Relatório
Trabalho Final
Árvores

Gabriel Differoni Leal
João Pastorello

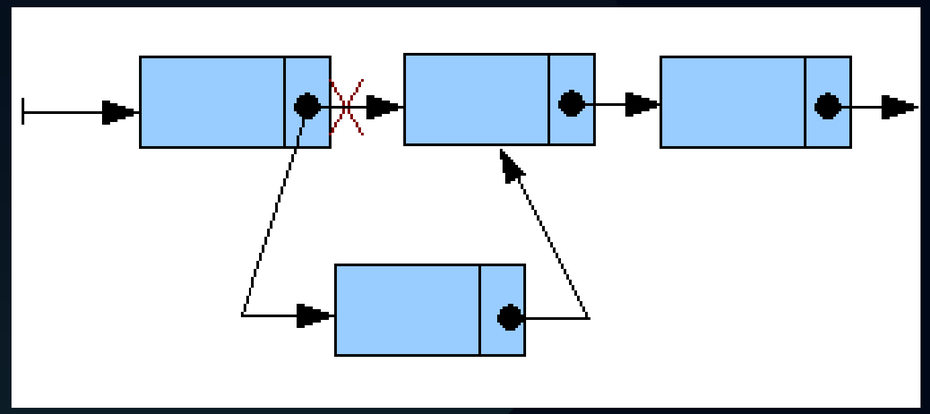
Introdução

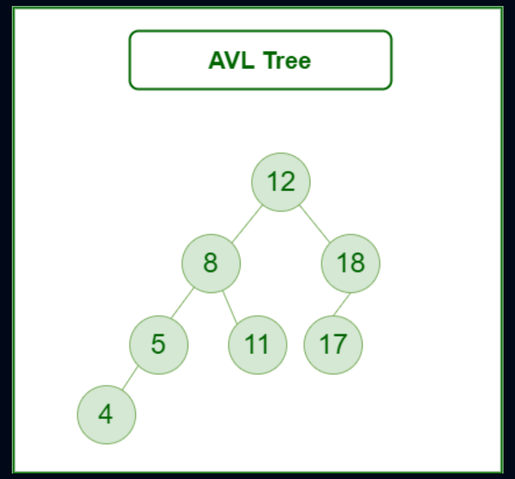
O trabalho foi feito com o objetivo de comparar diferentes estruturas de dados, sendo elas: Lista Simplesmente Encadeada e AVL (árvore de busca balanceada). Ambas foram submetidas à inserção de dados, sendo estes usuários com um número de login e uma senha composta por diferentes caracteres. Além disso, testes com 10,100 e 1000 usuários foram realizados para determinar qual das duas implementações é mais rápida em executar as operações de carregamento e consulta.

O conceito de **Benchmark** foi a base para o desenvolvimento do projeto, sendo esse um método na computação que consiste no ato de comparar de forma eficiente a performance entre objetos utilizando um ou mais programas.

Estrutura de dados e Geração de dados

- **LSE:** A escolha da **Lista Simplesmente Encadeada** se deu pela facilidade da implementação do algoritmo dessa estrutura, assim como as funções associadas à ela, sendo elas simples e intuitivas. Além disso, operações como a de inserção de elementos mostrou-se muito eficiente no carregamento dos usuários, no qual eles eram sempre inseridos no início da lista.
- **AVL:** A principal vantagem do uso da **AVL** em comparação a outras estruturas é a velocidade de checagem de dados. O balanceamento por altura (O(log n)) permite a manutenção da estrutura da árvore, facilitando, principalmente, na operação de consulta, na qual leva vantagem em relação à LSE. A complexidade de tempo garantida e o desempenho consistente foram características decisivas para a escolha da AVL neste trabalho, fazendo dela uma escolha popular para uma ampla variedade de aplicações que exigem operações eficientes em conjuntos de dados dinâmicos.





Estrutura de dados e Geração de dados

Os dados na implementação do trabalho eram usuários, representados por uma estrutura que possui um campo de login (apenas números) e uma senha (qualquer caractere). A geração dos usuários foi feita utilizando duas funções principais:

- **generate_random_string:** gera uma string aleatória para ser usada como senha do usuário.
- **gera_dados:** gera um número para ser o login do usuário e utiliza a função anterior para montar o usuário com a senha e o número gerado.

Após isso, os dados são inseridos em arquivos, com logins ordenados ou não, que variam entre 500, 10.000 e 100.000 usuários.

```c
char* generate_random_string(unsigned length) {
    char* string = malloc((length + 1) * sizeof(char));
    const char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    int charset_length = strlen(charset);

    for (unsigned i = 0; i < length; i++) {
        int random_index = rand() % charset_length;
        string[i] = charset[random_index];
    }

    string[length] = '\0';
    return string;
}
```

```c
void gera_dados(char nome_dados[], char nome_dados_ord[], int n) {
    srand((unsigned int)time(NULL));

    // Create and fill the array with data in order
    Login* data = malloc((size_t)n * sizeof(Login));
    for (int i = 0; i < n; i++) {
        data[i].usr = i + 1;
        strcpy(data[i].senha, generate_random_string(TAM_SENHA));
    }

    // Escreve os dados ordenados no arquivo
    FILE *arq = cria_arq_escrita(nome_dados_ord);
    for (int i = 0; i < n; i++) {
        fprintf(arq, "%d,%s\n", data[i].usr, data[i].senha);
    }
```

Metodologia

Desenvolvimento do algoritmo e construção do código para implementar as estruturas

Geração da base de dados do experimento pelas funções do slide anterior

Criação dos arquivos de testes (10% e 50%) utilizando como base os dados dos usuários

Execução das funções de inserção das estruturas de dados escolhidas

Escrita dos resultados em um arquivo contendo o tempo de carregamento e consulta da LSE e AVL

Análise dos dados obtidos para a criação de gráficos e desenvolvimento do relatório

Harware

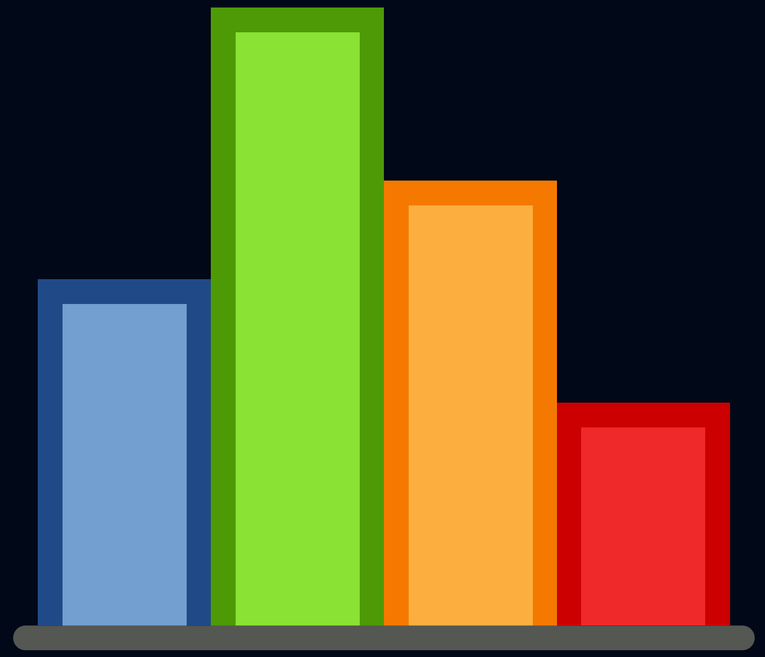
Foi utilizado para os testes um notebook Dell modelo Inspiron-13-7359 com um processador Intel Core i7-6500U CPU 2.5 GHz com 8GB de memória ram. O sistema operacional é o Linux Mint 21.3 Cinnamon.

Github Copilot

O GitHub Copilot é uma ferramenta de inteligência artificial desenvolvida pelo GitHub em conjunto com a OpenAI, afim de auxiliar usuários de ambientes de desenvolvimento integrados como Visual Studio Code, que foi o nosso caso. Essa ferramenta fornece dicas durante o desenvolvimento do código, oferecendo sugestões de preenchimento automático conforme você codifica. Durante o nosso projeto, utilizamos ela para criar funções que geravam os dados e os testes a serem realizados. As funções “generate_random_string” , “gera_dados” e “gera_testes” são exemplos em que o GitHub Copilot colaborou para o desenvolvimento do trabalho.

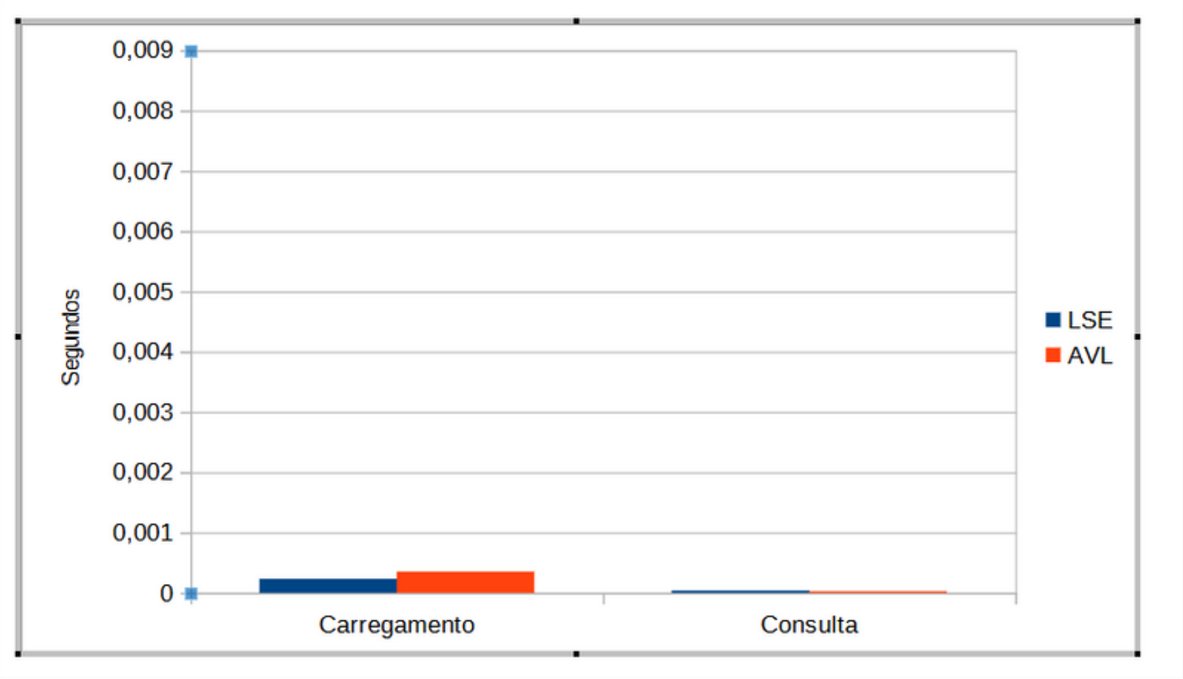
Análise dos resultados

Os slides a seguir referem às medidas de tempo em segundos encontradas em cada caso de teste. Foram utilizados ao todo 6 arquivos de usuários para a base de dados, variando entre 500,10.000 e 100.000, ordenados ou não. As estruturas foram testadas com 10, 100 e 1.000 usuários, com 10% ou 50% das senhas erradas, objetivando verificar se existe uma diferença significativa entre a **LSE e AVL**.



