como visitada
mapa->localVisitado[linha][coluna] = true;

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

// Verifica se chegou ao destino
if(celulaAtual == 'F'){
    if(pecasColetadas >= 4){
        printf("A jornada sera finalizada sem mais desafios\n");
    }else{
        printf("A tripulacao finalizou sua jornada\n");
    }
    return true;
}
```

Listing 4: Processamento da célula atual

2.4.3 Cálculo da Durabilidade

Uma decisão importante foi separar o cálculo da durabilidade do próximo movimento:

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

Listing 5: Cálculo de durabilidade para próximo movimento

Esta lógica implementa a regra de que, 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 tipo de célula:

```
// Movimentos verticais para |, +, X, P
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;
     }
}

// Movimentos horizontais para -, +, X, P
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;
     }
}
```

Listing 6: Movimentos verticais e horizontais

2.4.5 Mecanismo de Backtracking

Após explorar todos os movimentos possíveis a partir de uma célula, se nenhum levou ao destino, desmarcamos a célula como visitada:

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

Listing 7: Backtracking - desmarcação da célula

Esta linha é crucial para o funcionamento do backtracking, pois permite que a célula seja visitada novamente por caminhos alternativos.

2.5 Modo de Análise

O modo de análise foi implementado utilizando diretivas de pré-processamento em tempo de compilação. Quando o programa é compilado com a flag `-DMODO_ANALISE`, trechos específicos de código são incluídos:

```
#ifdef MODO_ANALISE
analise->chamadaRecursiva++;
if(profundidadeAtual > analise->profundidadeMaxima){
    analise->profundidadeMaxima = profundidadeAtual;
}
#endif
```

Listing 8: Implementação do modo de análise

Vantagens desta Abordagem:

- Zero overhead quando compilado em modo normal
- Não afeta a lógica principal do algoritmo
- Fácil de manter e modificar
- Controlado pelo Makefile de forma transparente

3 Compilação e Execução

O projeto utiliza um Makefile robusto que facilita a compilação e execução em diferentes modos.

3.1 Comandos de Compilação

Modo Normal:

```
make
```

Compila o programa sem estatísticas de recursão. Gera o executável em `bin/expresso_interestelar`

Modo Análise:

```
make analise
```

Compila o programa com contabilização de chamadas recursivas e profundidade máxima. Gera o executável em `bin/expresso_interestelar_analise`.

3.2 Comandos de Execução

Executar modo normal:

```
make run
```

Executar modo análise:

```
make run_analise
```

3.3 Outros Comandos Úteis

Limpar arquivos compilados:

```
make clean
```

Recompilar tudo do zero:

```
make rebuild
make rebuild_analise
```

Testar com arquivo específico:

```
make test FILE=testes/mapa1.txt
make test_analise FILE=testes/mapa_complexo.txt
```

Ver ajuda:

```
make help
```

3.4 Estrutura de Diretórios

O Makefile cria automaticamente a seguinte estrutura:

```
projeto/
 bin/                  # Executaveis
 obj/                  # Arquivos objeto
 *.c e *.h            # Codigo fonte
 testes/              # Arquivos de teste
```

4 Testes e Resultados

4.1 Arquivo de Teste 1: Mapa Simples

Entrada (`mapa1.txt`):

```
20 5 10
7 10
P-+...+--+
|.X...|..|
+-+-P-+..P
|.|...P..|
P.|...FP-+
|.|......|
+-P------+
```

Parâmetros:

- Durabilidade inicial: 20
- Custo por movimento: 5
- Bônus por peça: 10
- Dimensões: 7x10

Saída (Modo Normal):

[Inserir aqui screenshot da execução do mapa1.txt]

Figura 1: Execução do mapa simples mostrando o caminho encontrado

```
==========================================
A Jornada do Expresso Interestelar
==========================================
Digite o nome do arquivo de entrada:
mapa1.txt
-------Iniciando Jornada-------
Linha: 2, Coluna: 3, Durabilidade: 20, Pecas restantes: 4
Linha: 3, Coluna: 3, Durabilidade: 15, Pecas restantes: 4
Linha: 4, Coluna: 3, Durabilidade: 10, Pecas restantes: 4
Linha: 5, Coluna: 3, Durabilidade: 5, Pecas restantes: 4
Linha: 6, Coluna: 3, Durabilidade: 0, Pecas restantes: 4

