UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE COMPUTAÇÃO

ÍCARO FERNANDES, LUCAS BORGES, LUÍSA COELHO

ESTUDO EXPERIMENTAL DE MÉTODOS DE ORDENAÇÃO EXTERNA

Salvador
2024

SUMÁRIO

1. Descrição dos métodos estudados

1.1 Ordenação Balanceada Multi-Caminhos

A Intercalação Balanceada P-Caminhos é uma variação dos métodos de intercalação externa que busca otimizar o uso de recursos e melhorar a eficiência durante o processo de ordenação de grandes volumes de dados. Semelhante às abordagens Polifásica e Cascata, este método utiliza múltiplos arquivos intermediários para organizar os dados da sequência, mas adota uma estratégia distinta para a distribuição e intercalação das sequências ordenadas.

- Levando em consideração que temos $2p$ arquivos, sendo p de entrada e p de saída, intercalam-se arquivos de entrada gerando os de saída.

Essa ordenação utiliza o seguinte processo:

1. Distribui-se blocos ordenados de forma balanceada entre p arquivos;
2. Os bloco ordenados de cada arquivo de entrada são intercalados e distribuídos em blocos resultantes ordenados nos p arquivos de saída de forma balanceada;
3. Utilizar os arquivos de entrada como de saída e vice-versa.
4. Repetir esse processo até que todos os registros estejam em um único arquivo ordenado.

1.2 Ordenação Polifásica

A Intercalação Polifásica busca resolver dois problemas principais da Ordenação Balanceada Multi-Caminhos:

- a necessidade de um grande número de fitas; havendo f caminhos a intercalação balanceada necessita de $2f$ fitas, enquanto a polifásica utiliza apenas $f + 1$
- custo de uma cópia adicional do arquivo

Para isso, esse tipo de ordenação segue o seguinte processo:

1. Blocos ordenados são alocados de forma desigual nos arquivos disponíveis, de maneira que um é deixado vazio
2. O primeiro bloco de cada arquivo é intercalado na fita vazia. Caso um bloco termine antes, a intercalação segue apenas com os blocos restantes até que todos os valores de cada bloco tenham sido ordenados numa nova sequência na fita destino
3. Seguimos a intercalação para o segundo bloco de cada fita, depois o terceiro e assim por diante
4. Quando uma das fitas originalmente cheia fica vazia reiniciamos o processo de intercalação, tendo agora essa nova fita vazia como arquivo destino das sequências intercaladas

O nome desse tipo de ordenação vem justamente das muitas fases realizadas: cada intercalação do i-ésimo bloco de cada arquivo corresponde à fase i.

É importante ainda citar que podemos calcular a distribuição inicial ideal para minimizar o número de fases, ou seja, podemos prever a quantidade inicial de blocos necessária em cada arquivo disponível. Para isso, fazemos o processo reverso da ordenação polifásica: iniciamos os arquivos tais que apenas um contém uma única sequência e revertemos ao passo anterior até que o número total de blocos nas fitas seja maior ou igual ao número de blocos criados. Caso sejam necessários mais blocos do que temos para criar a configuração inicial ideal, são criadas as chamadas sequências falsas para ocupar os lugares que faltam.

1.3 Ordenação em Cascata

A principal diferença entre a Intercalação Polifásica e o Cascata é o fato de que este método visa passar por todos os registros salvos nos arquivos durante a execução de cada fase. Para isso, o método utiliza, a princípio, procedimentos similares ao método anterior: nós realizamos a intercalação entre a primeira sequência de cada arquivo, e vamos guardando o resultado num arquivo vazio (e sempre há, a qualquer momento da execução do algoritmo, ao menos um arquivo vazio). Uma vez que um dos arquivos é esvaziado, o antigo arquivo vazio (agora preenchido com sequências dos demais arquivos) é deixado de lado e então realizaremos o mesmo procedimento nos arquivos restantes, guardando os registros da intercalação no arquivo que foi esvaziado. Executamos esse procedimento até que nós tenhamos passado por todos os registros (e portanto todos os arquivos tenham sido esvaziados em algum momento durante cada fase), até que reste uma única sequência ordenada, que será a saída do algoritmo.

