Relatório da análise de complexidade do algoritmo de estudantes x dormitórios

---

Algoritmo

Função *obterPares*

```
37 func obterPares(estudantesC1, estudantesC2 []estudante, quantidadeDeDormitórios int) ([]par, []estudante) {
36   var pares []par
35   quantidadeMáximaDePares := quantidadeDeDormitórios / 2
34   for i := 0; i < len(estudantesC1); i++ {
33     e1 := estudantesC1[i]
32     var e2 estudante
31     for j := 0; j < len(estudantesC2); j++ {
30       if estudanteJáFoiEscolhido(estudantesC2[j], pares) || sãoIncompatíveis(estudantesC1[i], estudantesC2[j]) {
29         continue
28       }
27       e2 = estudantesC2[j]
26       break
25     }
24     pares = append(pares, par{&e1, &e2})
23     if len(pares) >= quantidadeMáximaDePares {
22       break
21     }
20   }
19   var estudantesSemPar []estudante
18   if len(pares) != quantidadeMáximaDePares {
17     for i := 0; i < len(estudantesC1); i++ {
16       if !estudanteJáFoiEscolhido(estudantesC1[i], pares) {
15         estudantesSemPar = append(estudantesSemPar, estudantesC1[i])
14       }
13     }
12     for i := 0; i < len(estudantesC2); i++ {
11       if !estudanteJáFoiEscolhido(estudantesC2[i], pares) {
10         estudantesSemPar = append(estudantesSemPar, estudantesC2[i])
 9       }
 8     }
 7   }
 6   return pares, estudantesSemPar
 5 }
```

A função instancia um array de pares e obtém a quantidade máxima de pares aceita (metade da quantidade de dormitórios). Em seguida, percorre-se cada um dos estudantes de C1, a fim de alocá-los em pares. Durante cada iteração é feita uma análise de cada estudante de C2, a fim de ver se ele já foi selecionado para um par ou se é incompatível com o estudante de C1. Se alguma das duas condições mencionadas for verdadeira, o par não é feito e o próximo estudante C2 é obtido para comparação.

Ao final, percorre-se novamente cada um dos arrays de estudantes para obter os estudantes que não participam de nenhum par.

Função *estudanteJáFoiEscolhido*

```
func estudanteJáFoiEscolhido(e estudante, par []par) bool {
  for i := 0; i < len(par); i++ {
    if par[i].estudante1.nome == e.nome || par[i].estudante2.nome == e.nome {
      return true
    }
  }
  return false
}
```

Essa é uma função auxiliar que ajuda a identificar se um estudante já foi escolhido ou não. Em suma, ela vasculha o array de pares e vê se algum dos estudantes do par é o estudante em questão.

Função *sãoIncompatíveis*

```
func sãoIncompatíveis(e1, e2 estudante) bool {
  for i := 0; i < len(e1.estudantesIncompatíveis); i++ {
    if e1.estudantesIncompatíveis[i].nome == e2.nome {
      return true
    }
  }

  for i := 0; i < len(e2.estudantesIncompatíveis); i++ {
    if e2.estudantesIncompatíveis[i].nome == e1.nome {
      return true
    }
  }

  return false
}
```

Como um dos desafios do exercício era justamente a noção de que estudantes podem querer não dividir o dormitório com determinados outros estudantes, foi necessário implementar essa função. Ela pega dois estudantes e vasculha a lista de incompatibilidades de cada. Se o nome do outro estiver nela, é porque eles não podem conviver.

Análise de complexidade

A quantidade de loops for no código já demonstra a falta de eficiência e a quantidade enorme de iterações pelos mesmos objetos. Vejamos.

Assumindo que C1 é a quantidade de estudantes do curso C1 e que C2 é a quantidade de estudantes do curso C2, temos que:

- Um loop que itera sobre cada item dos estudantes do curso C1 leva a uma complexidade de O(C1);
- Para cada iteração dos estudantes de C1, temos que verificar todos os pares já criados para ver se o estudante de C2 já foi escolhido ou não. Portanto, aqui é introduzida uma complexidade de O(C1 * C2 * P), em que P é a quantidade de pares;
- Além da validação para ver se o estudante de C2 já está em um par, também é feita a validação de compatibilidade. Vasculha-se a lista de incompatibilidades de cada estudante, resultando em uma complexidade de O(C1 * C2 * (EstudanteN.Incompatibilidades + EstudanteO.Incompatibilidades));
- Para obter a quantidade de estudantes sem par, precisamos vasculhar a lista de C1 e a lista de C2 e a lista dos pares. Portanto, complexidade de O(C1 * P + C2 * P)

Dado que nosso maior valor foi durante as etapas para conferir se poderia ser elegido um par em cima dos dois estudantes, temos que a complexidade é de O(N^3), pois há a necessidade de percorrer a lista de estudantes de C1, a lista de estudantes de C2 e as incompatibilidades de cada estudante.

Testes de performance

Para realização dos testes de performance, utilizei o pacote *time* para Go.

Cenário: *seed: 123321 C1: 50 C2: 50 Dormitórios: 100*

```
Duração: [247.362µs]
```

Cenário: *seed: 123321 C1: 500 C2: 500 Dormitórios: 1000*

```
Duração: [169.896786ms]
```

Cenário: seed 0 C1: 10 C2: 100 Dormitórios: 5

```
Duração: [15.86µs]
```