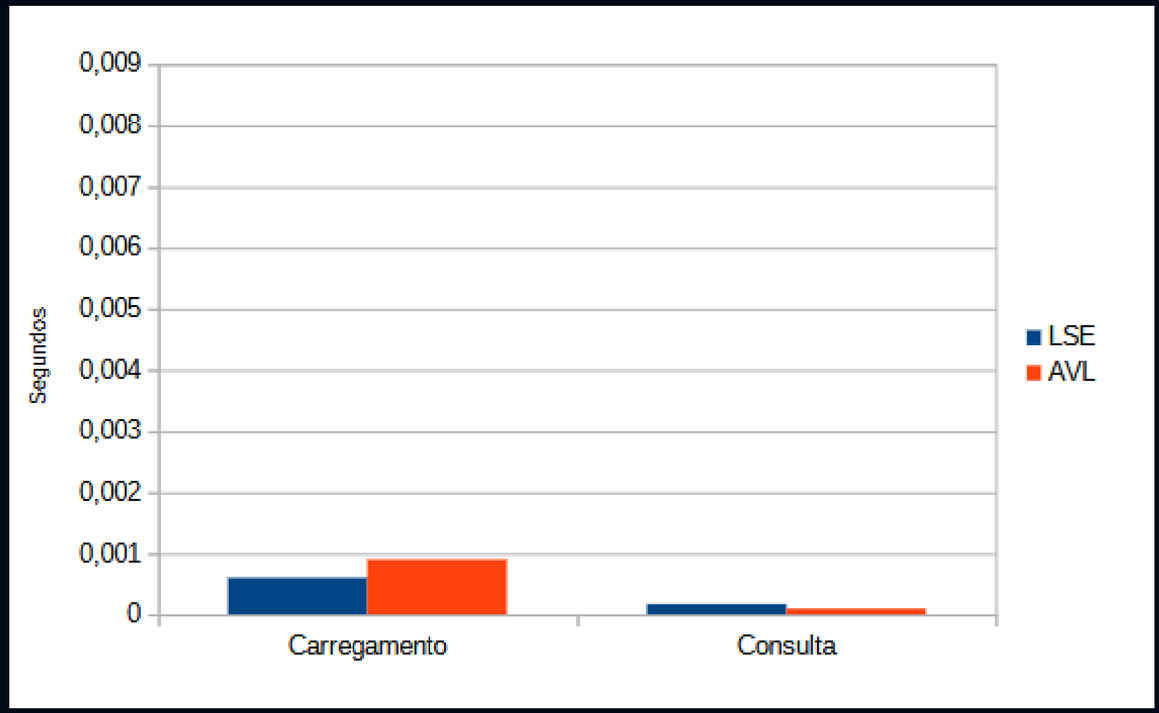
500 DADOS – TESTE COM 10 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.000233 segundos
Tempo de consulta LSE: 0.000038 segundos
Tempo de carregamento AVL: 0.000353 segundos
Tempo de consulta AVL: 0.000027 segundos
```

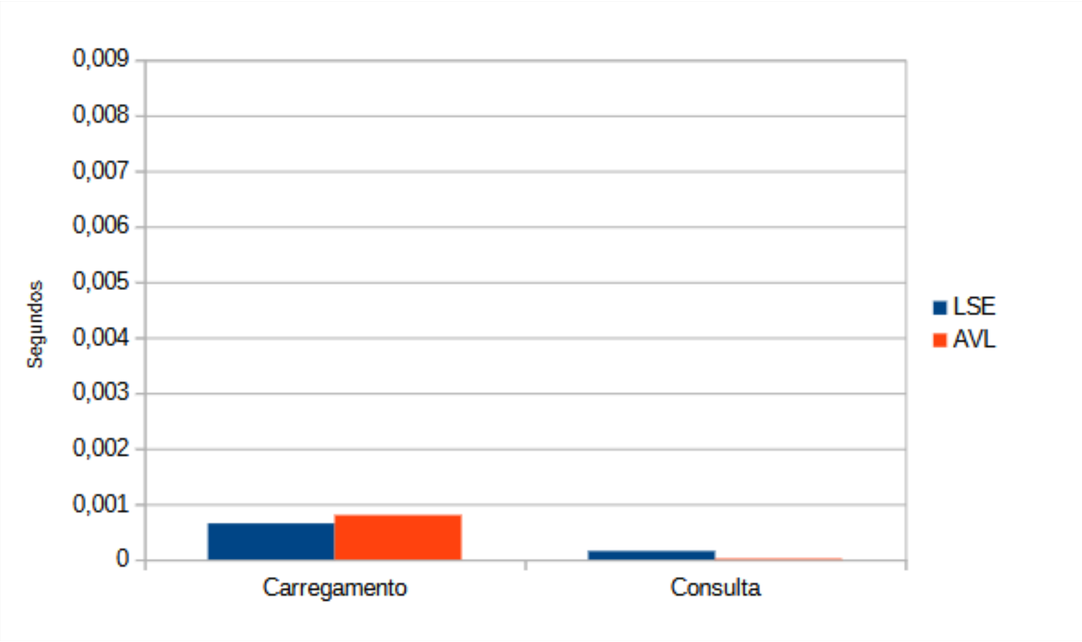
Ordenados:




```
Tempo de carregamento LSE: 0.000604 segundos
Tempo de consulta LSE: 0.000168 segundos
Tempo de carregamento AVL: 0.000900 segundos
Tempo de consulta AVL: 0.000093 segundos
```

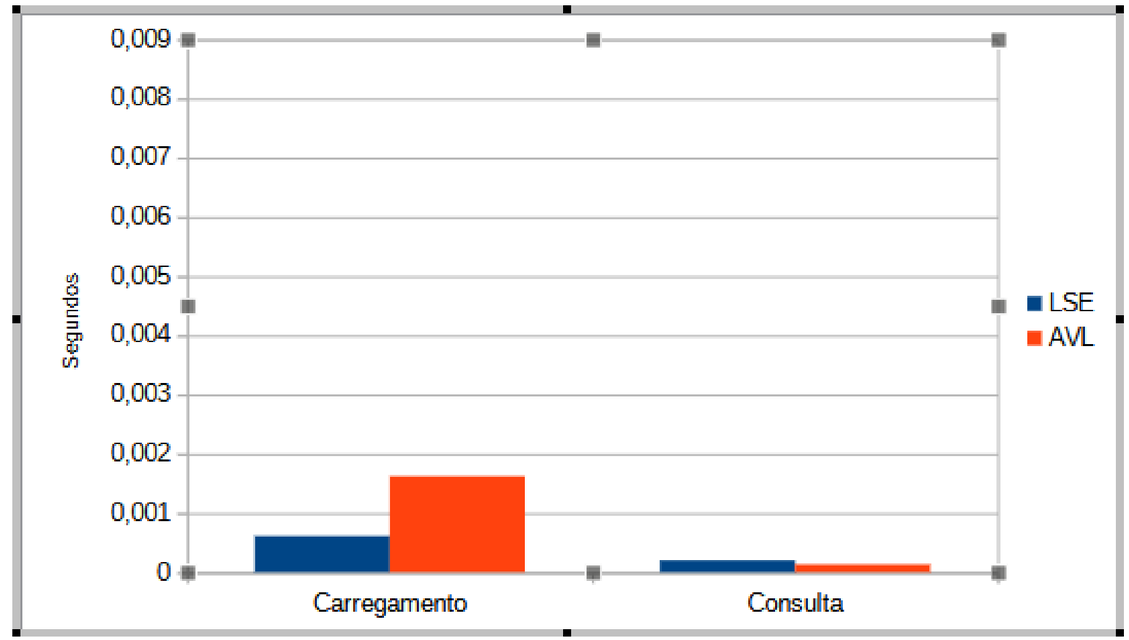
500 DADOS – TESTE COM 10 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.000650 segundos
Tempo de consulta LSE: 0.000152 segundos
Tempo de carregamento AVL: 0.000800 segundos
Tempo de consulta AVL: 0.000018 segundos
```

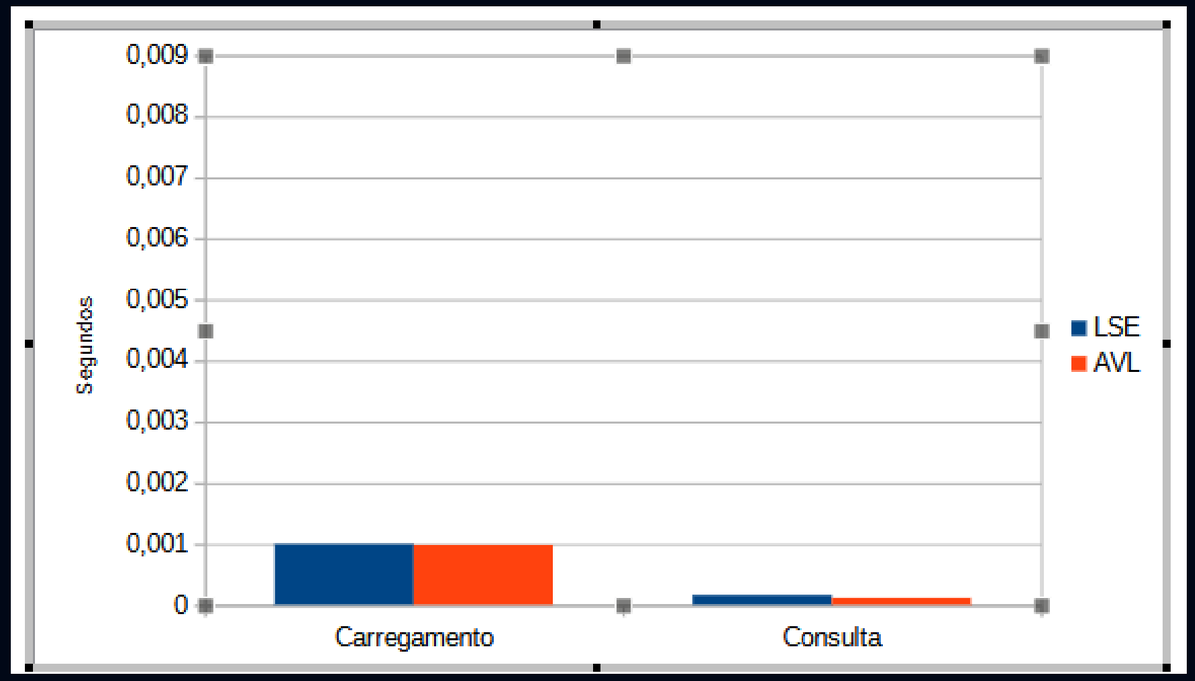
Ordenados:




```
Tempo de carregamento LSE: 0.000606 segundos
Tempo de consulta LSE: 0.000202 segundos
Tempo de carregamento AVL: 0.001616 segundos
Tempo de consulta AVL: 0.000125 segundos
```

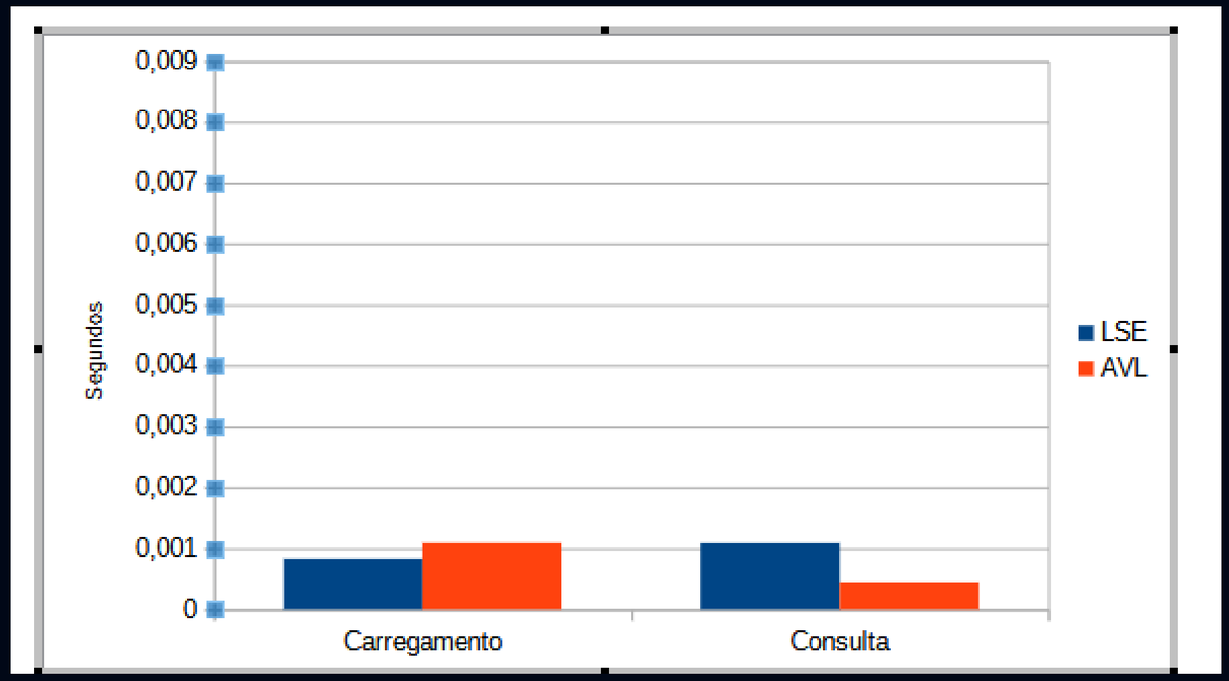
500 DADOS – TESTE COM 100 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.001003 segundos
Tempo de consulta LSE: 0.000167 segundos
Tempo de carregamento AVL: 0.000988 segundos
Tempo de consulta AVL: 0.000113 segundos
```

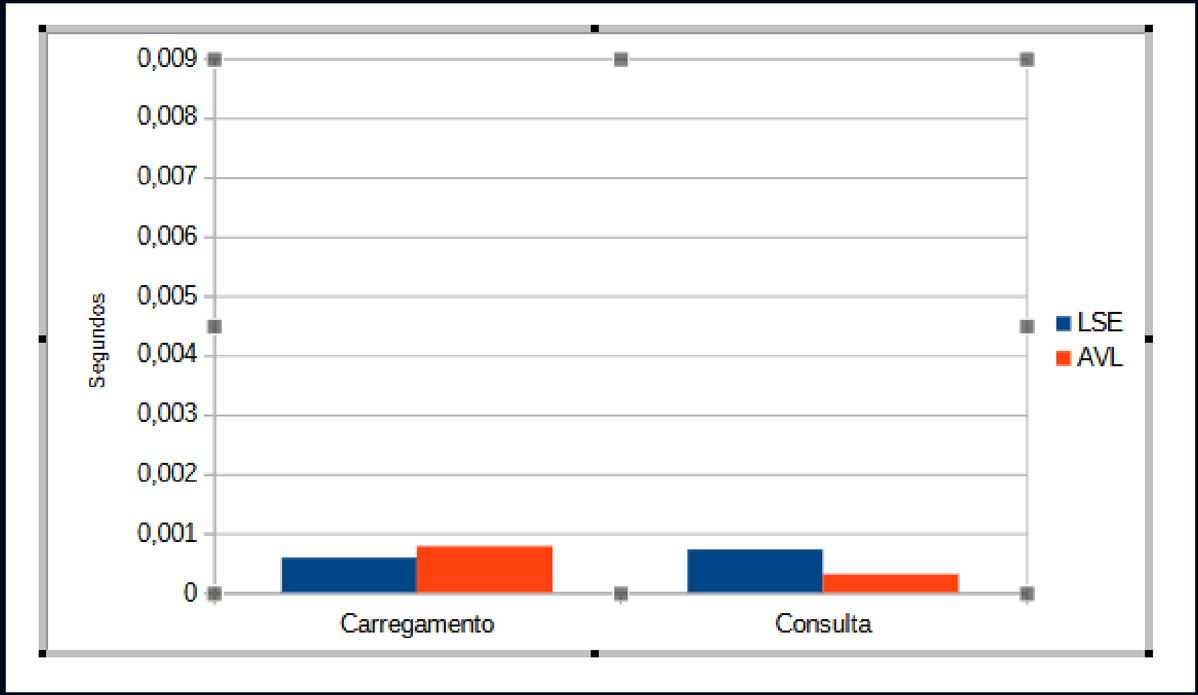
Ordenados:




```
Tempo de carregamento LSE: 0.000829 segundos
Tempo de consulta LSE: 0.001088 segundos
Tempo de carregamento AVL: 0.001090 segundos
Tempo de consulta AVL: 0.000437 segundos
```

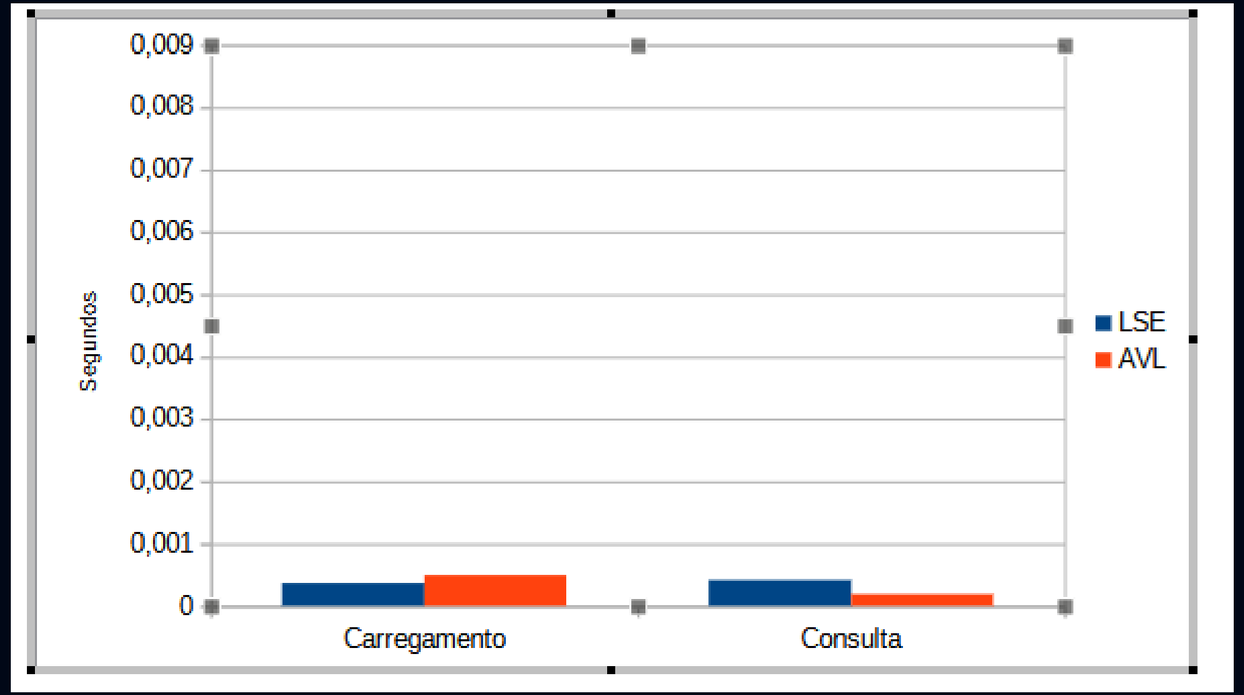
500 DADOS – TESTE COM 100 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.000592 segundos
Tempo de consulta LSE: 0.000725 segundos
Tempo de carregamento AVL: 0.000778 segundos
Tempo de consulta AVL: 0.000314 segundos
```

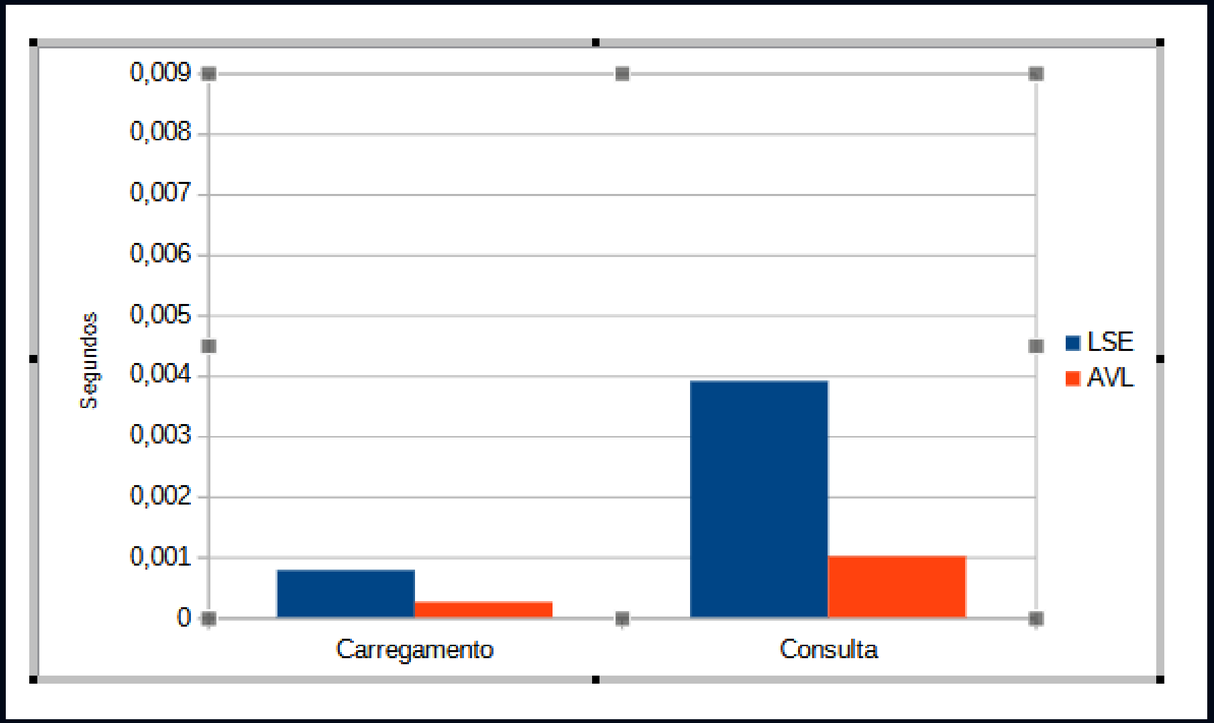
Ordenados:




```
Tempo de carregamento LSE: 0.000366 segundos
Tempo de consulta LSE: 0.000416 segundos
Tempo de carregamento AVL: 0.000496 segundos
Tempo de consulta AVL: 0.000188 segundos
```

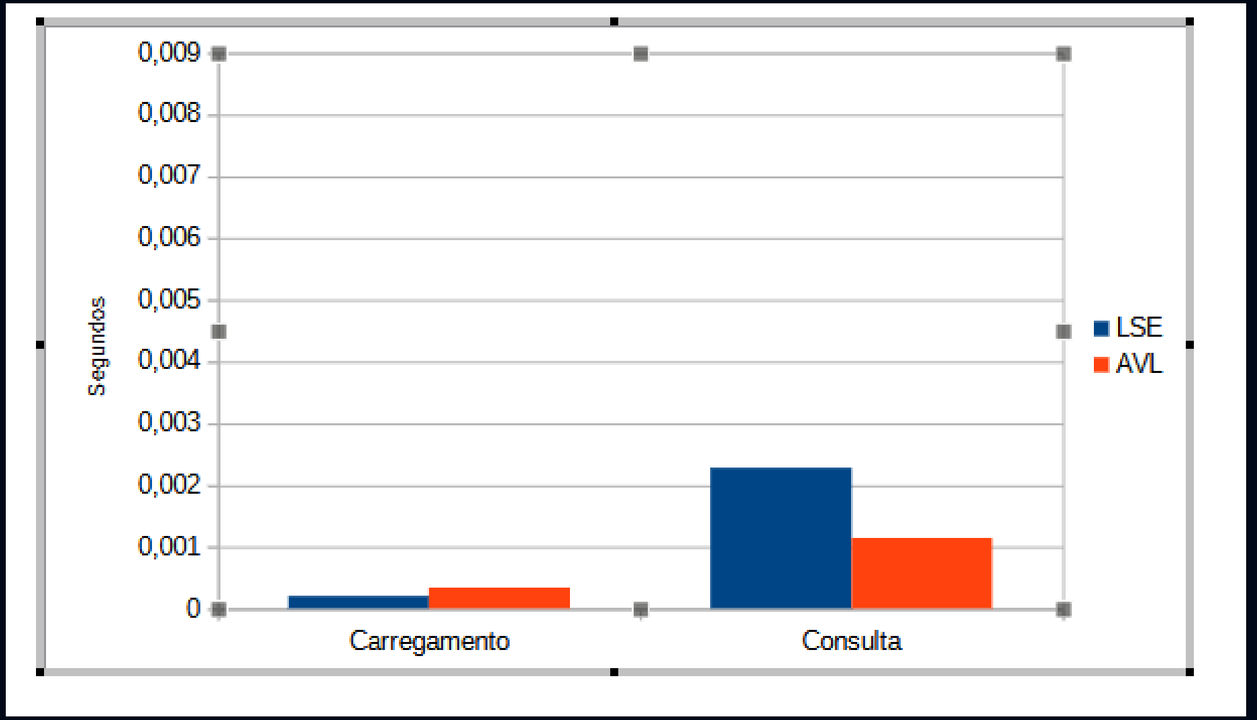
500 DADOS – TESTE COM 1000 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.000782 segundos
Tempo de consulta LSE: 0.003902 segundos
Tempo de carregamento AVL: 0.000257 segundos
Tempo de consulta AVL: 0.001008 segundos
```

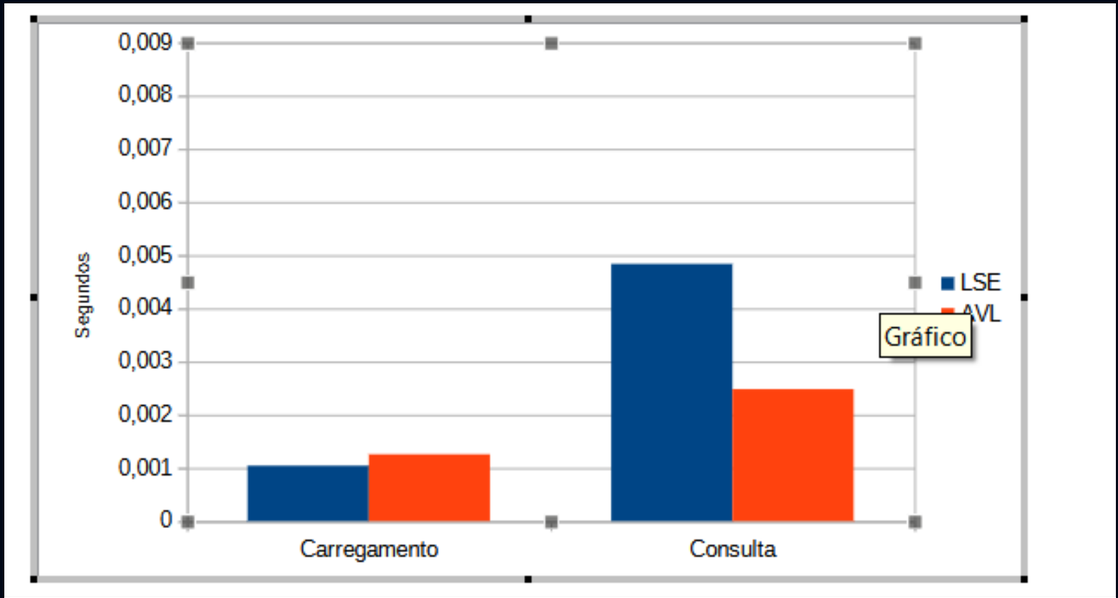
Ordenados:




```
Tempo de carregamento LSE: 0.000201 segundos
Tempo de consulta LSE: 0.002283 segundos
Tempo de carregamento AVL: 0.000336 segundos
Tempo de consulta AVL: 0.001150 segundos
```

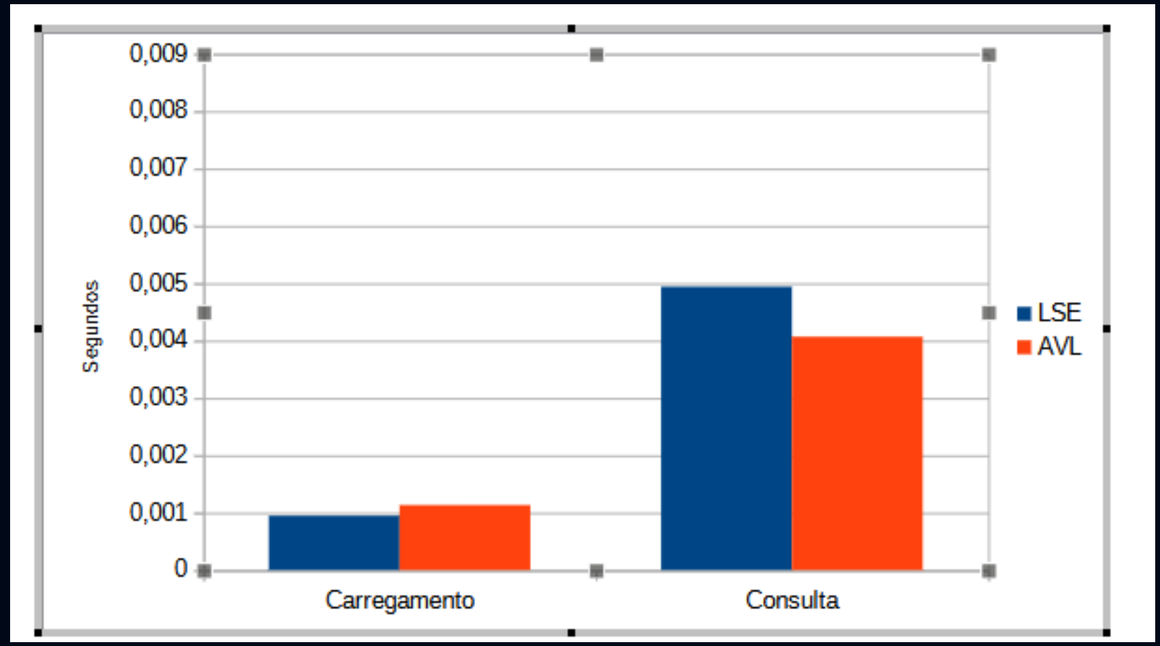
500 DADOS – TESTE COM 1000 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.001041 segundos
Tempo de consulta LSE: 0.004836 segundos
Tempo de carregamento AVL: 0.001255 segundos
Tempo de consulta AVL: 0.002481 segundos
```