Tal qual o algoritmo de ordenação por intercalação polifásica, distribuir as sequências ordenadas de forma aleatória ou uniforme entre os arquivos não é ideal pois provavelmente haverá mais de um arquivo vazio em dado momento, e estaremos desperdiçando arquivos que poderiam estar sendo utilizados para uma execução mais eficiente do algoritmo (como veremos da seção de análise experimental, o uso de mais arquivos tende a diminuir o número de operações de escrita por registro). Dessa forma, é inteligente que calculemos a distribuição ideal das sequências ordenadas de forma que o único momento no qual teremos mais de um arquivo vazio simultaneamente seja o final da execução, quando há somente uma sequência ordenada restante. Para isso, pode ser necessário a criação de sequências falsas, que não farão parte da saída.

2. Descrição dos experimentos e implementação dos métodos

2.1 Execução dos testes

Para medir o esforço realizado pelo algoritmo em função do número r de sequências ordenadas iniciais, e como o número de arquivos utilizados influencia esse quesito, utilizamos o cálculo do fator α, definido da seguinte forma:

Seja n o número de registros da entrada, e x o número de operações de escrita sobre os arquivos, $\alpha = \frac{x}{n}$. Nós realizamos os testes da seguinte forma:

Geramos 10 entradas aleatórias, com $n = 5000$. Então, seja $K = \{4, 6, 8, 10, 12\}$ e $R = \{i \cdot j \leq 5000 | i \in \{1, \dots, 10\}, j \in \{10, \dots, 1000\}\}$, realizamos 10 testes para cada par $(k, r) \in K \times R$. Em cada teste, selecionamos r sequências ordenadas de uma das 10 entradas aleatórias utilizando a nossa implementação da seleção natural, e executamos o algoritmo com k arquivos. Ao fim de cada execução, calculamos o fator alpha e, após as 10 execuções para um dado par (k, r), calculamos a média aritmética dos fatores alpha, e esse é o valor registrado.

2.2 Implementação da Ordenação Balanceada P-Caminhos

Para implementação do algoritmo de ordenação Balanceada P-Caminhos foram criados três classes, sendo as seguintes:

Classe **`Balanceada,`** principal classe do arquivo em que constam os métodos essenciais para a ordenação balanceada, sendo os seguintes:

1. **balancear**
   Essa função é responsável por gerir os arquivos, seguindo o processo descrito no item 1.1: inicia com p arquivos de entrada e p arquivos de saída, enquanto todos os arquivos de entrada não estiverem vazios, aloca o primeiro bloco ordenado de cada arquivo para a função de intercalação que será explicada posteriormente.
   Após a intercalação entre esses blocos, teremos um novo bloco ordenado que será escrito em um arquivo de saída. Esse processo será repetido de forma balanceada entre os arquivos de saída.
   Ao final do processo, a referência entre os arquivos de entrada e saída são invertidas. Concluindo uma fase do balanceamento.
2. **intercalar**
   Esse método é responsável por intercalar as sequências ordenadas gerando uma nova e única sequência ordenada. Muito parecido com o funcionamento com o algoritmo merge sort, comparando o primeiro elemento de cada bloco, identificando o menor e o inserindo na nova sequência, esse processo é repetido até que as listas iniciais estejam vazias.

O restante dos métodos funcionam para controle de atributos e cálculo do β de cada fase e do α ao final do processo de Balanceamento.

Classe **ArquivoBalanceada,** responsável pelo controle de cada fita e armazenar as sequências, além de guardar informações de escrita.

Classe **DistribuidorBalanceada,** responsável pela distribuição das sequências iniciais entre os arquivos de entrada. Essa distribuição é feita de forma balanceada, ou seja, intercalando entre os arquivos de entrada qual será escrito a próxima sequência inicial.