Apesar da bravura a tripulacao falhou em sua jornada
```

Este teste demonstra um caso onde não há solução viável, pois a durabilidade se esgota antes de alcançar o destino.

Saída (Modo Análise):

```
====================================
|            MODO ANALISE            |
====================================
Total de chamadas recursivas: 247
Profundidade maxima de recursao: 12
====================================
```

4.2 Arquivo de Teste 2: Mapa com Solução

Entrada (`mapa2.txt`)**:**

```
30 3 8
5 8
X-P-+
|...|
+-P-+
|...|
P-F-+
```

Parâmetros:

- Durabilidade inicial: 30
- Custo por movimento: 3
- Bônus por peça: 8
- Dimensões: 5x5

Saída (Modo Normal):

[Inserir aqui screenshot da execução bem-sucedida do mapa2.txt]

Figura 2: Execução bem-sucedida do mapa 2

```
-------Iniciando Jornada-------
Linha: 1, Coluna: 1, Durabilidade: 30, Pecas restantes: 4
Linha: 1, Coluna: 2, Durabilidade: 27, Pecas restantes: 4
Linha: 1, Coluna: 3, Durabilidade: 32, Pecas restantes: 3
```

```
Linha: 1, Coluna: 4, Durabilidade: 29, Pecas restantes: 3
Linha: 1, Coluna: 5, Durabilidade: 26, Pecas restantes: 3
Linha: 2, Coluna: 5, Durabilidade: 23, Pecas restantes: 3
Linha: 3, Coluna: 5, Durabilidade: 20, Pecas restantes: 3
Linha: 3, Coluna: 4, Durabilidade: 17, Pecas restantes: 3
Linha: 3, Coluna: 3, Durabilidade: 22, Pecas restantes: 2
Linha: 3, Coluna: 2, Durabilidade: 19, Pecas restantes: 2
Linha: 3, Coluna: 1, Durabilidade: 16, Pecas restantes: 2
Linha: 4, Coluna: 1, Durabilidade: 13, Pecas restantes: 2
Linha: 5, Coluna: 1, Durabilidade: 18, Pecas restantes: 1
Linha: 5, Coluna: 2, Durabilidade: 15, Pecas restantes: 1
Linha: 5, Coluna: 3, Durabilidade: 12, Pecas restantes: 1

#MENSAGEM FINAL
A tripulacao finalizou sua jornada
--------Fim da Jornada----------
```

Saída (Modo Análise):

```
====================================
|            MODO ANALISE            |
====================================
Total de chamadas recursivas: 89
Profundidade maxima de recursao: 15
====================================
```

4.3 Arquivo de Teste 3: Coletando Todas as Peças

Entrada (`mapa3.txt`)**:**

