O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

Introdução

O trabalho prático consiste no cálculo do mínimo de operações de medição necessárias para se obter uma certa medida, se baseando nos frascos disponíveis no laboratório. Além disso, durante a execução podem ser removidos ou inseridos frascos.

Desse modo, o programa usa duas listas encadeadas, uma contém os frascos do laboratório e a outra as medidas feitas com os então frascos.

A medida inicial é zero e a partir dela é feito a soma e subtração do seu valor com a medida do frasco atual, de maneira que caso o resultado seja maior do que zero, esse valor é inserido na lista de medidas. Após percorrer toda a lista de frascos – e fazer as operações – é seguido para próxima medida, enquanto não é encontrado a medida desejada.

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

Implementação

No programa existem três métodos principais, sendo eles ***insert***, ***remove*** e ***measurement*** – todas pertencentes a classe Lab. O compilador é o padrão do linux, ***g++***.

Insert

```
void Lab::insert(int ml, int operations)
{
  Bottle* newBottle = new Bottle();
  newBottle->setMl(ml);
  newBottle->setOperations(operations);
  newBottle->next = nullptr;

  if(head == nullptr)
  {
    head = newBottle;
    tail = newBottle;
  }

  else
  {
    tail->next = newBottle;
    tail = newBottle;
  }
}
```

O método ***insert*** recebe como parâmetro os inteiros ***ml*** e ***operations***, que serão inseridos como informações do novo nó. O ***ml*** pode ser tanto o valor de ml do frasco ou a medicação feita, enquanto ***operations*** se refere quantas operações foram necessárias para chegar a essa medida de ml, a função é do tipo ***void***. O insert é utilizado tanto na inserção de novos fracos no laboratório quanto na inserção de novas medidas.

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

A inserção é sempre feita no final da lista - caso o ***head*** da lista esteja vazio, o nó será tanto o **head** quanto o ***tail***, caso não, o nó é inserido no final da lista e se torna o novo ***tail*** -, pois como será explicado na função measurement, é necessário para o funcionamento do código.

Remove

```
void Lab::remove(int ml)
{
  Bottle* previous = nullptr;
  Bottle* current = nullptr;

  current = this->head;

  if (current->getMl() == ml)
    this->head = this->head->next;

  else
  {
    while (current->getMl() != ml)
    {
      previous = current;
      current = current->next;
    }

  previous->next = current->next;
  }

  if(current == this->tail)
    this->tail = previous;

  delete current;
}
```

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

O método ***remove*** recebe como parâmetro o inteiro ***ml***, que corresponde ao ml do nó que deverá ser removido da lista do laboratório, sendo utilizado apenas pela lista de frascos, a função é do tipo ***void***.

Primeiro, é verificado se o frasco a ser removido é o ***head*** da lista, caso sim, o ***head*** da lista será alterado para o nó seguinte a ele, caso não, a lista é percorrida, utilizando dois nós - o nó ***previous*** sempre aponta para o nó anterior ao nó ***current*** -, até o current ser o nó que possui o valor de ml igual ao que se deseja remover, feito isso, o próximo nó do ***previous*** passa a ser o próximo nó do ***current***, de forma a remover a linkagem do nó ***current*** com a lista.

Por último, é verificado se o nó ***current*** e a o ***tail*** da lista apontam para o mesmo nó - o ***current*** é o último nó da lista -, caso afirmativo, alteramos o apontamento do ***tail*** para o ***previous*** - o nó anterior ao ***current*** -, com isso qualquer linkagem do nó ***current*** com a lista de frascos é completamente removida e então, deletamos o mesmo.

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

Measument

```
int Lab::measurement(int ml)
{
  Lab* measures = new Lab();
  int aux1 = 0;
  int aux2 = 0;
  int result = 0;

  measures->insert(0, 0);

  Bottle* currentBottle = this->head;
  Bottle* currentMeasure = measures->head;

  while(aux1 != ml && aux2 != ml)
  {
    aux1 = currentMeasure->getMl() + currentBottle->getMl();
    aux2 = currentMeasure->getMl() - currentBottle->getMl();

    if(aux1 > 0)
      measures->insert(aux1, currentMeasure->getOperations() + 1);

    if(aux2 > 0)
      measures->insert(aux2, currentMeasure->getOperations() + 1);

    currentBottle = currentBottle->next;
    result = currentMeasure->getOperations() + 1;

    if(currentBottle == nullptr)
    {
      currentBottle = this->head;
      currentMeasure = currentMeasure->next;
    }
  }
  delete measures;
  return result;
}
```

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

O método ***measurement*** recebe como parâmetro o inteiro ***ml***, que corresponde ao ml que se deseja saber quantas operações são necessárias para chegar a ele, a função é do tipo ***int*** - retorna o número de operações.

