# Trabalho 1 de Estrutura de Dados II

Professora: Vânia de Oliveira Neves

2018.3

José Santos Sá Carvalho – 201665557AC

Brian Maia –

Matheus Oliveira –

Davi Rezende –

**Universidade Federal de Juiz de Fora**

# Introdução

O objetivo deste trabalho é analisar os gastos dos deputados do Congresso brasileiro nos últimos anos. Para este fim, parte-se de um conjunto de dados de cerca de 3 milhões de registros de gastos. Para encontrar os deputados e partidos responsáveis pelos maiores gastos, serão comparados vários algoritmos de ordenação e de busca, a fim de utilizar o algoritmo que se sair melhor. Serão consideradas três métricas de desempenho dos algoritmos de ordenação: tempo de CPU gasto, número de comparações de chaves, e número de cópias de registros. Para os algoritmos de pesquisa, serão considerados o número de comparações de chaves e o gasto de memória. Todas as funções, rotinas e algoritmos foram implementados usando a linguagem C++11.

# 1 Análise dos Algoritmos

Nesta seção, foram comparados os desempenhos de diversos algoritmos de ordenação. Para tal, foi lido o arquivo `deputies_dataset_tratado.csv`, e os registros foram armazenados em um array `allDeputados` da classe `GastoDeputado`, especificada como segue. Cada objeto `GastoDeputado` corresponde a uma linha da tabela, e contém como atributos:

- `bugged_date`
- `receipt_date` – data do recibo;
- `deputy_id` – id do deputado;
- `political_party` – partido político;
- `state_code` – estado que o deputado representa;
- `deputy_name` – nome do deputado
- `receipt_social_security_number` – número do recibo;
- `receipt_description` – descrição do recibo;
- `establishment_name` – nome do estabelecimento;
- `receipt_value` – valor do recibo.

`deputy_id` e `receipt_value` são armazenados como inteiros (`int`). O resto dos atributos são armazanados como strings (biblioteca `<string>`.

A partir daí, são geradas 5 sementes aleatoriamente, a partir de `srand(1)`. Para cada semente, é aberto um arquivo `saida_seed_XXXX.txt` (para a seed `XXXX`, por exemplo), e a semente é passada como argumento para `srand` para a geração de números aleatórios. Em seguida, para cada valor de N lido do arquivo `entrada.txt` – $N = 1000, 5000, 10000, 50000, 100000, 500000$ –, é alocado um array do tipo `GastoDeputado` e um vetor de inteiros, de tamanho N. O primeiro array receberá registros aleatoriamente escolhidos do array `allDeputados`, e o segundo será inicializado com os números `deputy_id` dos mesmos deputados. Desta forma, temos um array de registros, e um de inteiros. Esses arrays serão ordenados conforme o cenário requisitado. Ao final da ordenação por um determinado algoritmo, serão imprimidos no `saida_seed_XXXX.txt` um valor booleano (0/1 corresponente) que diz se o vetor foi ordenado ou não, o tempo de CPU gasto na ordenação em segundos, o número de comparações de chaves, e o número de cópias de registros. Para este fim, uma troca da posição entre dois registros é contabilizada como uma (1) cópia.

Para o cálculo dos números de comparações de chaves, e cópias de registros, em cada função de ordenação há dois argumentos `unsigned int *comp` e `unsigned int *copias`, que são ponteiros para inteiros positivos. Os valores `*comp` e `*copias` são inicializados para 0 no programa principal, e os ponteiros são passados para as funções de ordenação, e incrementados toda vez que há uma comparação entre chaves (`(*comp) += 1`), ou uma cópia de registros (`(*cópias) += 1`).

Funções auxiliares à implementação dos algoritmos foram implementadas conforme necessário. Como exemplos, uma função `swapPtr` para trocar dois elementos de um vetor, uma função `ordenado` que checa se um vetor está ordenado, uma wrapper para a função `rand` que retorna um inteiro (pseudo)aleatório entre dois valores especificados.

Ao término das tarefas requeridas em cada senário especificado, para cada semente, são computadas as médias de cada estatística, e essas são imprimidas no arquivo `saidaFinalCenarioX.txt`, assim como o algoritmo mais rápido dentre os comparados, para cada valor de N.

As computações foram realizadas em um notebook Lenovo Ideapad 710s, com processador Intel Core i7-6700HQ CPU @ 2.60GHz × 8, arquitetura x86_64, 16 GiB de memória RAM, placa de vídeo NVIDIA GeForce GTX 950M, rodando Ubuntu 16.04 64-bit.

## 1.1 Cenário I: Impacto de diferentes estruturas de dados

Neste cenário, foi avaliado o desempenho do algoritmo Quicksort recursivo ao ordenar:

1. Um vetor de inteiros, correspondentes a ID's de deputados aleatoriamente selecionados do conjunto de dados, como delineado acima;

2. Um vetor de registros de gastos de deputados, isto é, um vetor de estruturas `GastoDeputado`, especificadas acima. O ID dos deputados foi usado como chave.

O algoritmo Quicksort recursivo foi implementado de forma similar à vista em sala de aula. Os detalhes do uso das funções e rotinas seguem abaixo.

---

```
void quickSort(int *left, int *right, unsigned int *comp, unsigned int *copias)
```

---

```
void quickSortDeputyId(GastoDeputado *leftDep, GastoDeputado *rightDep,
unsigned int *comp, unsigned int *copias)
```

---

**Argumentos:**

`*left` ponteiro para o primeiro elemento do vetor a ser ordenado.