2.3 Implementação da Ordenação Polifásica

Para garantir o funcionamento do código foram feitos experimentos de forma simples, apenas gerando valores aleatórios para os parâmetros necessários e imprimindo cada fase no terminal.

Com relação à implementação do método, primeiro foi criada a classe **Polifasica**, nela podemos encontrar os dois principais métodos que efetivamente realizam a ordenação polifásica:

1. **polifasear**
   Essa função representa uma fase da ordenação polifásica e segue o processo descrito no item 1.2: encontra o arquivo vazio, determina quais sequências devem ser intercaladas e as passa para a função 2, adiciona o bloco intercalado na fita vazia, retira os blocos já intercalados de cada arquivo e pára quando uma das fitas originalmente cheia fica vazia.
2. **ordenarSequencias**
   Essa função recebe uma lista com blocos e retorna a intercalação de todos em uma única sequência. Para o processo de intercalação é utilizado um vetor index, onde `index[i]` representa a posição do próximo valor da lista i a ser comparado.

Para facilitar o manejo dos blocos de cada fita foi criada a classe **ArquivoPolifasica**, que guarda todas as informações necessárias e implementa algumas propriedades.

Depois de garantir o funcionamento da ordenação, foram adicionadas as métricas necessárias para análises posteriores como o cálculo de alpha (**`setAlpha`**) e beta (**`avgSeqSize`**) além de formatado o output da ordenação, como estabelecido na definição do trabalho.

Ademais, foi implementado o **`DistribuidorPolifasica`**, que calcula a distribuição inicial de sequências ideal e está estruturado da seguinte forma:

1. **`firstStep`**
   Transforma a configuração final $[0, 0, ..., 0, 1]$ na penúltima configuração $[1, 1, ..., 1, 0]$
2. **`nextStep`**
   Soma o tamanho do maior arquivo (o tamanho de um arquivo é determinado pela quantidade de blocos nele) ao tamanho dos outros e zera o maior arquivo
3. **`reversePolifase`**
   Aplica o primeiro passo e continua aplicando `nextStep` até que o total de sequências seja maior ou igual ao número de sequências desejado
4. **`criarArquivos`**
   Aplica **`reversePolifase`** e cria k arquivos seguindo a distribuição desejada. Aloca primeiro os blocos existentes e depois cria sequências falsas se necessário

2.4 Implementação da Ordenação em Cascata

Assim como nos outros métodos, processamos a entrada utilizando a seleção natural para gerar as sequências iniciais.

Nós começamos, partindo de uma única sequência ordenada, e então simulamos a execução do algoritmo reverso para descobrir a configuração dos k arquivos na fase anterior para que a configuração de uma única sequência ordenada seja obtida. Repetimos esse procedimento até que o número total de sequências seja maior ou igual ao número de sequências iniciais da entrada, e distribuímos as sequências de acordo. Caso o número de sequências seja estritamente maior que o número de sequências da entrada, injetamos sequências falsas para adequar o número de sequências. Na saída da execução do programa de correção do trabalho, as sequências falsas são representadas como “{*}”.

Uma vez que as sequências estejam distribuídas, iniciamos as intercalações. A cada fase, marcamos o arquivo vazio como o arquivo-alvo, e vamos intercalando as sequências dos demais arquivos e escrevendo o resultado no arquivo-alvo. Quando um dos demais arquivos é esvaziado, o arquivo esvaziado passa a ser o novo arquivo alvo, e o antigo alvo é congelado, e não mudará seu estado até que a fase termine. Nós repetimos o procedimento até que todos os arquivos que não o alvo estejam congelados, ponto no qual nós descongelamos todos os arquivos e iniciamos uma nova fase se ainda não atingimos a configuração desejada (uma única sequência).

