Trabalho Prático I: O Problema da medição de Rick Sanchez

Eduardo de Paiva Dias
2018126657

Belo Horizonte – MG – Brasil

Introdução

O objetivo deste trabalho é utilizar a estrutura de dados lista para implementar um programa que fornece a quantidade de operações mínimas para medir uma determinada quantidade de mililitros com os recipientes usados. É dividido em algumas partes como:

- Ler a medida de recipientes usado no laboratório.
- Ler a informação de um recipiente quebrado e retirá-lo.
- Ler a medição que Rick quer.
- Informar o número de operações mínima para medir a quantidade exata que Rick quer.

Implementação

Funcionamento geral do programa:

A parte principal do programa consiste em gerar a soma de todas as permutações possíveis de um determinado o vetor começando da quantidade mínimas de operações e aumentar até achar o valor requisitado por Rick

As possíveis entradas para o funcionamento do programa seria:

A entrada do programa consiste em um número e um char:

- (Valor) i : para inserir um recipiente com o valor definido.
- (Valor) r : para retirar um recipiente com o valor definido
- (Valor) p : para medir a quantidade de operações mínimas com esse valor definido.

Funcionamento das funções:

TipoItem, TipoCelula, TipoLista, TipoApontador : Estruturas usadas na criação da lista encadeada.
FLVazia() : Cria uma lista vazia.
Vazia() : Verifica se a lista está vazia.
Insere() : Insere um elemento na última posição da lista.

Imprime() : Imprime a soma dos valores da permutação e a quantidade minima de operações para se chegar nesse valor, onde tudo está contido em uma lista.
gerarPermutacao() : Essa é a parte principal do programa, gera todas as permutações possíveis e soma seus valores e logo depois insere esses valores em uma lista, logo depois verifica se o valor foi achado, se não, entra na função novamente sempre aumentando o tamanho. Contudo, uma variável faz toda a parte principal que gera todas as permutações possíveis, que é a controle, ela fica aumentando seu valor e funciona como um vetor de posição para ser colocado em uma variável auxiliar e assim somar seus valores e inserir na lista.
removerRecipiente() : Remove um recipiente do vetor e "puxa" todas as posições subsequentes.
validarpermutacao() : Verifica se o valor que rick pediu foi encontrado, se não aumenta a posição em que se pode fazer a permutação.
verificalguais() : Verifica se tem valor de permutação igual na lista para ver se vai ser inserido ou não.
verificaResposta() : Verifica se a resposta foi achada.
verificaSeRecipienteExisteParaRetirar() : Verifica se o recipiente existe antes de retirá-lo.
main() : Contém a chamada das funções de validarPermutacao() e removerRecipiente() e ainda insere os recipientes no vetor.

Instruções de compilação e execução

O programa foi desenvolvido no sistema operacional windows na linguagem C e compilado pelo ambiente de desenvolvimento integrado de código aberto e multiplataforma livre Code::Blocks. Contudo para compilar o programa no linux basta escrever no terminal o seguinte comando:

```
gcc trabalhopratico.c -o trabalhopratico
```

E para executar o programa podemos escrever no terminal:

```
./trabalhopratico
```

Ou usar um dos casos de teste:

```
./trabalhopratico < 00.in > 00.out
```

sendo 00.in o arquivo de entrada e 00.out o arquivo gerado de saída.

Análise de complexidade

A **complexidade de tempo** para a inserção de um recipiente é **O(1)**, pois se insere diretamente na ultima posição vazia.

A **complexidade de tempo** para remoção de um recipiente é **O(n)**, pois tem que percorrer o vetor inteiro para procurar a posição e assim remover a posição e arrumar as posições subsequentes.

A **complexidade de tempo** para medir a quantidade ml é **O(n^r)** para cada vez em que se entra na função gerarPermutação(), começando r=1. Em casos muito grandes isso pode ser demorado.

Análise de complexidade de espaço é a quantidade de espaço necessária para alocar os itens da lista e os recipientes que é **O(k)** e **O(n)**, respectivamente.

Conclusão

O trabalho foi concluído com êxito, tanto em questão da construção da lista como em gerar todas as permutações possíveis, entretanto em alguns casos de teste pode ser demorado para concluí-lo, pois o algoritmo não é eficiente em casos muito grandes.

Bibliografia

CHAIMOWICZ, Luiz; PRATES, Raquel. Listas Lineares. Belo Horizonte, 2019/1.