Davi Mantovani Rici

Teste técnico ATECH

Repositório com desenvolvimento completo:

git clone git@github.com:daviricci/atech\_cpp.git

1. A seguir estão comentadas ambas as funções comparando suas implementações.

int fibonacci\_01::fib(int n) {  
 /\*  
 \* Aqui utiliza-se função recursiva, em que são gerados "espaços" dentro de "espaços" análogo a  
 \* uma busca em profundidade. Quando a codição n<=1 é satisfeita, a função chamada dentro do último  
 \* "espaço" retorna um valor e assim as demais chamadas retornam valores sucessivamente obedecendo  
 \* a soma fib(n-1) + fib(n-2). o Valor final retornado é o n-ésimo termo da sequência de fibonacci.  
 \*/  
 if (n<=1){  
 return n;  
 }  
 return fib(n-1) + fib(n-2);  
}  
int fibonacci\_02::fib(int n) {  
 //int f[] = new int[n+2]; declaração original incorreta  
 /\*  
 \* Nesse caso, primero são gerados todos os valores da sequência de fibonacci até o n-ésimo item,  
 \* retornando-o no final  
 \*/  
 int \*f = new int(n+2);  
 int i;  
 f[0] = 0;// o termo zero é atribuído  
 f[1] = 1;// o primeiro termo é atribuído  
 for (i=2;i<=n;i++){  
 f[i] = f[i-1] + f[i-2];// com os primeiros termos inicializados, os demais são gerados até o n-ésimo  
 }  
 return f[n]; //retorna o n-ésimo termo

1. Função que retorna string sem caracteres duplicados e estimativa da complexidade assintótica

#include <map>

#include <algorithm>

#include <iostream>

#include <string>

std::string remove\_multiple\_chars(const std::string& in){  
 std::string out = in;  
 const char \*p\_s = out.c\_str();  
 std::map<char,int> m;  
 int i=0;  
 //N  
 while (i<out.length()) { //1  
 if (m.find(p\_s[i])!=m.end()){ // (log(i)), i<K, K é a quantindade de caracteres não repetidos da string   
 out.erase(i,1);// 1  
 }  
 else{  
 m.insert(std::pair<char, int>(p\_s[i],1)); // 1  
 i++;  
 }  
 }  
 return out;  
}  
/\*  
 \* Para determinação da complexidade assintótica, foi considerado  
 \* a complexidade assintótica do método find da classe std::map,  
 \* uma vez que esse recurso foi utilizado.  
 \* A função foi feita de tal forma que uma única varredura pela  
 \* string fosse o suficiente para encontrar todos os caracteres duplicados.  
 \* Para isso foi utilizado a estrutura std::map para armazenar novos  
 \* caracteres encontrados e para verificar a ocorrência de repetições.  
 \* Caso algum caractere já tiver sido armazenado na estrutura std::map,   
 \* onde isso é verificado pela função find da classe std::map, esse é  
 \* removido da string analisada.  
 \* Ao chegar no último caractere da string, ter-se-á obtido uma string  
 \* sem caracteres repetidos.  
 \* Assim temos:  
 \* N -> número de caracteres totais da string  
 \* K -> número de caracteres não repetidos da string  
 \* W -> número de execuções por iteração  
 \* onde K<=W  
 \* Portanto a complexidade assintótica pode ser estimada da seguinte forma:  
 \*   
 \* W  
 \* +  
 \* W + log(1) // existe um caractere na estrutura que se busca se o caractere analisado já foi ou não encontrado  
 \* +  
 \* W + log(2) // existem dois caracteres na estrutura que se busca se o caractere analisado já foi ou não encontrado  
 \* +  
 \* .  
 \* .  
 \* .  
 \* +  
 \* W + log(K) // existem K caracteres na estrutura que se busca se o caractere analisado já foi ou não encontrado  
 \* +  
 \* W + log(K)  
 \* .  
 \* .  
 \* .  
 \* +  
 \* W + log(K)  
 \*   
 \* Assim podemos dizer que a complexidade assintótica será de O(N\*log(N)).  
 \* \*/