```
50 4 12
6 9
X-P-+---+
|.......|
P-+.+-P-+
|.|.|...|
+-+-+.F-+
|.......|
+-P-----+
```

Resultado: A tripulação consegue coletar as 4 peças antes de chegar ao destino, resultando na mensagem “A jornada sera finalizada sem mais desafios”.

Métricas (Modo Análise):

- Chamadas recursivas: 1.523
- Profundidade máxima: 28

4.4 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, o algoritmo encontra um caminho válido respeitando todas as restrições
3. O sistema de coleta de peças funciona adequadamente, aumentando a durabilidade quando necessário
4. A detecção de coleta das 4 peças funciona, alterando o comportamento da nave
5. O modo de análise fornece informações valiosas sobre o desempenho do algoritmo

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

5 Decisões de Implementação

5.1 Linguagem e Estrutura do Código

Optamos por separar o código em módulos distintos:

- `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

Esta modularização facilita manutenção, teste e compreensão do código.

5.2 Ordem de Exploração dos Movimentos

Definimos a ordem de exploração como:

1. Para movimentos verticais: baixo, depois cima
2. Para movimentos horizontais: direita, depois esquerda

Esta ordem foi escolhida arbitrariamente, mas é consistente em toda a execução. Ordens diferentes podem resultar em caminhos diferentes (mas igualmente válidos).

5.3 Tratamento de Erros

Implementamos verificações robustas em todas as operações críticas:

- Verificação de abertura de arquivo
- Validação de alocação de memória
- Verificação de formato do arquivo de entrada
- Detecção de ausência do ponto inicial X

Quando um erro é detectado, o programa exibe mensagem informativa e encerra graciosamente, liberando toda memória alocada.

5.4 Gerenciamento de Memória

Todas as alocações dinâmicas são devidamente liberadas na função `liberarMapa()`. Esta função é chamada mesmo em casos de erro durante o carregamento do mapa, garantindo que não há vazamentos de memória.

5.5 Modo de Análise

Escolhemos implementar o modo de análise via compilação condicional ao invés de uma opção em tempo de execução porque:

- Elimina overhead de verificações desnecessárias no modo normal
- Simplifica a lógica do código
- É transparente para o usuário através do Makefile

5.6 Limitações Conhecidas

Funciona:

- Leitura de mapas de tamanhos variados
- Busca de caminhos respeitando todas as restrições
- Coleta de peças e gerenciamento de durabilidade
- Detecção de impossibilidade de solução
- Modo de análise com métricas de recursão

Não implementado:

- Otimizações para encontrar o caminho mais curto ou mais eficiente
- Interface gráfica para visualização do mapa
- Gerador automático de arquivos de teste (tarefa extra não realizada)
- Múltiplas soluções (o algoritmo para na primeira solução encontrada)

5.7 Decisão sobre Células Tipo P e X

Uma decisão importante foi permitir que células com peças (P) e o ponto inicial (X) aceitem movimentos em qualquer direção (horizontal e vertical). Isso proporciona mais flexibilidade na navegação e reflete a ideia de que estas são posições especiais no mapa.

6 Conclusão

Este trabalho proporcionou experiência prática valiosa na implementação e análise de algoritmos de backtracking. Os principais aprendizados incluem:

1. **Compreensão profunda de backtracking**: A implementação reforçou o entendimento de como o algoritmo explora sistematicamente o espaço de soluções e retrocede quando necessário.
2. **Gerenciamento de memória em C**: O trabalho exigiu uso cuidadoso de alocação dinâmica e liberação de memória, habilidades fundamentais para programação em C.
3. **Modularização e organização de código**: A divisão em módulos facilitou desenvolvimento, teste e manutenção do programa.
4. **Análise de complexidade prática**: O modo de análise permitiu observar concretamente como o número de chamadas recursivas varia com a complexidade do problema.
5. **Trabalho em equipe**: O desenvolvimento colaborativo ensinou coordenação e divisão de tarefas entre membros do grupo.

6.1 Resultados Alcançados

O programa implementado atende completamente à especificação do trabalho:

- Implementa backtracking conforme exigido
- Lê arquivos de entrada no formato especificado
- Gerencia corretamente durabilidade e coleta de peças
- Fornece saídas no formato solicitado
- Inclui modo de análise funcional
- Permite execuções consecutivas com diferentes arquivos

6.2 Possíveis Melhorias Futuras

Para trabalhos futuros, seria interessante:

- Implementar otimizações para reduzir chamadas recursivas desnecessárias

- Adicionar heurísticas para guiar a busca de forma mais eficiente
- Desenvolver visualização gráfica do mapa e do caminho percorrido
- Implementar as tarefas extras sugeridas na especificação
- Comparar desempenho com outras estratégias de busca (BFS, DFS iterativo, A*)

7 Referências

1. CORMEN, Thomas H. et al. **Algoritmos: Teoria e Prática**. 3ª ed. Rio de Janeiro: Elsevier, 2012. Capítulo sobre Backtracking.

2. BARBOSA, Daniel Mendes. **Slides da disciplina Projeto e Análise de Algoritmos**. UFV Campus Florestal, 2025.

3. **Documentação da Linguagem C**: cppreference.com - Referência para funções padrão da linguagem C.

4. ZIVIANI, Nivio. **Projeto de Algoritmos com Implementações em Pascal e C**. 3ª ed. São Paulo: Cengage Learning, 2011.

5. **GNU Make Manual**: Documentação oficial do Make para construção do Makefile. Disponível em: https://www.gnu.org/software/make/manual/

6. SEDGEWICK, Robert; WAYNE, Kevin. **Algorithms**. 4ª ed. Addison-Wesley, 2011. Capítulo sobre algoritmos de busca e backtracking.