Relatório do Trabalho Prático I (TP I)

Alocação Dinâmica, TAD e Recursão

Nome dos alunos:
João Pedro Seabra Nogueira e Maurício de Oliveira Santos Rodrigues

Professora: Karla A S Joriatti

21 de novembro de 2025

Resumo

Este relatório descreve a implementação de um programa em C para resolver o problema da satisfatibilidade booleana para fórmulas na forma 3-CNF (cada cláusula com exatamente 3 variáveis). São apresentados: a estrutura de dados (TAD) para Formula e Clausula, as decisões de projeto, instruções para compilar e executar, códigos-fonte utilizados e uma análise de resultados e uso de memória.

Sumário

1 Implementação

1.1 Visão geral

O programa implementado segue o enunciado do TP: a entrada descreve n (número de variáveis) e m (número de cláusulas), cada cláusula com 3 inteiros representando variáveis ou suas negações. O objetivo é encontrar uma valoração (atribuição TRUE/FALSE) para as n variáveis que torne a fórmula verdadeira, ou declarar que a fórmula é insatisfatível. O TAD `Formula` e o TAD `Clausula` são implementados em arquivos separados.

1.2 Arquitetura e TADs

Clausula Representada por uma struct que guarda um vetor dinâmico de inteiros de tamanho 3 (o valor negativo indica negação).
Operações principais:

- `criaClausula(int *variaveis, int numVariaveis)`: aloca e copia o vetor de inteiros.
- `destroiClausula(Clausula *c)`: libera a memória.
- `tostringClausula(...)`: imprime a cláusula no formato (a v b v c).
- `testaClausula(...)`: verifica se a cláusula é satisfeita dada uma valoração.

Formula Representada por uma struct com campos:

- vetor dinâmico de ponteiros para Clausula (as cláusulas da fórmula);
- número de cláusulas e variáveis (inteiros);
- lista de variáveis (char[]) para impressão das letras (a,b,c...).

Operações principais:

- `criaFormula(int numVariaveis, int numClausulas)`: aloca a estrutura e o espaço para cláusulas;
- `adicionaClausula(Formula*, int index, int *var,...)`: cria a cláusula e atualiza lista de variáveis;
- `imprimeFormula(Formula*, int tamClausula)`: imprime no formato 3-CNF exigido;
- `solucionaFormula(...)`: interface que chama a função recursiva de backtracking `solucaoFormula`.
- `destroiFormula(Formula*)`: libera toda a memória alocada.

1.3 Decisões de projeto e detalhes relevantes

- As cláusulas são armazenadas como objetos do TAD `Clausula` para melhor modularidade.
- A função de busca `solucaoFormula` é recursiva e implementa o algoritmo de backtracking, que explora exaustivamente o espaço de busca de $\mathbf{2^n}$ valorações. O processo funciona da esquerda para a direita (variável 0 até $n-1$), atribuindo TRUE e, em seguida, FALSE a cada variável. A busca é interrompida (retorna true) imediatamente após a primeira valoração satisfatória ser encontrada, ou encerra (retorna false) se todas as $\mathbf{2^n}$ combinações forem testadas sem sucesso.
- Todo espaço alocado no heap é liberado nas funções `destroiClausula` e `destroiFormula`.
- O código está modularizado em: `tp.c`, `formula.c`, `clausula.c` e os respectivos headers (`formula.h`, `clausula.h`).

2 Implementação

2.1 Visão geral

O programa implementado tem como objetivo verificar a satisfatibilidade de uma fórmula booleana na forma normal conjuntiva com três literais por cláusula (3-CNF). A entrada é composta pelo número de variáveis e número de cláusulas, seguidos das cláusulas propriamente ditas, onde cada número representa uma variável (positiva) ou sua negação (negativa). O sistema testa todas as possíveis valorações das variáveis até encontrar uma combinação que satisfaça a fórmula ou determine que ela é insatisfatível.

2.2 Arquitetura geral do programa

O código foi dividido em três módulos principais:

- **tp.c:** módulo principal responsável pela interação com o usuário, leitura dos dados e execução das funções do TAD `Formula`.
- **clausula.c:** implementação das funções responsáveis pela criação, manipulação e liberação de cláusulas individuais.
- **formula.c:** implementação das operações sobre o conjunto de cláusulas (fórmula), além da lógica de busca recursiva por uma valoração satisfatória.