2.5 Seleção Natural

Para gerar as sequências iniciais para uma dada entrada, utilizamos o método da seleção natural. Ele consiste em utilizar uma heap mínima de tamanho m para gerar as sequências. Nós vamos inserindo os registros da entrada na árvore heap, e, quando ela fica totalmente preenchida, vamos realizando a remoção do valor do topo e acrescentando à sequência que está sendo formada, e então

adicionando um novo registro da entrada à heap. Se o novo registro for menor que o último registro adicionado à sequência em formação, ele é marcado, e registros marcados são considerados maiores que os não marcados. Quando removemos um registro marcado da heap, isso indica que todos os registros da heap estão marcados, então encerramos a sequência atual, desmarcamos todos os registros da heap, e inserimos o registro removido numa nova sequência. Repetimos esse processo até não haverem mais registros na entrada.

Para quantificar a eficiência deste método de geração de sequências iniciais, utilizamos o fator beta. Seja n o número de registros da entrada, j o número de fases realizadas desde o início da execução do algoritmo, m o tamanho da memória principal, e r_j o número de sequências ordenadas presentes na fase j da execução do algoritmo, o fator beta é definido por:

$$\beta(m, j) = \frac{n}{r_j m}$$

Para cada $m \in \{3, 15, 30, 45, 60\}$, nós realizamos o teste com 10 entradas de 1000 registros aleatórios, e registramos o valor médio dos resultados obtidos para $\beta(m, 0)$.

Durante os testes que requerem um valor r fixo de sequências iniciais, e, para configuração de entrada e saída especificada no escopo do trabalho, onde um r fixo é especificado, implementamos a seleção natural de modo a forçar a seleção de exatamente r sequências iniciais:

Durante a seleção natural, se o valor r de sequências foi atingido, o resto da entrada é descartado.

Se a seleção natural termina e não foi atingido o valor r de sequências, quebramos sequências na metade até que tenhamos obtido r sequências. Observe que, para isso, é necessário $n \geq r$. Quando $n = r$, a seleção quebra as sequências na metade até que tenhamos r sequências de um registro. Mas, se $n < r$, não é possível gerar r sequências a partir da entrada em questão, e o programa lança um erro.

2.6 APIs

Para facilitar a criação do programa e garantir que todos os integrantes do grupo seriam capazes de rodar e testar todas as implementações dos métodos de ordenação de maneira rápida e fácil (sem precisar entender linhas e linhas do código de outra pessoa), cada um fez uma API com as mesmas funções que roda o seu respectivo método.

2.7 Programa

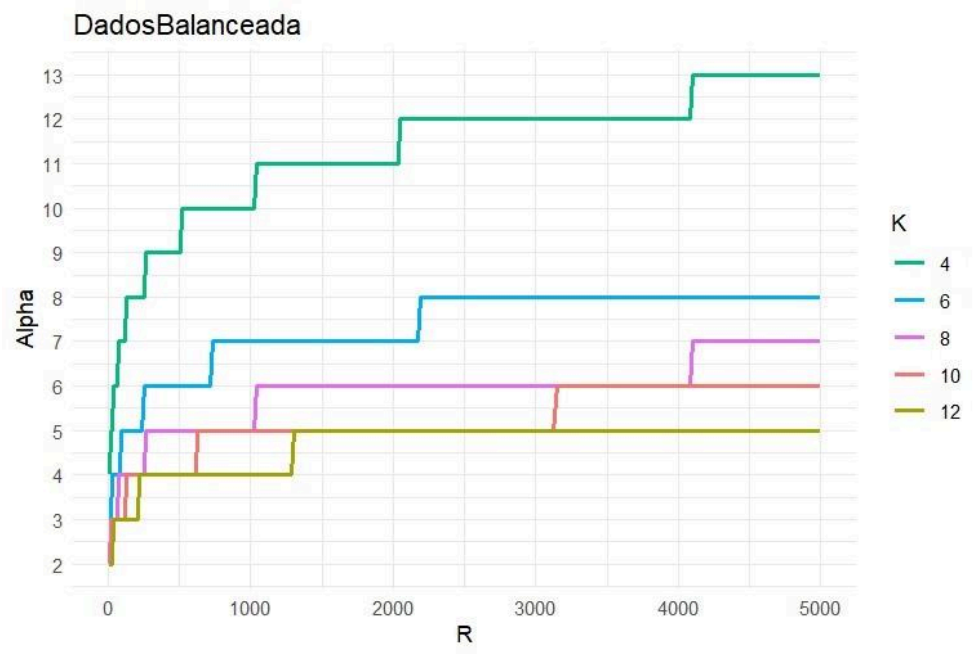
A construção do arquivo que contém o programa principal foi extremamente simples e rápida por conta das APIs criadas anteriormente. Foi feito o upload da aplicação no Replit (aqui ou https://replit.com/@LucasTBorges/OrdenacaoExterna), para testá-la é preciso fazer login com o email gmalima@gmail.com para o qual foi dada a permissão necessária. Depois, basta pressionar o botão ‘Run’ e a página automaticamente executará o arquivo ‘Programa.py’. Enfim podem ser colocadas as entradas da maneira determinada pela especificação do trabalho: a primeira linha contém o método a ser considerado (B, P ou C), a segunda fornece os valores de m, k, r e n, nesta ordem, e a terceira e última linha contém os n valores inteiros a serem considerados para a ordenação.

