Análise de complexidade

Por que aprender?

- **Habilidade essencial para um desenvolvedor;**
- **Decidir qual algoritmo rodar para cada problema;**
- **Entender o que pode estar causando lentidões na execução do projeto.**

Objetivos da aula

Ao final da aula:

- **Analisar códigos seguindo método experimental;**
- **Analisar códigos seguindo modelo RAM;**
- **Executar as ferramentas Gcov e Gprof e interpretar os resultados.**

Introdução

- É uma ferramenta útil para escolha e/ou desenvolvimento do melhor algoritmo a ser utilizado para resolver determinado problema

Eficiência + Eficácia

Como escolher um algoritmo?

- Tempo de processamento: Um algoritmo que realiza uma tarefa em 10 horas é melhor que outro que realiza em 10 dias

- Quantidade de memória necessária: Um algoritmo que usa 1MB de memória RAM é melhor que outro que usa 1GB

Análise de complexidade

Estudo da eficiência de um código.

- Método experimental
- Modelo RAM
- Análise assintótica

Método experimental

- Testar programas de terceiros
- Varia de máquina pra máquina
- Estuda o tempo que cada código leva para ser executado

Método experimental

- Não é o mais indicado;
- É necessário levar em considerações algumas variáveis na hora de comparar algoritmos experimentalmente: máquinas, compiladores, sistemas e entradas problemáticas

GNU Profiler(gprof)

Uso de *profilers* para analisar o tempo consumido para rodar cada função de um código.

Para saber mais: *Profilers* ou um *profiler* é uma ferramenta que possibilita a coleta de dados e exibição de informação sobre o código compilado/executado. O gprof faz parte da GNU Binary Utility (binutils).

GNU Profiler(gprof)

Para usar o gprof, precisamos seguir os seguintes pontos:

- Ativar a ferramenta de profile no processo de compilação;
- Executar o programa para gerar os profilers;
- Rodar a ferramenta de gprof no arquivo de profiling gerado no passo anterior.

gprof no Windows

1) Abrir o diretório do programa no terminal
2) Execute os seguintes comandos:

```
g++ -g -pg -o main.exe main.cpp
main.exe
gprof main.exe > log.txt
```

```
-g -> Gerar código com símbolos de debug
-pg -> Profiling das infos
-o -> Especificar forma/nome do arquivo de saida
```

Para saber mais: Link com as flags do g++: https://caiorss.github.io/C-Cpp-Notes/compiler-flags-options.html

gprof no Linux

1) Abrir o diretório do programa no terminal
2) Execute os seguintes commandos:

```
g++ -g -pg -o main.out main.cpp
./main.out
gprof main.out > log.txt
```

GNU Profiler(gprof)

```
Flat profile:


Each sample counts as 0.01 seconds.
%    cumulative self          self   total
time seconds    seconds calls s/call s/call name
33.86 15.52     15.52    1    15.52  15.52  func2
33.82 31.02     15.50    1    15.50  15.50  new_func1
33.29 46.27     15.26    1    15.26  30.75  func1
0.07  46.30     0.03                       main
```

GNU Profiler(gprof)

```
Flat profile:


Each sample counts as 0.01 seconds.
%    cumulative self          self   total
time seconds    seconds calls s/call s/call name
33.86 15.52     15.52    1    15.52  15.52  func2
33.82 31.02     15.50    1    15.50  15.50  new_func1
33.29 46.27     15.26    1    15.26  30.75  func1
0.07  46.30     0.03                       main
```

% -> Porcentagem do tempo total que o programa gastou naquela função;
Cumulative -> Tempo total,em segundos, gastos pelo programa até aquele ponto;
Self -> Tempo total gasto naquela parte específica do programa;
Calls -> Quantas vezes uma função(não main) foi chamada;
Self s/ call -> Tempo total gasto rodando códigos APENAS daquela função, por chamada;
Total s/ call -> Tempo total gasto na função, por chamada;
Name -> Nome da função;

GNU Profiler(gprof)

```
C:\Users\Matheus\Documents\Projects\NovoArquivo.exe
 Inside main  Inside func1  Inside new_func1  Inside func2
O Processo retornou 0   tempo de execução : 11.627 s
Pressione uma tecla para continuar...
```