2.3 Estrutura de dados e funções principais

TAD Clausula Representa uma cláusula lógica com três variáveis (ou suas negações). Internamente, contém um vetor dinâmico de inteiros que armazena os literais. As principais operações são:

- `criaClausula(int *variaveis, int numVariaveis)`: aloca e inicializa uma nova cláusula com as variáveis informadas.
- `testaClausula(...)`: avalia se a cláusula é satisfeita para uma dada valoração booleana das variáveis.

- `tostringClausula(...)`: gera a representação textual da cláusula, no formato $(a \vee \neg b \vee c)$.
- `destroiClausula(...)`: libera a memória associada à cláusula.

TAD Formula Representa a expressão completa composta por várias cláusulas. Armazena o número de variáveis, o número de cláusulas e um vetor dinâmico de ponteiros para `Clausula`. As funções principais incluem:

- `criaFormula(int numVariaveis, int numClausulas)`: aloca a estrutura principal e inicializa os vetores internos.
- `adicionaClausula(...)`: adiciona uma nova cláusula ao vetor de cláusulas da fórmula.
- `testaFormula(...)`: verifica se a fórmula é satisfeita para uma valoração específica.
- `solucionaFormula(...)`: função que realiza a recursão de busca pela primeira solução possível.
- `destroiFormula(...)`: libera toda a memória alocada pela estrutura.

2.4 Função recursiva de busca de solução

A função `solucaoFormula()` realiza uma busca exaustiva sobre todas as combinações possíveis de valores lógicos para as variáveis, utilizando uma abordagem recursiva de backtracking.

Seu funcionamento pode ser resumido da seguinte forma:

1. Gera uma combinação de valores booleanos (inicialmente todas verdadeiras).
2. Testa se a fórmula é satisfeita com essa combinação.
3. Caso não seja, inverte o valor da variável atual (de acordo com um índice, que é atualizado a cada chamada recursiva), gerando a próxima combinação.
4. Repete o processo até encontrar uma solução válida ou até testar todas as 2^n combinações possíveis.

2.5 Tratamento de memória e modularização

Todas as estruturas dinâmicas são alocadas explicitamente via `malloc()` e liberadas pelas funções destrutoras correspondentes. A modularização entre os arquivos permite testar e depurar separadamente os componentes:

- `tp.c` executa apenas a lógica de interação e controle;
- `clausula.c` lida com os detalhes de cada cláusula;
- `formula.c` implementa a manipulação da estrutura principal e a rotina de resolução recursiva.

2.6 Compilação e execução

O programa pode ser compilado no terminal com:

```
$ gcc -c formula.c -Wall
$ gcc -c clausula.c -Wall
$ gcc -c tp.c -Wall
$ gcc *.o -o exe
```

Ou pode ser compilado diretamente com:

```
$ gcc tp.c formula.c clausula.c -g -o exe -Wall
```

E executado via:

```
$ ./exe < entrada.txt
```

Onde o arquivo `entrada.txt` contém o número de variáveis, o número de cláusulas e as cláusulas propriamente ditas.

3 Impressões gerais

Durante a implementação, foram adotadas práticas de modularização e verificação de erros nas alocações. O uso de TADs ajuda na manutenção e facilita testes unitários nas funções de manipulação de cláusulas e fórmulas.

4 Análise de Complexidade

4.1 Tempo

Em realação à tempo, a abordagem de backtracking tem custo no pior caso de $O(2^n)$, onde n é o número de variáveis.

4.2 Espaço

A complexidade em relação à espaço será O(n), pois é determinada pelo número de cláusulas inseridas pelo usuário.

5 Conclusão

O programa atende aos requisitos do enunciado: implementa o TAD Formula, armazena cláusulas de 3 literais, imprime a fórmula no formato pedido, e encontra (quando existe) uma valoração que satisfaz a fórmula. A implementação foi modularizada e inclui rotinas para liberar toda memória alocada.