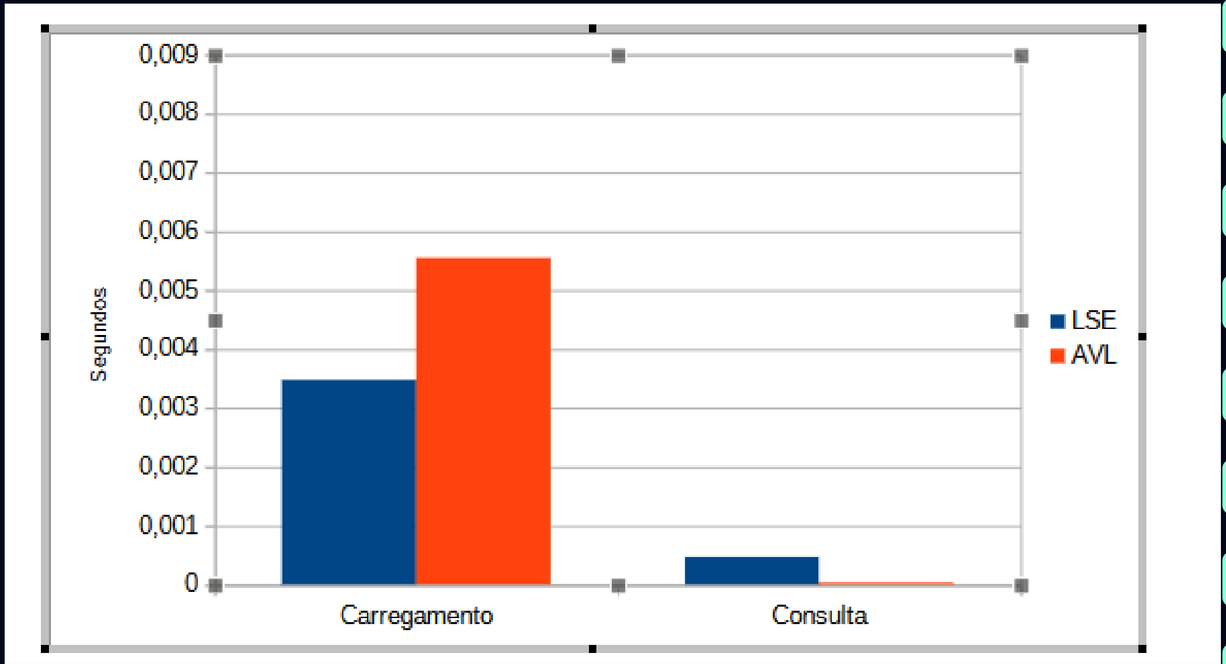
Ordenados:




```
Tempo de carregamento LSE: 0.000947 segundos
Tempo de consulta LSE: 0.004939 segundos
Tempo de carregamento AVL: 0.001132 segundos
Tempo de consulta AVL: 0.004067 segundos
```

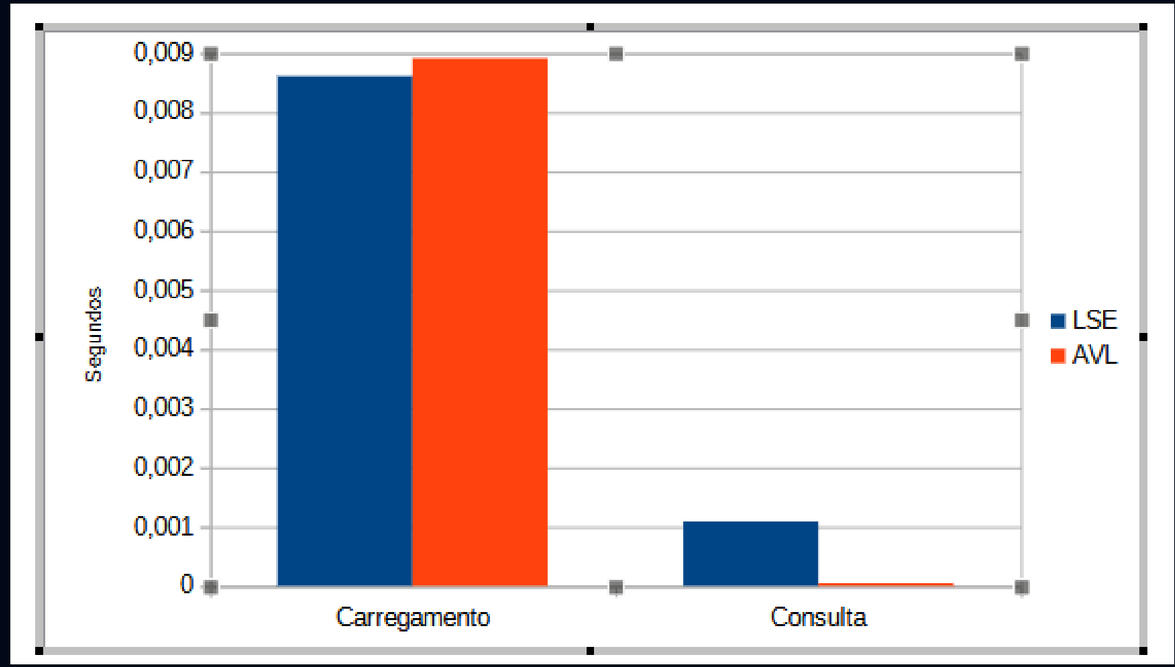
10.000 DADOS – TESTE COM 10 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.003476 segundos
Tempo de consulta LSE: 0.000468 segundos
Tempo de carregamento AVL: 0.005550 segundos
Tempo de consulta AVL: 0.000031 segundos
```

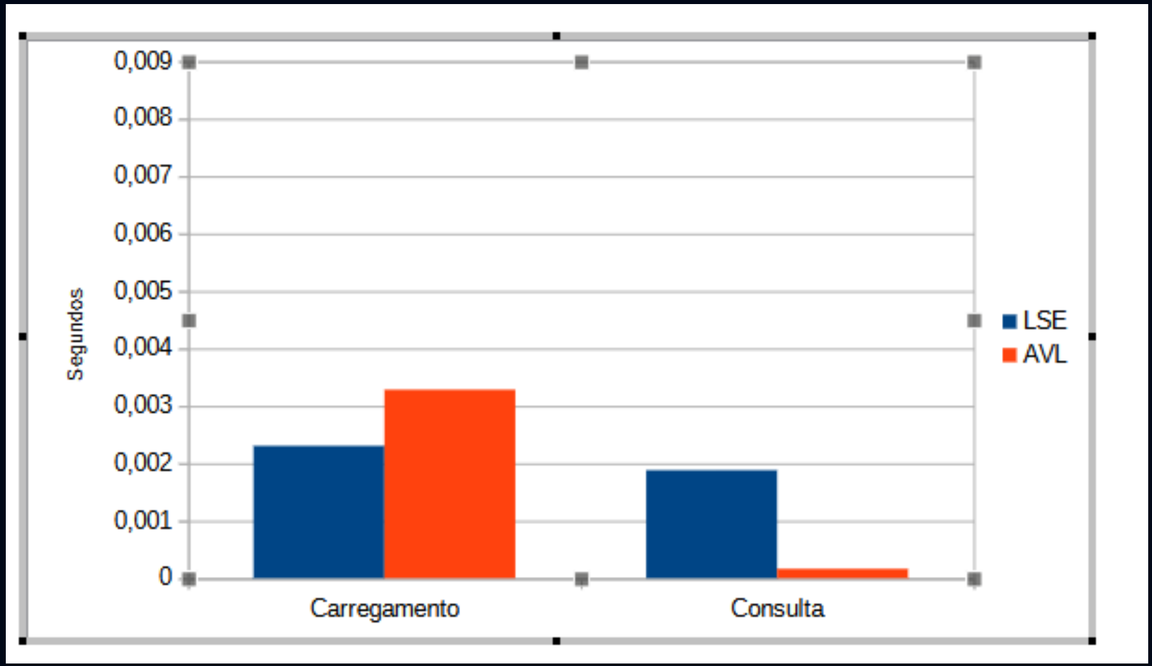
Ordenados:




```
Tempo de carregamento LSE: 0.008606 segundos
Tempo de consulta LSE: 0.001083 segundos
Tempo de carregamento AVL: 0.008905 segundos
Tempo de consulta AVL: 0.000045 segundos
```

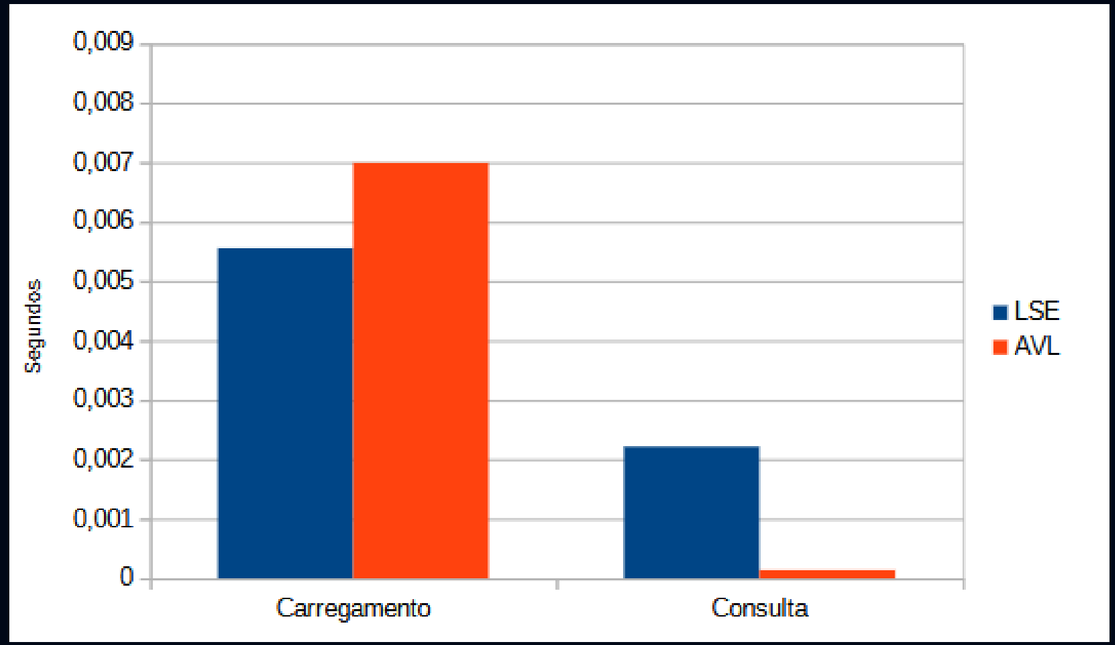
10.000 DADOS – TESTE COM 10 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.002303 segundos
Tempo de consulta LSE: 0.001879 segundos
Tempo de carregamento AVL: 0.003282 segundos
Tempo de consulta AVL: 0.000163 segundos
```

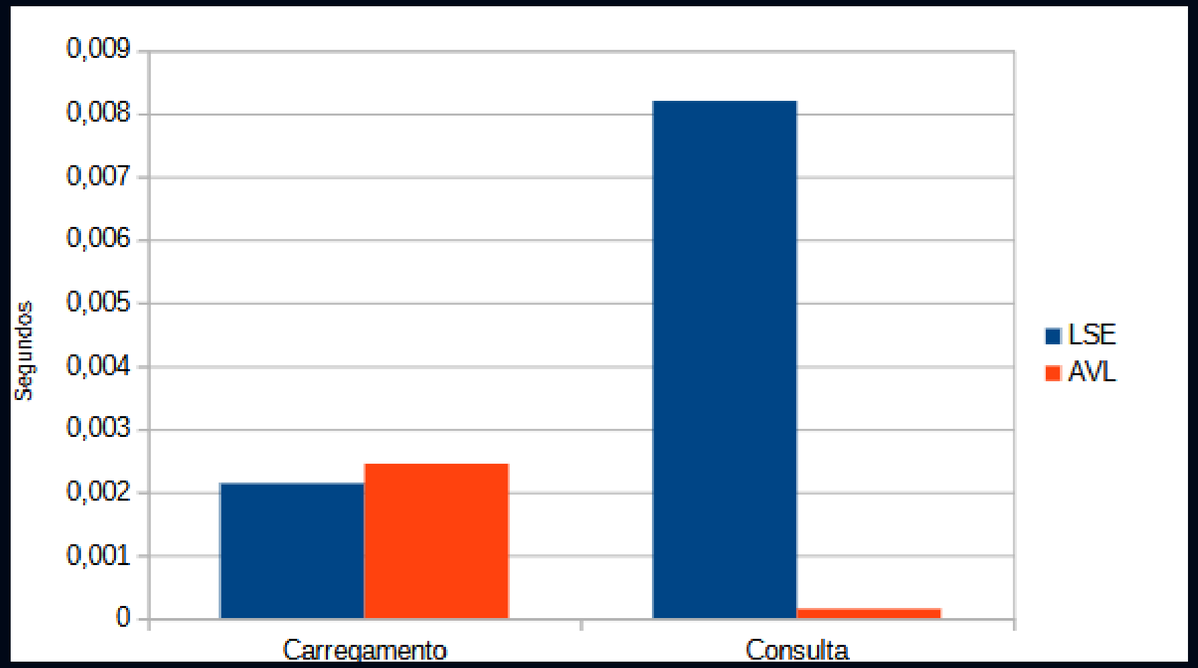
Ordenados:




```
Tempo de carregamento LSE: 0.005553 segundos
Tempo de consulta LSE: 0.002212 segundos
Tempo de carregamento AVL: 0.006990 segundos
Tempo de consulta AVL: 0.000135 segundos
```

