UFMG - ICEx

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TP 1
TADs

Aluno: Gabriel Lopes Machado
Mat: 2018019516

Turma: TF
Professor: Carlos Henrique de Carvalho Teixeira

INTRODUÇÃO

Todo o programa foi desenvolvendo explorando-se de orientação a objetos e boas práticas de programação, além de sempre optar pela versão mais recursiva e “limpa” do algoritmo. Portanto, foi utilizada a linguagem C++ para a codificação do problema proposto.

O programa consiste da interação de objetos de algumas classes e são elas:

Registro -> simplesmente consiste em uma classe que possui um inteiro como atributo.

Recipiente -> É a composição de um registro em uma estrutura capaz de ser uma célula de uma lista/fila

Lista -> representa uma TAD do tipo lista com alocação de memória dinâmica, que implementa as funções necessárias para o problema em questão.

Fila -> Uma classe que filha de Lista que aproveita parte de seu código e modifica seu comportamento para ser uma Fila.

Árvore -> Uma classe com todos seus membros estáticos que contem o algoritmo para realizar a busca pelo número mínimo de operações necessárias.

IMPLEMENTAÇÃO

- Começando sobre as classes Fila e Lista, ambas possuem o método insere (void insere(ml, const int&);) que possui o mesmo comportamento para ambas as classes, logo, a principal diferença de uma para outra fica na forma como elas vão retirar o elemento, no caso da Lista que é a responsável por guardar todos os recipientes disponíveis no laboratório só se faz necessário a remoção quando uma entrada do usuário assim determina (void remove(ml);), que retira um item da Lista e ajusta corretamente os ponteiros. Já na Fila a remoção (Recipiente * remove();) de um item se faz por meio da filosofia FIFO (Frist In, Last Out), esse tipo de estrutura representa o comportamento de uma pesquisa BFS (Breadth Firt Search) em uma esturura em árvore.

- Agora sobre o algorítimo de busca que foi implementado, seu comportamento conciste em percorrer a lista que contem os recipientes disponíveis e realizar a adição e subtrações dos reultados maiores que zero sobre o último item que retirado da frente da Fila.

- O fomato da entrada segue o templete proposto, logo uma entrada "50 i\n30 i\n80 p" retrna uma saída "2\n" onde o '\n' representa a quebra de linha (a tecla "enter" do tecclado pelo stdin).
- O sistema operácional utilizado ->
  - Kernel: 4.19.69-1-MANJARO x86_64
  - bits: 64
  - Desktop: Xfce 4.14.1
  - Distro: Manjaro Linux
- Da versão do Make:
  - GNU Make 4.2.1
    Built for x86_64-pc-linux-gnu
- Da versão do compilador utilizado:
  - gcc version 9.1.0 (GCC)

COMPILAÇÃO

Para compilar basta abrir um terminal linux na raíx do projeto (pasta "Gabriel_Machado") e executar o comando "make tp1" e um executável tp1 será gerado, ai pode ser utilizado "./tp1".

ANALISE DE COMPLEXIDADE

Essa análise foi feita em cima do algoritmo de BFS, que foi a base para a implementação, logo a complexidade de tempo e espaço são equivalentes. (obs: fiz em latex)

Para o pior caso temos que ambas as operações de adição e subtração são computadas, logo, a fila que apresenta todas as possibilidades possíveis tem tamanho 2 vezes maior que:

$$\left.\begin{array}{r} T(n) = T(n+1) + 2b^n \\ T(n+1) = T(n+2) + 2b^{n+1} \\ T(n+2) = T(n+3) + 2b^{n+2} \\ \vdots \\ T(d) = 2b^d \end{array}\right\} \quad 2b^1 + 2b^2 + 2b^3 + ... + 2b^d = 2b^{d+1} - 1$$

No melhor caso o número avaliado é menor do que qualquer recipiente disponível:

$$\left.\begin{array}{r} T(n) = T(n+1) + b^n \\ T(n+1) = T(n+2) + b^{n+1} \\ T(n+2) = T(n+3) + b^{n+2} \\ \vdots \\ T(d) = b^d \end{array}\right\} \quad b^1 + b^2 + b^3 + ... + b^d = b^{d+1} - 1$$

De fato não sei computar a complexidade média, pois não sei avaliar quantas vezes um número vai ser maior ou menor que os presentes na lista.

O que sei dizer com precisão é que a ordem de complexidade é sempre a mesma e do tipo:

$$O(b^d)$$

CONCLUSÕES

Concluo por meio desse trabalho que a abstração de um problema por meio de estruturas de dados é capaz de resolver em alta performance problemas que tem uma grande complexidade. O maior desafio enfrentado definitivamente foi ao tentar diminuir a complexidade de espaço causada pela utilização do algoritmo em sua forma recursiva, pois a pilha de execução do sistema operacional parece que não foi grande o suficiente para executar todos os testes em tempo hábil, mesmo que manualmente todos tenham passados.

O maior aprendizado prático fica na modelagem do problema a partir de TADs e a experiência na criação de estruturas de dados mais robustas para solucionar problemas reais.