2ªaula prática - Complexidade de algoritmos

Instruções

- Faça download do ficheiro ***aed2122_p02.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***myVector.h, fibonacci.h, cycle.h*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)
- No CLion, abra um ***projeto***, selecionando a pasta que contém os ficheiros do ponto anterior.
- Efetuar "*Load CMake Project*" sobre o ficheiro *CMakeLists.txt*
- Execute o projeto (**Run**)

Nota: O ficheiro *cycle.h* possui funções úteis para a contagem do tempo (ticks). Um exemplo de como usar esta funcionalidade pode ser visto nos testes unitários do exercício 1 a).

1. Considere a classe ***myVector***, que inclui funcionalidades adicionais na estrutura vetor.

```
template <class T>
class MyVector {
    vector<T> v;
public:
    MyVector();
    ~MyVector();
    T max() const;
    bool hasDuplicates(void) const;
};
```

a) Implemente o membro-função:

T MyVector<T>::max() const

Esta função retorna o valor máximo do vetor. Se o vetor estiver vaio, é lançada a exceção *EmptyVector*. A exceção *EmptyVector* já está implementada.

Qual a complexidade temporal e espacial da função *max()*? Comprove empiricamente a complexidade temporal da função.

b) Implemente o membro-função:

bool MyVector<T>::hasDuplicates() const

Esta função verifica se o vetor contém valores repetidos. Retorna *true* se o vetor contém valores repetidos e false se não contém nenhum valor repetido. Não deve ser alterada a ordem relativa dos elementos do vetor.

Qual a complexidade temporal e espacial da função *hasDuplicates()*? Comprove empiricamente a complexidade temporal da função.

c) Implemente o membro-função:

void MyVector<T>::removeDuplicates()

Esta função remove os valores repetidos no vetor. Não deve ser alterada a ordem relativa dos elementos do vetor, sendo mantida a 1ªocorrência do(s) valor(es) repetido(s) e removidas as ocorrências seguintes. Qual a complexidade temporal e espacial da função *removeDuplicates()*?

2. A sequência de Fibonacci é uma sequência de números inteiros, começando por 0 e 1, na qual cada termo subsequente corresponde à soma dos dois anteriores:

0,1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, …

Em termos matemáticos, a sequência é definida como:

$F_n = F_{n-1} + F_{n-2}$, para $n > 1$

sendo $F_0 = 0$ *e* $F_1 = 1$

a) Considere a função que calcula o valor de n-ésimo elemento da sequência de Fibonacci:

unsigned fibonacci_1(unsigned n);

Determine a complexidade temporal e espacial da função *fibonacci_1*. Comprove empiricamente a complexidade temporal da função.

```
unsigned fibonacci_1(unsigned n) {
    unsigned valPrevPrev = 0, valPrev = 1;
    if (n == 0)
        return valPrevPrev;
    if (n == 1)
        return valPrev;
    unsigned val;
    for (unsigned i = 2; i <= n; i ++)
    {
        val = valPrevPrev + valPrev;
        valPrevPrev = valPrev;
        valPrev = val;
    }
    return val;
}
```

b) Considere agora a função que calcula o valor de n-ésimo elemento da sequência de Fibonacci:

unsigned fibonacci_2(unsigned n);

Determine a complexidade temporal e espacial da função *fibonacci_2*. Comprove empiricamente a complexidade temporal da função.

```
unsigned fibonacci_2(unsigned n) {
    if (n <= 1)
        return n;
    return fibonacci_2(n-1) + fibonacci_2(n-2);
}
```

3. Determine a complexidade temporal e espacial do seguinte fragmento de código

a)

```
void funcao1(vector<int> &v) {
   for(int i=1; i< v.size(); i=i*2)
      for(int j=i; j< v.size(); j+=2)
         v[i][j]-=1;
}
```

b)

```
void funcao2(vector<int> &v) {
   for(int i=0; i< v.size(); i++)
      for(int j=0; j< v.size()/2; j++)
         v[i]*=2;
   for(int k=0; k< v.size(); k++)
      v[k]-=1;
}
```