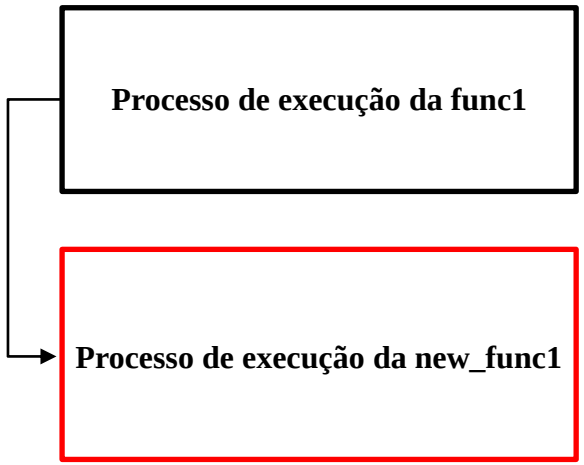
Por que o tempo Total /s call da func1 foi 30.75s ?


```
1  #include <iostream>
2  #include <climits>
3
4  using namespace std;
5
6  void new_func1(void)
7  {
8      cout << " Inside new_func1 ";
9      for(int i = 0; i < INT_MAX; i++);
10     return;
11 };
12
13 void func1(void)
14 {
15     cout << " Inside func1 ";
16     for(int i = 0; i < INT_MAX; i++);
17     new_func1();
18     return;
19 }
20
21 void func2(void)
22 {
23     cout << " Inside func2 ";
24     for(int i = 0; i < INT_MAX; i++);
25     return;
26 }
27
28 int main(void)
29 {
30     cout << " Inside main ";
31     for(int i = 0; i < INT_MAX; i++);
32     func1();
33     func2();
34
35     return 0;
36 }
37
```

GNU Profiler(gprof)

Por que o tempo Total /s call da func1 foi 30.75s ?

Porque é a soma do tempo de execução da func1 e da new_func1




```
6   void new_func1(void)
7   {
8       cout << " Inside new_func1 ";
9       for(int i = 0; i < INT_MAX; i++);
10      return;
11  };
12
13  void func1(void)
14  {
15      cout << " Inside func1 ";
16      for(int i = 0; i < INT_MAX; i++);
17      new_func1();
18      return;
19  }
```

Modelo RAM

- Faz o estudo teórico de cada linha do código;
  - Operações;
  - Acessos de memória;
  - Quantas vezes a linha vai ser executada.

Modelo RAM

- Cada comando tem um “custo”

Operações	Custo
Atribuição: =	1
Aritméticas: +,-,*,/,%	1
Leitura/escrita: cin, cout	1
Comparação: ==, !=, <. >, <=, >=	1
Lógicas: &&, ||	1
Acesso à memória	1
Incremento: i++, ++i,i--, --i	1

Modelo RAM

Exemplo: Vezes rodadas/Custo da linha

```
i = 0;      1 * 1 (atribuição)
x = 2 * n; 1 * 3 (acesso à memória do n,
multiplicação, atribuição)
While(i < x) (2*n + 1) * 3 (acesso à memória do x, acesso à
memória do i, comparação)
{
      a++; (2*n) * 1 (incremento)
      i++; (2*n) * 1 (incremento)
}
T(n) = 1 + 3 + (2n+1)*3 + 2n + 2n
T(n) = 7 + 10n
```

Gcov

- Ferramenta inclusa no GCC(Compilador), feita para ajudar no profiling e teste de cobertura do código.
- Segue o Modelo RAM, tendo como principal característica a indicação de quantas vezes uma linha foi executada.

Gcov (Windows)

1) Para executar

```
g++ -fprofile-arcs -ftest-coverage –o main.exe main.cpp
main.exe
gcov main.cpp
type main.cpp.gcov
```

```
D:\Inatel EAD\Monitorias\Aula1\Gcov>type main.cpp.gcov
        -:    0:Source:main.cpp
        -:    0:Graph:main.gcno
        -:    0:Data:main.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:#include <iostream>
        -:    2:#include <locale>
        -:    3:
        -:    4:using namespace std;
        -:    5:
        1:    6:int main()
        -:    7:{
        1:    8:        setlocale(LC_ALL, "Portuguese");
        1:    9:        char comando = ' ';
        -:   10:
        8:   11:        do
        -:   12:        {
        8:   13:                cin >> comando;
        -:   14:
        8:   15:                switch(comando)
        -:   16:                {
        -:   17:                case 'a':
        3:   18:                        cout << "Tudo que temos de decidir Ú o que fazer com o tempo que nos Ú dado." << endl;
        3:   19:                        break;
        -:   20:                case 'b':
        2:   21:                        cout << "Aþ§es nÒo valerÒo menos porque nÒo foram elogiadas." << endl;
        2:   22:                        break;
        -:   23:                case 'q':
        1:   24:                        break;
        -:   25:                default:
        2:   26:                        cout << "Comando nÒo identificado" << endl;
        -:   27:                        break;
        -:   28:                }
        -:   29:        }
        -:   30:        while(comando != 'q');
        1:   31:        return 0;
        3:   32:}
```

Gcov(Linux)

1) Para executar

```
g++ -fprofile-arcs -ftest-coverage –o main.out main.cpp
./main.out
gcov main.cpp
gedit main.cpp.gcov
```

Dúvidas?

Exercícios

Gcov + custo do programa

```
int n = 8;
int i = 0;
while (i < n)
{
    i = calcula(i);
}
return 0;
```

```
x = 2*i;
cout<<x<<endl;
return (i+1);
```

Exercícios

Gprof + interpretar valores de saída

```
int n = 50000;
recursivo(n);
return 0;
```

```
if(n!=0) recursivo(n-1);
cout<<n<<endl;
```