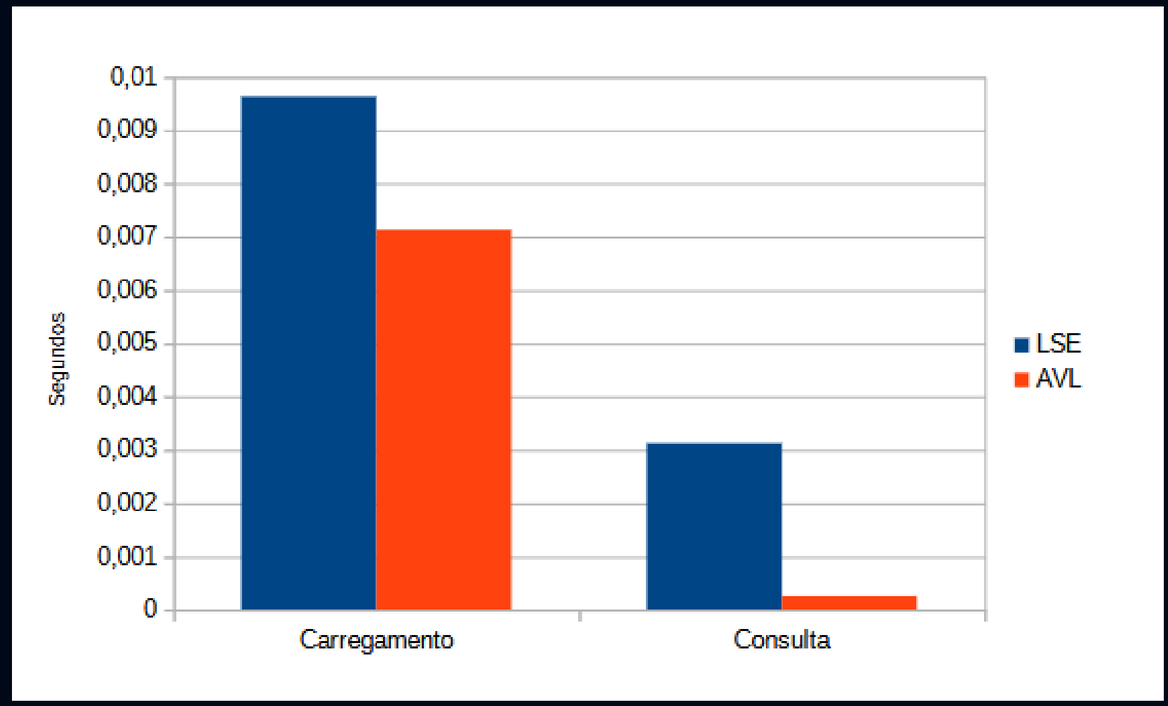
10.000 DADOS – TESTE COM 100 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.002141 segundos
Tempo de consulta LSE: 0.008193 segundos
Tempo de carregamento AVL: 0.002450 segundos
Tempo de consulta AVL: 0.000150 segundos
```

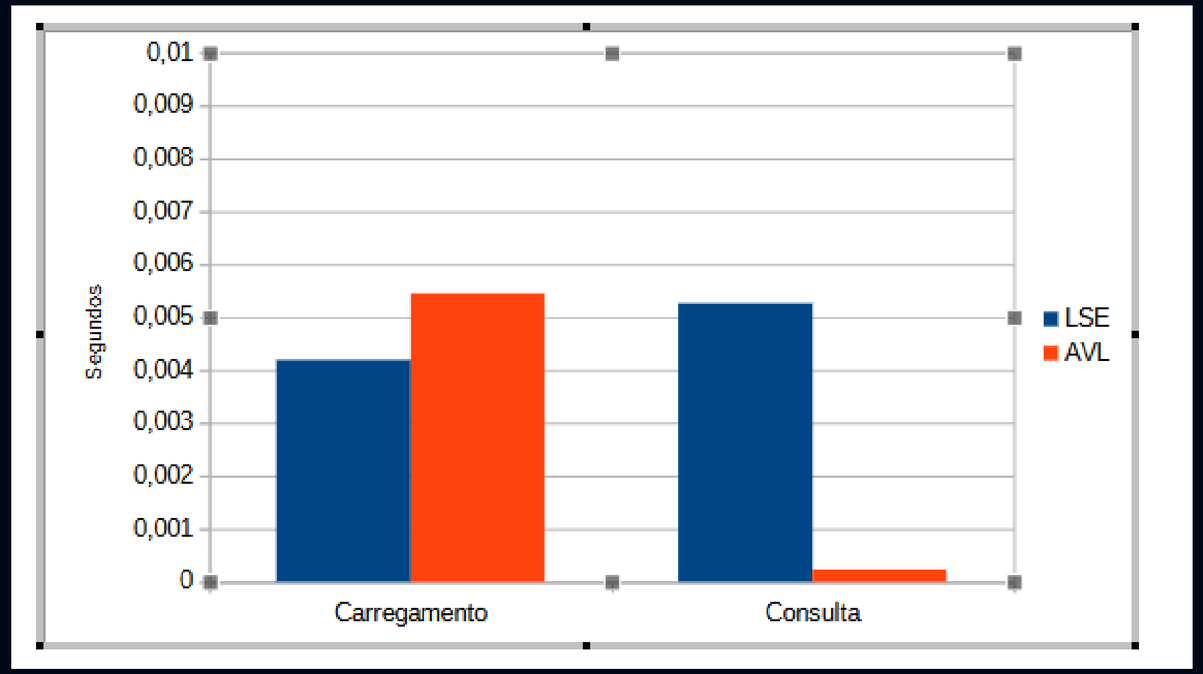
Ordenados:




```
Tempo de carregamento LSE: 0.009635 segundos
Tempo de consulta LSE: 0.003131 segundos
Tempo de carregamento AVL: 0.007133 segundos
Tempo de consulta AVL: 0.000256 segundos
```

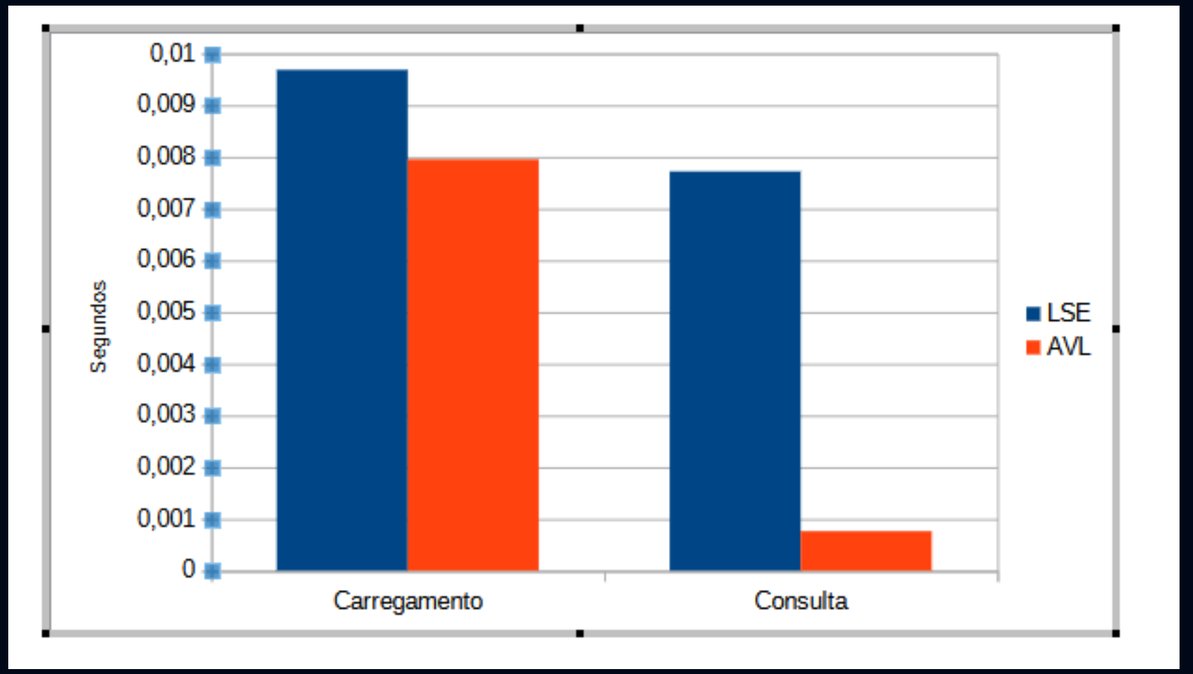
10.000 DADOS – TESTE COM 100 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.004186 segundos
Tempo de consulta LSE: 0.005264 segundos
Tempo de carregamento AVL: 0.005443 segundos
Tempo de consulta AVL: 0.000230 segundos
```

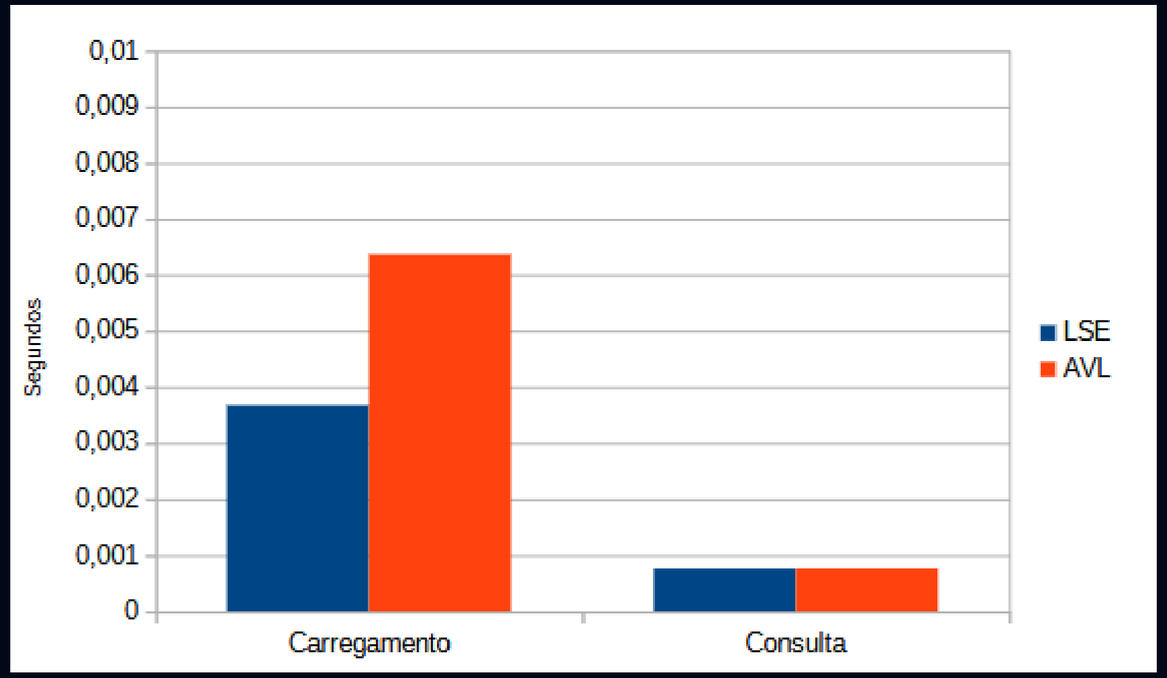
Ordenados:




```
Tempo de carregamento LSE: 0.009683 segundos
Tempo de consulta LSE: 0.007718 segundos
Tempo de carregamento AVL: 0.007951 segundos
Tempo de consulta AVL: 0.000763 segundos
```