1. Foram criadas as classes bases para implementação da arquitetura Publisher Subscriber. Essas são: Message, Subscriber, Broker e Publisher. Então a partir dessas foi proposto a classe User. A classe User é utilizada para instanciar os terminais do Chat Server, tal que de qualquer terminal é possível enviar e receber mensagens. Para isso, todo User contém objetos das classes Subscriber e Publisher. A classe Broker é utilizada como o servidor de toda operação e troca de mensagens entre Users. Cada User tem sua lista de contatos (tópicos) dos quais prefere receber mensagens. O desafio seria, usando da arquitetura proposta, selecionar quais User iriam receber as mensagens.

**#include <string>  
#include <vector>  
#include <queue>  
#include <string>  
#include <utility>  
#include <iostream>**

class Message{  
 public:  
 Message(std::string topic, std::string msg);  
 ~Message();  
 std::string get\_msg();  
 std::string get\_topic();  
 private:  
 std::string topic;  
 std::string msg;  
};  
  
class Subscriber{  
 public:  
 Subscriber();  
 ~Subscriber();  
 void show\_message();  
 void add\_topic(std::string topic);  
 std::vector<std::string> topics;  
 std::queue<Message> msgs;  
};  
  
class Broker{  
 public:  
 Broker();  
 ~Broker();  
 void forward();  
 void subscribe(Subscriber \*subscriber);  
 void receive(Message &new\_message);  
  
 private:  
 std::queue<Message> msgs;  
 std::vector<Subscriber\*> subscribers;  
};  
  
class Publisher{  
 public:  
 Publisher();  
 ~Publisher();  
 void send\_msg(Message &new\_message, Broker &my\_server);  
};  
  
class User{  
 /\*  
 \* *Todo usuário é um Publisher e um Subscriber*  
 \* Quando o usuário atuar como Publisher, toda mensagem enviada terá um tópico fixo,  
 \* sendo esse sua assinatura.  
 \* *Todo usuário tem seu vetor de tópicos de seus contatos, quais deseja receber*  
 \* mensagens, quando atuar como Subscriber  
 \*/  
 public:  
 User(std::string topic\_signature, Broker \*broker);  
 ~User();  
 void send\_message(std::string msg);  
 void add\_contact(std::string topic);  
 void show\_message();  
 Subscriber \*get\_subscriber();  
 private:  
 std::string topic\_signature;  
 Broker \*broker;  
 Publisher publisher;  
 Subscriber subscriber;  
};

//--- **#include "core.h"**int main() {  
 core::Broker Server;  
 core::User Davi(**"Davi"**,&Server);  
 core::User Paulo(**"Paulo"**,&Server);  
 core::User Daniel(**"Daniel"**,&Server);  
  
 Server.subscribe(Davi.get\_subscriber());  
 Server.subscribe(Paulo.get\_subscriber());  
 Server.subscribe(Daniel.get\_subscriber());  
  
 Davi.add\_contact(**"Paulo"**);  
 Davi.add\_contact(**"Daniel"**);  
  
 Paulo.add\_contact(**"Davi"**);  
 Paulo.add\_contact(**"Daniel"**);  
  
 Daniel.add\_contact(**"Davi"**);  
 Daniel.add\_contact(**"Paulo"**);  
  
 Davi.send\_message(**"Eu, Davi, estou com covid19"**);  
 Davi.send\_message(**"Já fiz dois exames e ambos deram positivos"**);  
  
 Server.forward();  
 Paulo.show\_message();  
 Daniel.show\_message();  
  
  
 return 0;  
}

Output:

Eu, Davi, estou com covid19

Já fiz dois exames e ambos deram positivos

Eu, Davi, estou com covid19

Já fiz dois exames e ambos deram positivos

---

---