`*right` ponteiro para o último elemento do vetor a ser ordenado.

`*comp` ponteiro para contador de comparações de chaves.

`*copias` ponteiro para contador de cópias de registros.

Os algoritmos fazem uso de uma rotina para particionar o vetor, a partir de um pivô, que corresponde ao elemento mais à direita do vetor. A rotina está especificada como segue.

---

```
int *partitionIt(int *left, int *right, int pivot, unsigned int *comp,
unsigned int *copias)
```

---

```
GastoDeputado *partitionItDeputyId(GastoDeputado *leftDep,
GastoDeputado *rightDep, int pivot, unsigned int *comp, unsigned int *copias)
```

---

**Argumentos:**

`*left` ponteiro para o primeiro elemento do vetor a ser ordenado.

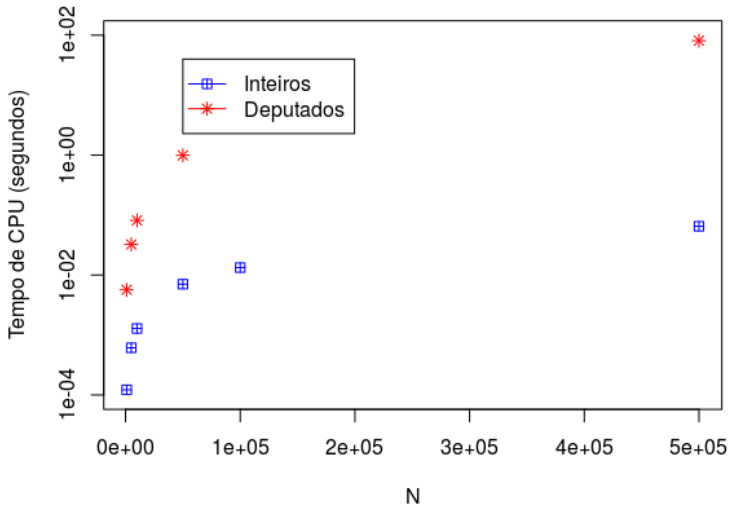
`*right` ponteiro para o último elemento do vetor a ser ordenado.

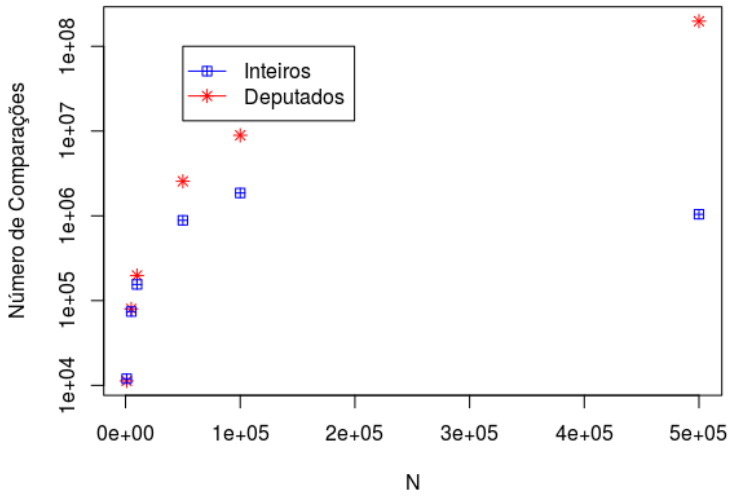
`pivot` pivô a partir do qual se faz o particionamento.

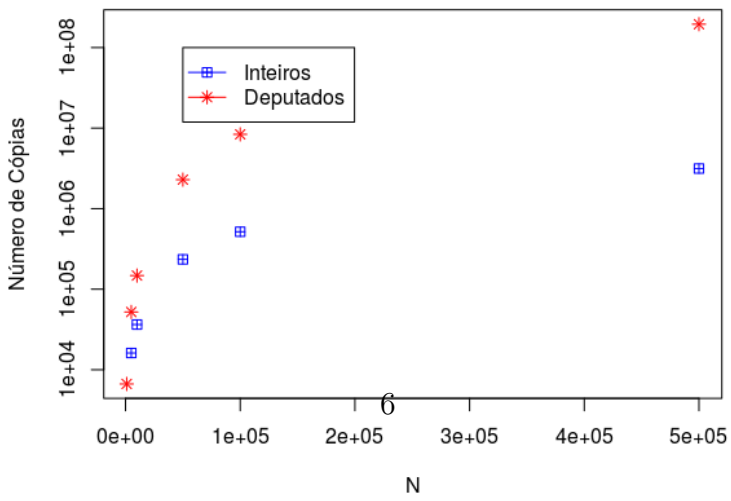
`*comp` ponteiro para contador de comparações de chaves.

`*copias` ponteiro para contador de cópias de registros.

As médias de cada estatística computada, para cada algoritmo, e cada valor de N, são exibidas nos gráficos abaixo.


Tempo de CPU (segundos)
1e-04
1e-02
1e+00
1e+02
Inteiros
Deputados
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
N


Número de Comparações
1e+04
1e+05
1e+06
1e+07
1e+08
Inteiros
Deputados
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
N


Número de Cópias
1e+04
1e+05
1e+06
1e+07
1e+08
Inteiros
Deputados
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
N