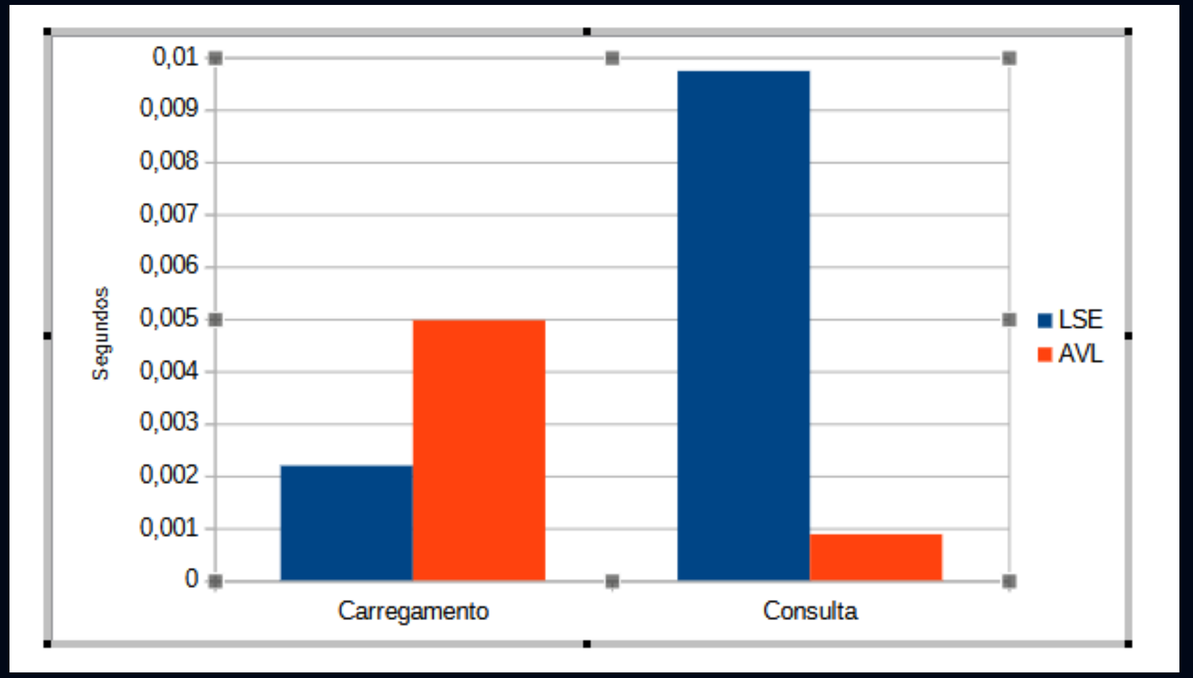
10.000 DADOS – TESTE COM 1000 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.003680 segundos
Tempo de consulta LSE: 0.000760 segundos
Tempo de carregamento AVL: 0.006364 segundos
Tempo de consulta AVL: 0.000764 segundos
```

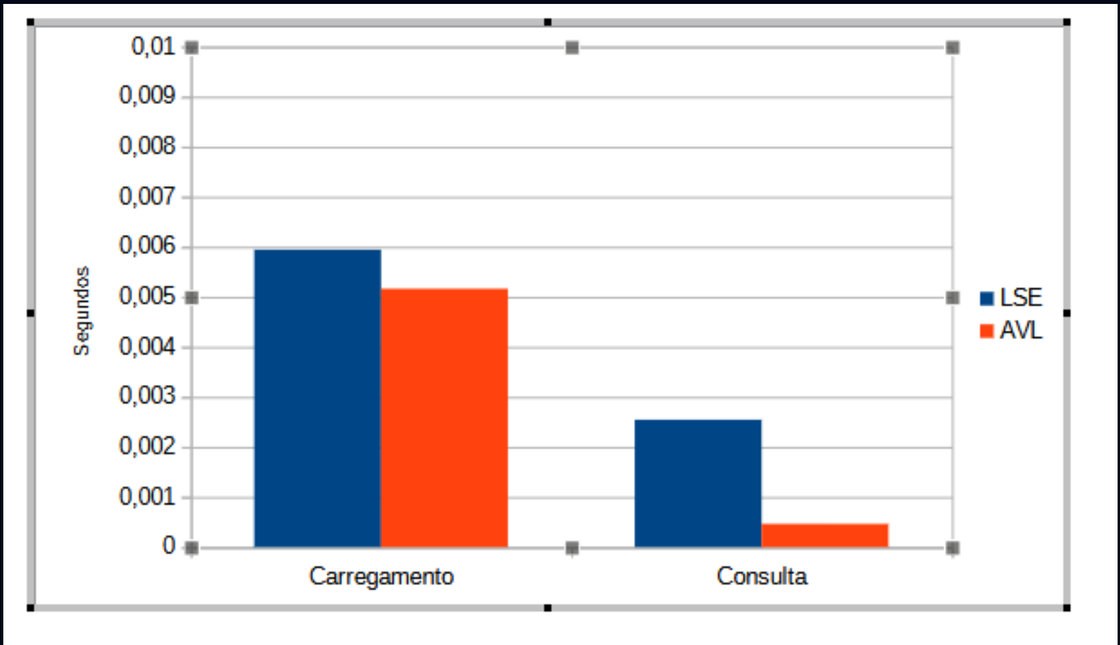
Ordenados:




```
Tempo de carregamento LSE: 0.002193 segundos
Tempo de consulta LSE: 0.009734 segundos
Tempo de carregamento AVL: 0.004970 segundos
Tempo de consulta AVL: 0.000881 segundos
```

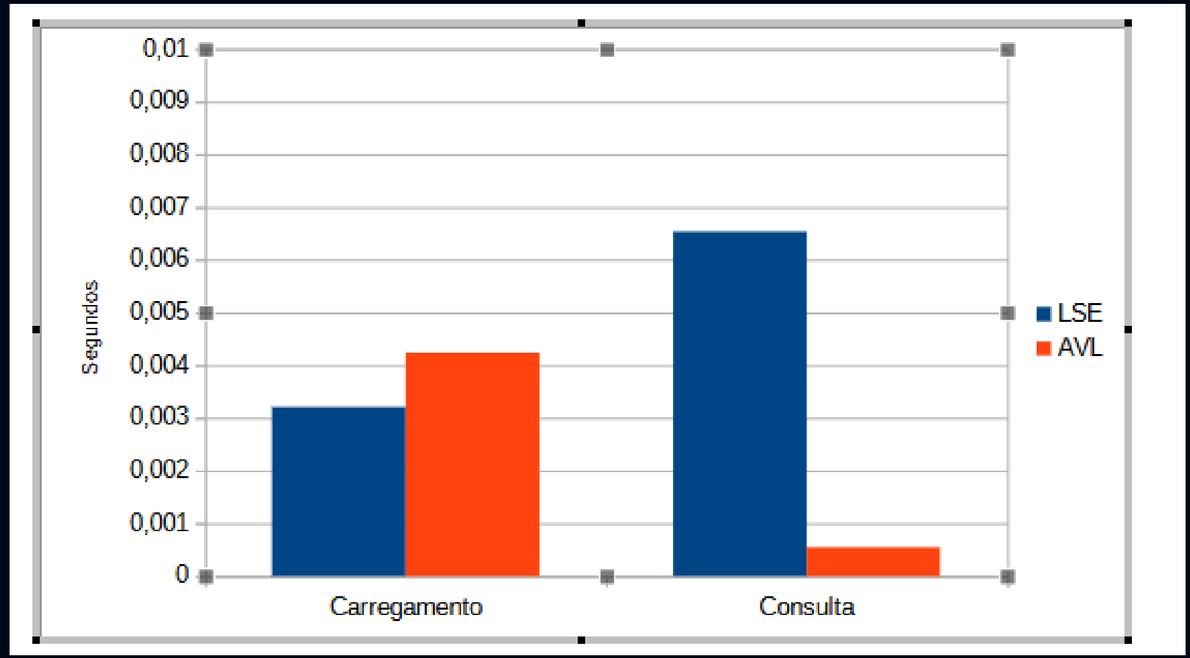
10.000 DADOS – TESTE COM 1000 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.005942 segundos
Tempo de consulta LSE: 0.002544 segundos
Tempo de carregamento AVL: 0.005162 segundos
Tempo de consulta AVL: 0.000462 segundos
```

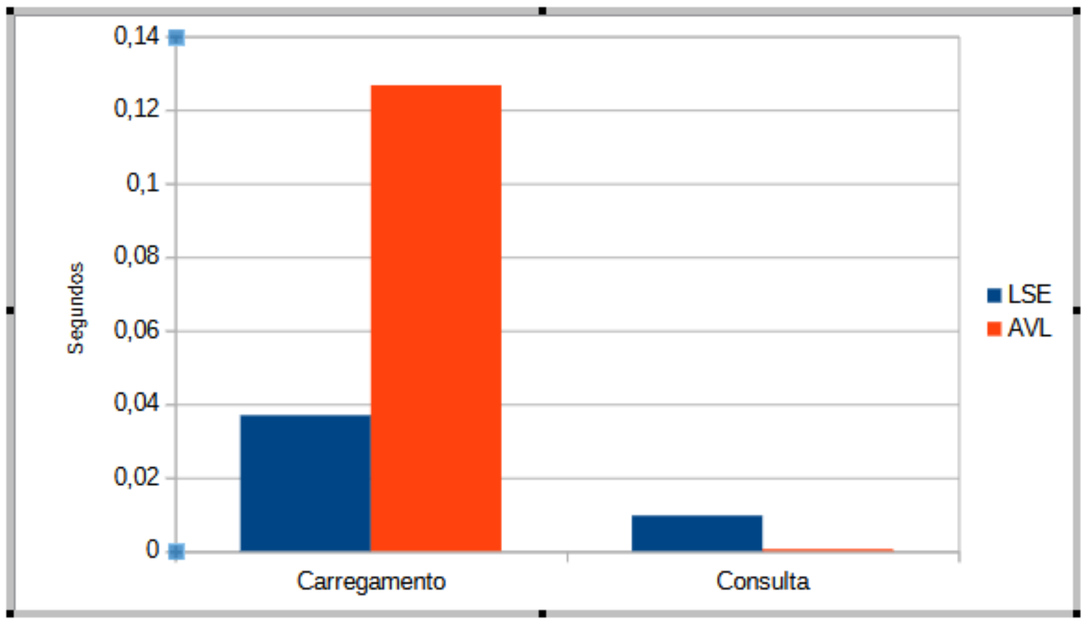
Ordenados:




```
Tempo de carregamento LSE: 0.003205 segundos
Tempo de consulta LSE: 0.006534 segundos
Tempo de carregamento AVL: 0.004240 segundos
Tempo de consulta AVL: 0.000540 segundos
```

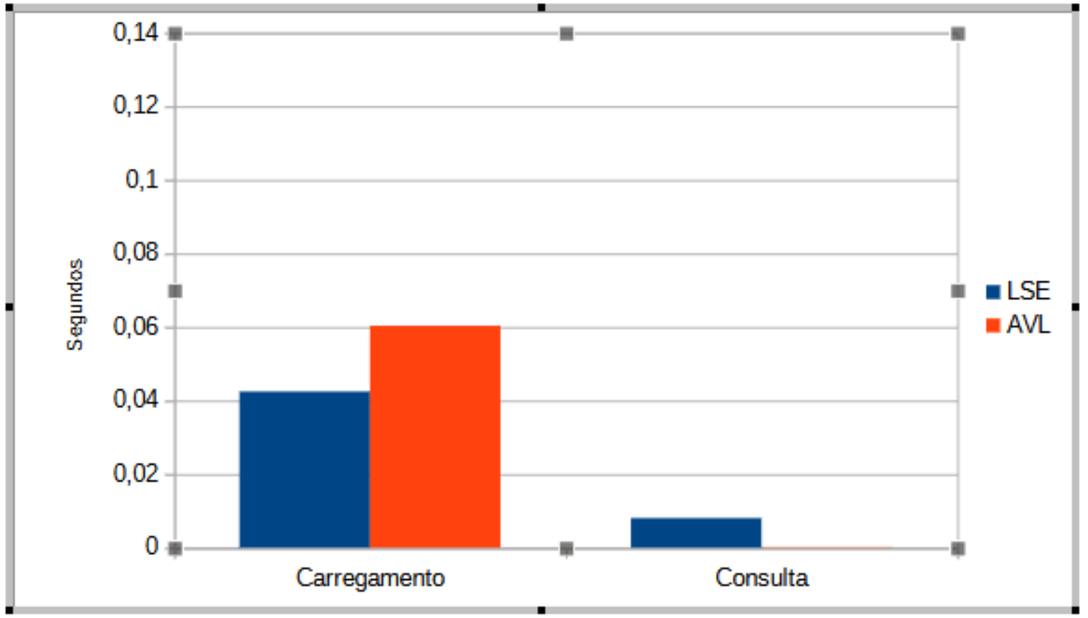
100.000 DADOS – TESTE COM 10 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.036929 segundos
Tempo de consulta LSE: 0.009659 segundos
Tempo de carregamento AVL: 0.126639 segundos
Tempo de consulta AVL: 0.000526 segundos
```

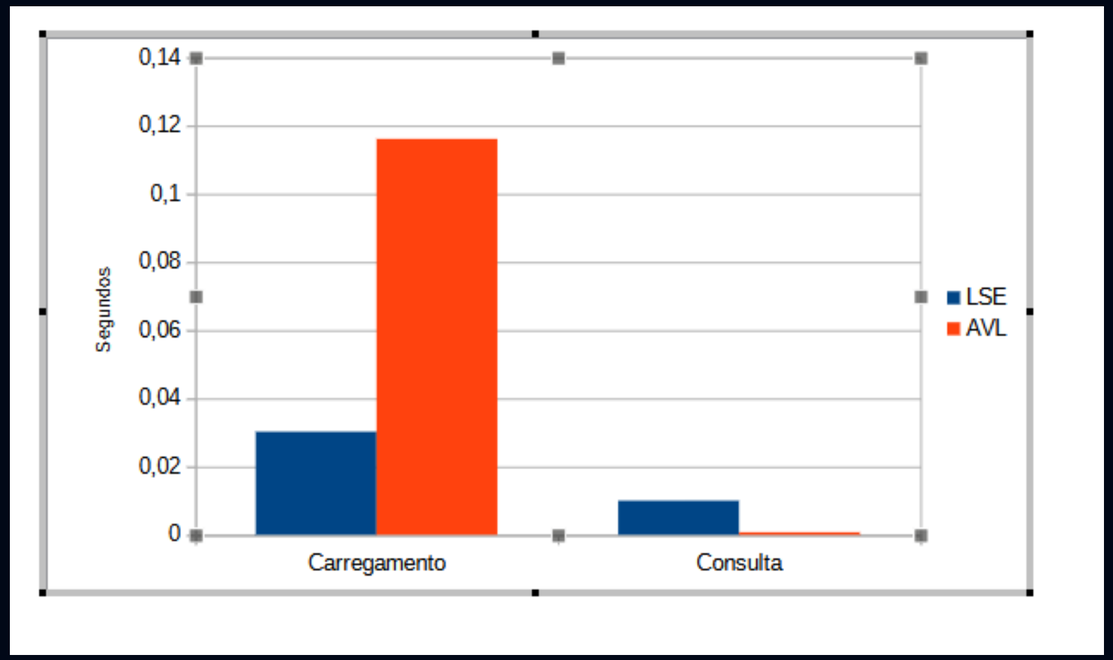
Ordenados:




```
Tempo de carregamento LSE: 0.042439 segundos
Tempo de consulta LSE: 0.008027 segundos
Tempo de carregamento AVL: 0.060316 segundos
Tempo de consulta AVL: 0.000143 segundos
```

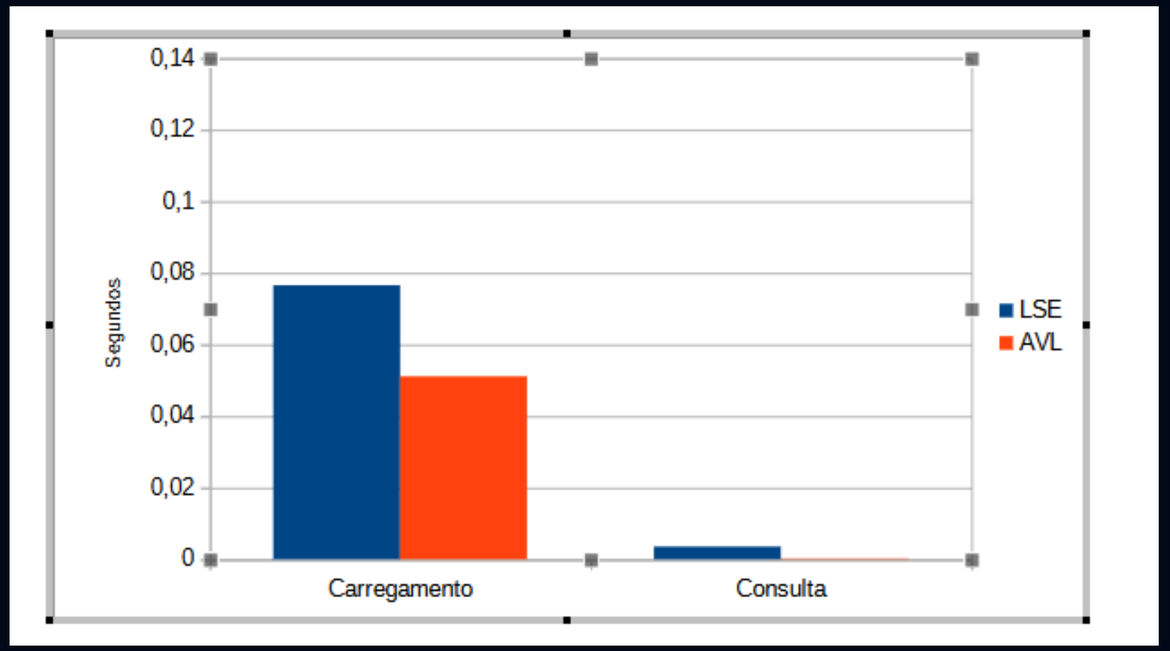
100.000 DADOS – TESTE COM 10 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.030211 segundos
Tempo de consulta LSE: 0.009990 segundos
Tempo de carregamento AVL: 0.116080 segundos
Tempo de consulta AVL: 0.000726 segundos
```

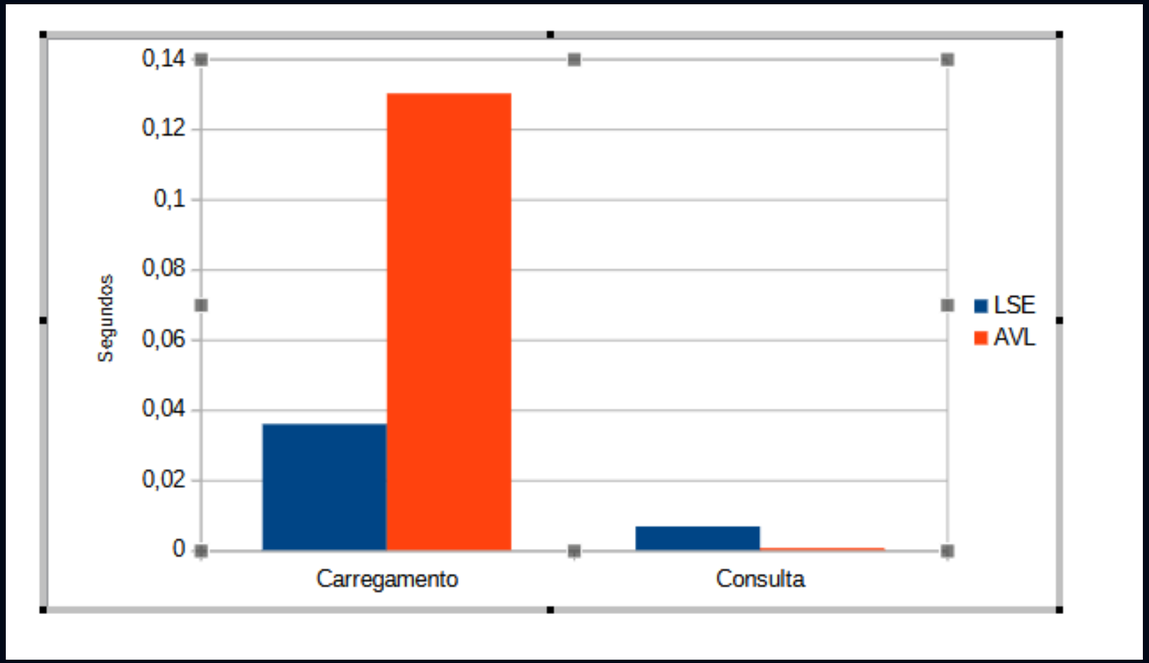
Ordenados:




```
Tempo de carregamento LSE: 0.076535 segundos
Tempo de consulta LSE: 0.003534 segundos
Tempo de carregamento AVL: 0.051098 segundos
Tempo de consulta AVL: 0.000180 segundos
```

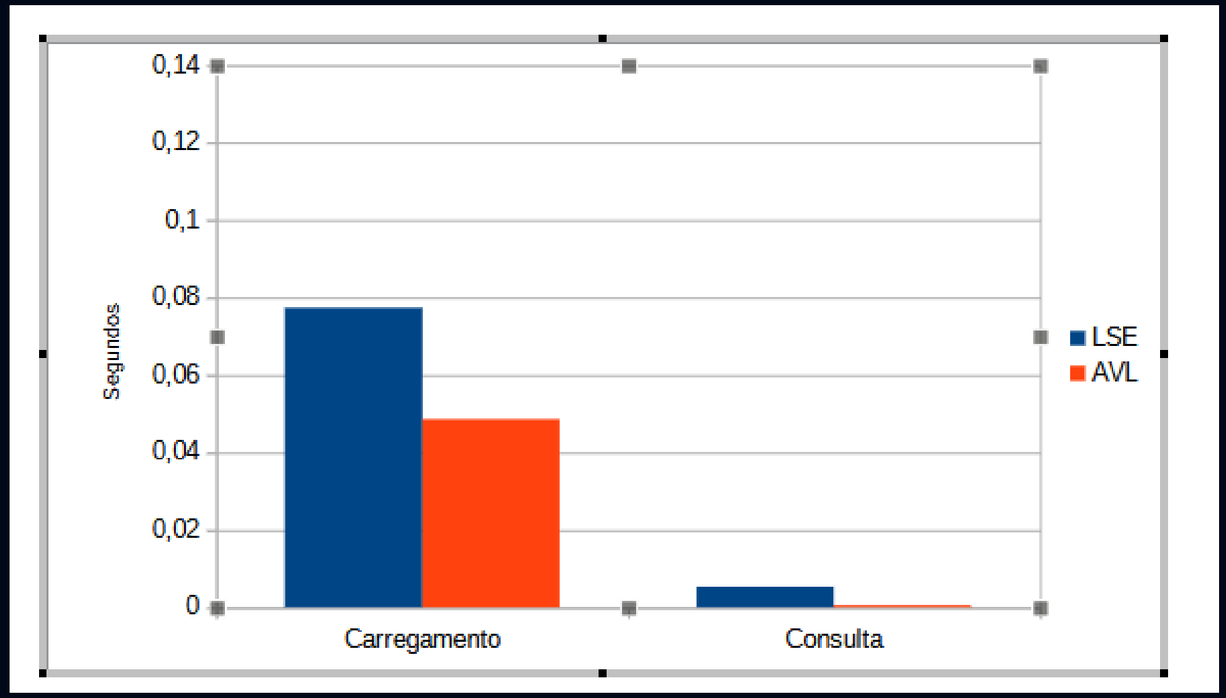
100.000 DADOS – TESTE COM 100 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.035912 segundos
Tempo de consulta LSE: 0.006733 segundos
Tempo de carregamento AVL: 0.130072 segundos
Tempo de consulta AVL: 0.000572 segundos
```

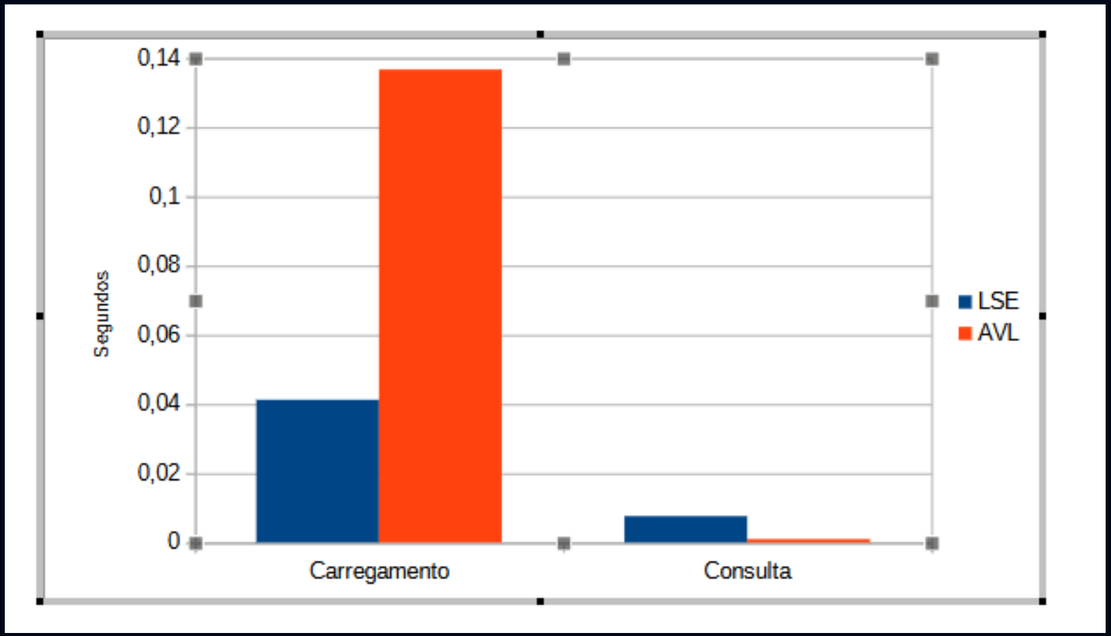
Ordenados:




```
Tempo de carregamento LSE: 0.077395 segundos
Tempo de consulta LSE: 0.005245 segundos
Tempo de carregamento AVL: 0.048535 segundos
Tempo de consulta AVL: 0.000560 segundos
```

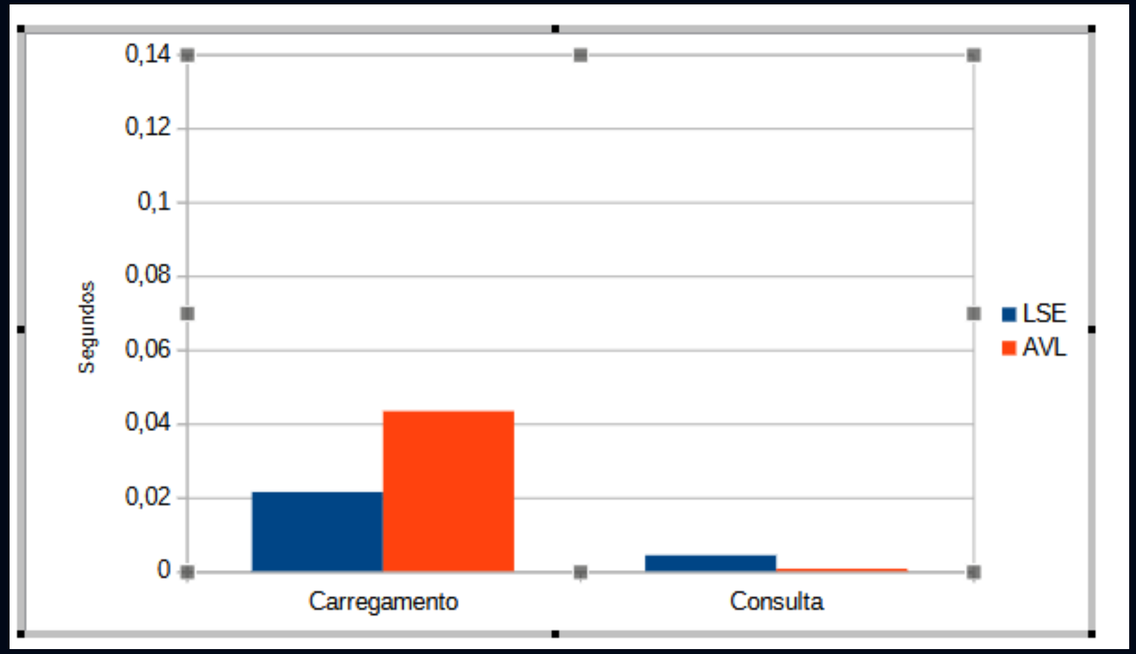
100.000 DADOS – TESTE COM 100 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.041225 segundos
Tempo de consulta LSE: 0.007625 segundos
Tempo de carregamento AVL: 0.136668 segundos
Tempo de consulta AVL: 0.000967 segundos
```