A função utiliza 6 variáveis, ***measures*** - a lista de medições que serão feitas -, ***aux1*** - auxiliar que irá corresponder a soma da medição com o frasco atual-, ***aux2*** - auxiliar que irá corresponder a subtração da medição com o frasco atual-. ***result*** - valor final do número de operações -, ***currentBottle*** - nó correspondente ao frasco atual -, ***currentMeasure*** - nó correspondente a medida atual.

É inserido a medição zero na lista de medições (***measures***) para inicializar ela e então ***currentBottle*** e ***currentMeasure*** passam a apontar para o ***head*** das suas respectivas listas, de frasco e de medições.

Feito isso, é iniciado um **while** que finaliza apenas quando o valor de pelo menos uma das auxiliares (***aux1*** ou ***aux2***) for igual a medida desejada.
O ***while*** faz as medições, utilizando ***aux1*** e ***aux***2, como explicado acima e então, é verificado se seus valores são positivos e maior que zero, caso sim, a auxiliar respectiva é inserido na lista de medições como uma nova medida, após isso, muda-se de frasco e é armazenado o valor de operações para chegar a essas medidas - número de operações da medida anterior somado 1 -, caso nó de frasco atual é vazio (***nullptr***), quer dizer que foram feitas todas as operações com a medida atual, então é voltado o nó de frascos para o início (***head***) e segue-se para o próximo nó da lista de medições.

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

Ao sair do ***while***, a lista de medições (***measures***) é deletada e retornamos o resultado (**result**) - o número final de operações.
Devido a isso, a inserção dos fracos / medições deve ser sempre no final, pois a lista de medições é sempre percorrida do seu nó anterior em direção ao próximo, sem nunca voltar ao nó inicial (***head***).

Instruções de compilação e execução

Compilação

Acesse pelo terminal a pasta lucas_marinho e digite o comando ***make*** para compilar os arquivos, será gerado um arquivo com o nome ***tp1***.

Execução

Acesse pelo terminal a pasta lucas_marinho e digite o comando ***make test*** para executar os testes e ***./tp1*** para executar o trabalho.

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

Análise de Complexidade

Seja ***n*** o número de operações necessárias para chegar a medida desejada.

A complexidade de tempo do ***insert***, no pior e melhor caso, é :

f(n) = O(1), em que ***n ≥ 1***

Complexidade espacial do ***insert*** , no pior e melhor caso, é :

f(n) = O(1), em que ***n ≥ 1***

A complexidade de tempo do ***measurement*** é :

f(n) = O(n^n ***)***, para o pior caso, em que ***n > 1***

f(n) = O(1), para o melhor caso, em que ***n = 1***

A complexidade espacial do **measurement** é

f(n) = O(n^n ***)***, para o pior caso, em que ***n > 1***

f(n) = O(n), para o melhor caso, em que ***n = 1***

O problema da medição de Rick Sanchez

Lucas Nascimento Marinho

2018088100

Conclusão

Durante o trabalho foi necessário fazer uma rápida revisão sobre como implementar uma lista encadeada, de início foi implementado utilizando uma lista duplamente encadeada, por ser mais fácil fazer operações de inserção, remoção e percorrer a mesma, porém a um custo de um ponteiro a mais e por isso foi decidido mudar para uma lista singularmente encadeada.

O foco principal foi nos métodos de medição, remoção, sendo os que levaram mais esforços.

O primeiro devido a função de inserção, anteriormente, não receber como parâmetro o número de operações realizadas, impedindo o controle de quantas operações foram feitas.

O segundo devido ao fato de que a função deveria abranger três situações de remoção - no início, em dada posição e no final -, no final das contas, o método implementa uma variação da remoção de um nó dado uma posição, porém, nesse caso, é levado em consideração se essa posição é o início ou o final da lista.

Resolvido essas questões, o restante da implementação foi feita sem demais dificuldades, foi tentando alterar todas as variáveis do tipo ***int*** para o tipo ***short*** por questões de uso de memória, no entanto, ao executar o programa, nenhum resultado era retornado. Após diversas tentativas de solucionar, foi decidido manter o tipo ***int***.

Bibliografia

Durante a programação foi apenas consultado um site : codementor[1], para revisar sobre a implementação de listas encadeadas.

---

[1] https://www.codementor.io/codementorteam/a-comprehensive-guide-to-implementation-of-singly-linked-list-using-c_plus_plus-ondlm5azr