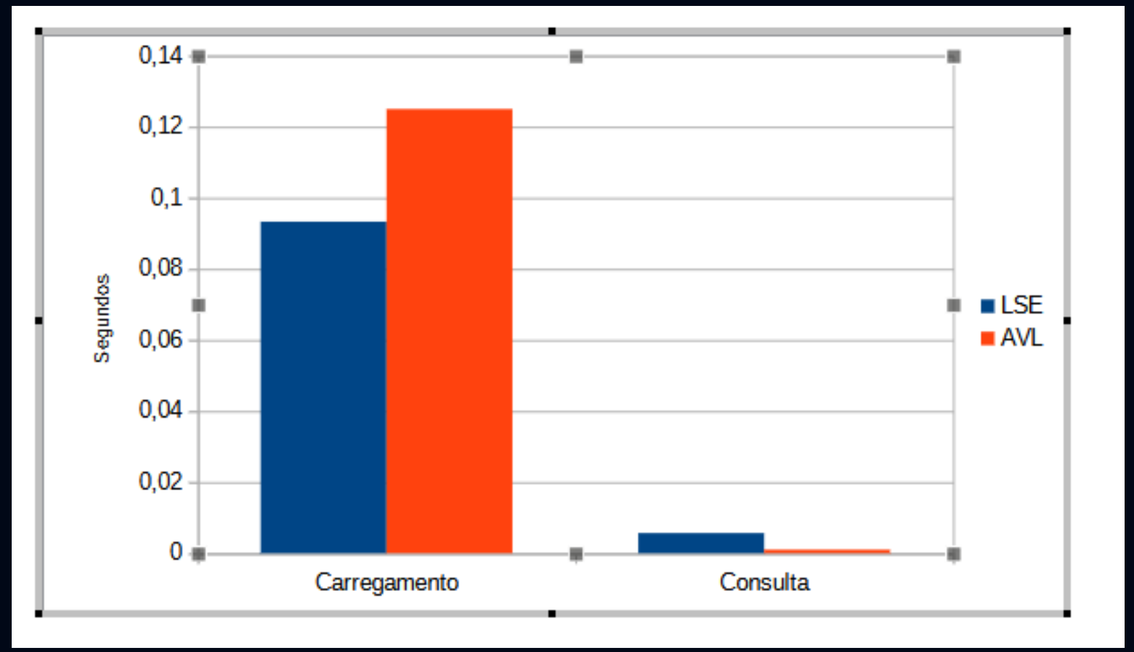
Ordenados:




```
Tempo de carregamento LSE: 0.021446 segundos
Tempo de consulta LSE: 0.004313 segundos
Tempo de carregamento AVL: 0.043343 segundos
Tempo de consulta AVL: 0.000683 segundos
```

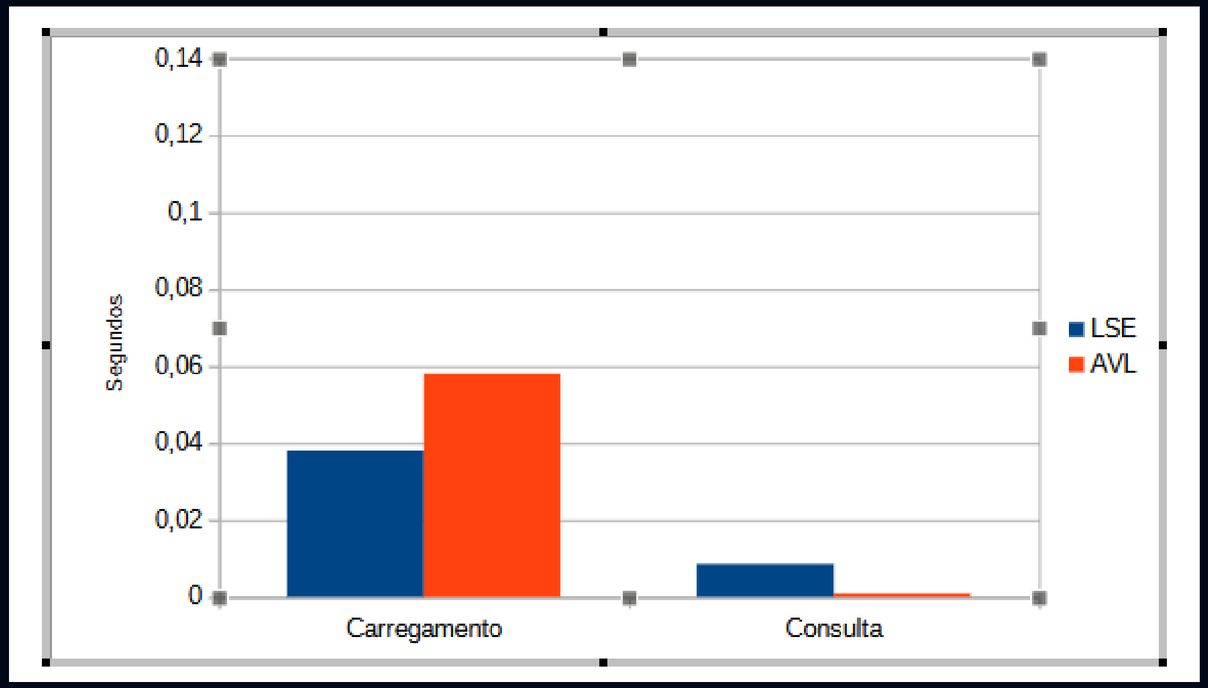
100.000 DADOS – TESTE COM 1000 USUÁRIOS– 10%

Não ordenados:




```
Tempo de carregamento LSE: 0.093272 segundos
Tempo de consulta LSE: 0.005652 segundos
Tempo de carregamento AVL: 0.124955 segundos
Tempo de consulta AVL: 0.000975 segundos
```

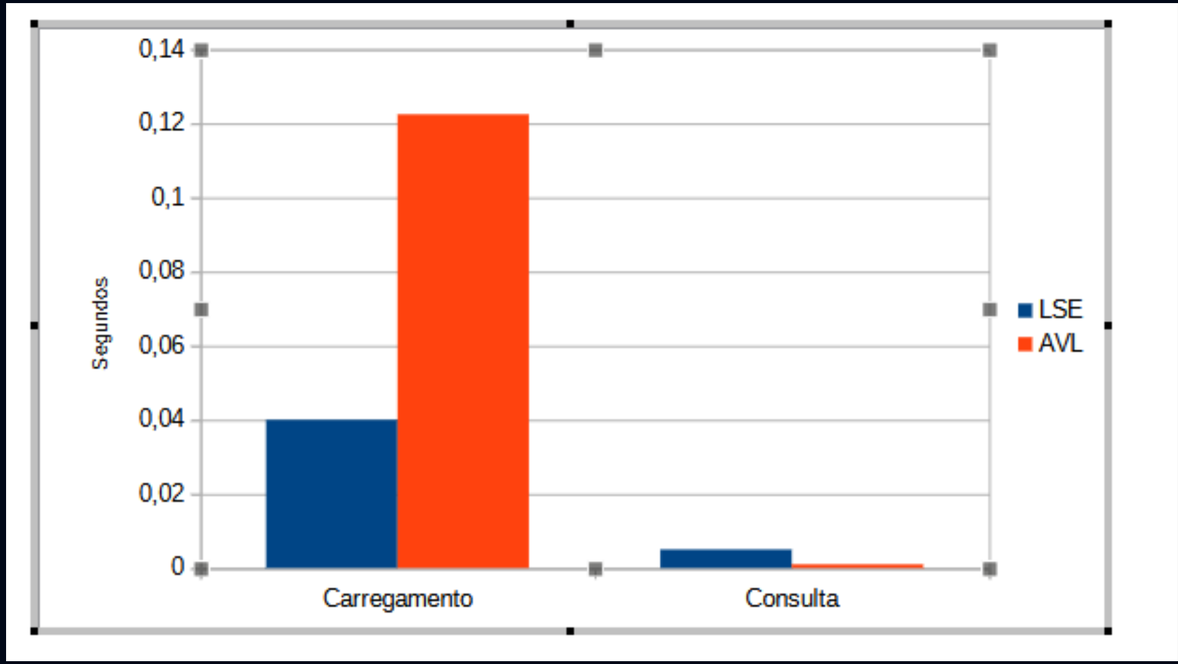
Ordenados:




```
Tempo de carregamento LSE: 0.038108 segundos
Tempo de consulta LSE: 0.008565 segundos
Tempo de carregamento AVL: 0.057974 segundos
Tempo de consulta AVL: 0.000925 segundos
```

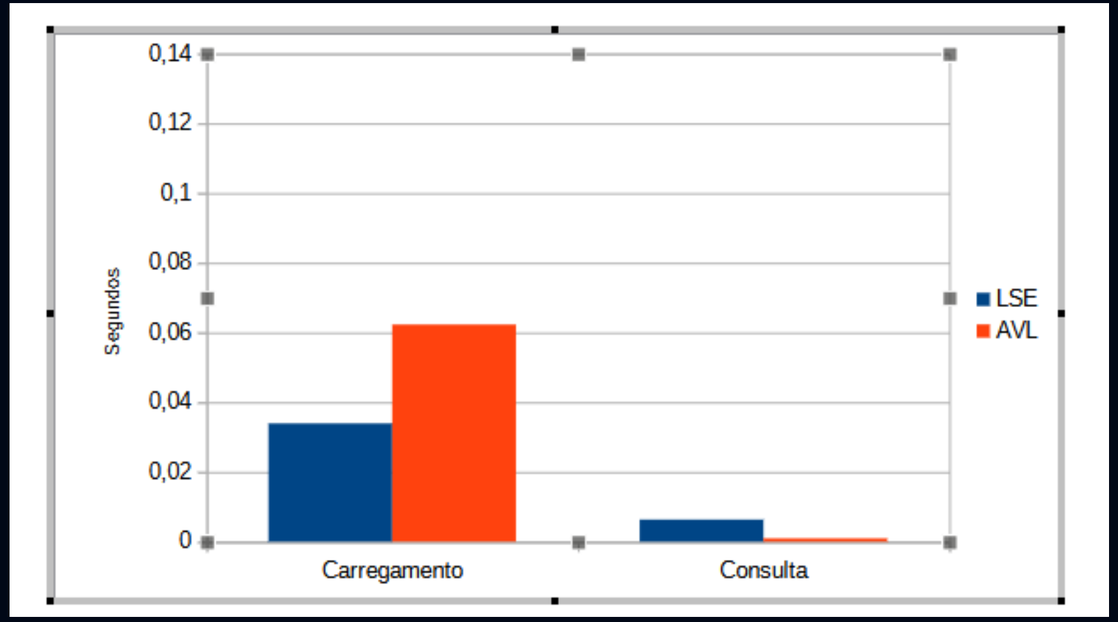
100.000 DADOS – TESTE COM 1000 USUÁRIOS– 50%

Não ordenados:




```
Tempo de carregamento LSE: 0.040006 segundos
Tempo de consulta LSE: 0.005006 segundos
Tempo de carregamento AVL: 0.122423 segundos
Tempo de consulta AVL: 0.000997 segundos
```

Ordenados:




```
Tempo de carregamento LSE: 0.033911 segundos
Tempo de consulta LSE: 0.006322 segundos
Tempo de carregamento AVL: 0.062260 segundos
Tempo de consulta AVL: 0.000933 segundos
```

Análise dos resultados

Após a realização de todos os testes tornou-se possível concluir alguns fatos:

- O tempo de carregamento dos dados da LSE em relação à AVL se mostrou significativamente menor durante os testes. Com dados maiores essa diferença é ainda mais expressiva.
- A AVL é incrivelmente mais rápida na consulta dos dados, principalmente devido às suas rotações que ocorrem na inserção. Essa característica torna a árvore mais demorada na operação de carregar os usuários, mas compensa na verificação de erros nas senhas.
- A diferença de carregamento entre as duas estruturas em relação à dados ordenados não é muito expressiva, tendo em vista que a AVL realiza um número menor de rotações.

Conclusão

Ao finalizar esse projeto, ficou claro que ambas as estruturas utilizadas são válidas para diferentes aplicações, sendo necessário levar em conta as facilidades que uma possui sobre a outra. A LSE possui uma facilidade muito maior de implementação, sendo recomendada para execuções que envolvem mais inserção do que consulta, visto que, como percebido nos testes, ela leva vantagem em relação à AVL. Um exemplo de aplicabilidade da LSE seria um sistema de gerenciamento de pedidos em um restaurante movimentado, onde cada pedido seria inserido rapidamente no final da lista.

Já a AVL leva vantagem para práticas que necessitam consultar um banco de dados grande a todo momento: a diferença conforme o aumento dos usuários mostra isso. Essa árvore pode ser utilizada, por exemplo, em um gerenciamento de estoque em um grande armazém ou empresa de comércio eletrônico. A necessidade de consulta a todo momento torna ela perfeita para essa situação.

Conclusão

As principais dificuldades encontradas ficaram em torno da aplicação do código da AVL e geração da base de dados. As rotações que ocorrem durante a inserção tornam a AVL mais complexa em comparação a LSE, sendo necessária a revisão constante do código durante o desenvolvimento do projeto.
A base de dados também se mostrou desafiadora para o trabalho. Como era uma grande quantidade de usuários, ficamos em dúvida inicialmente como poderíamos começar a geração das senhas. O GitHub Copilot nos auxiliou imensamente nessa parte, possibilitando que gerássemos ao todo 24 arquivos (incluindo os testes) para a análise de dados.