A saída também foi feita de acordo com as especificações, indicando as fases de ordenação, os valores de $\beta(m, j)$ correspondentes a cada fase, as sequências sendo ordenadas e, por fim, o valor encontrado para α(r). Nas ordenações Polifásica e em Cascata, as sequências falsas são representadas por '{*}'.

3. Resultados experimentais

3.1 Ordenação Balanceada P-Caminhos

Os dados colhidos do teste executado no item 2.2 está representado no gráfico a seguir:

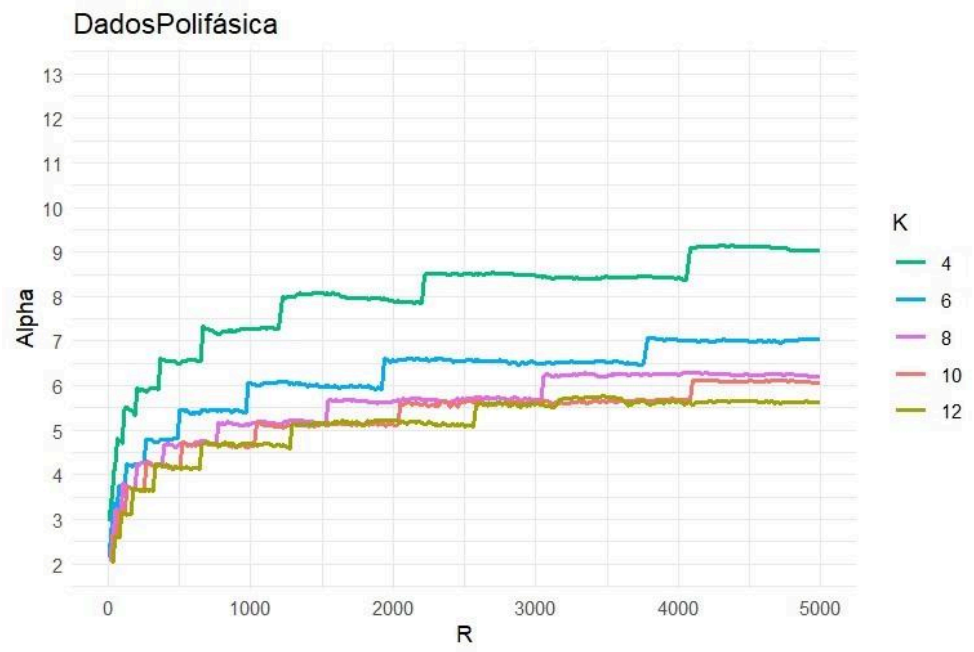


Com a análise do gráfico, observamos que existe uma relação inversa entre o valor de α e o número de arquivos utilizados. Isso demonstra que quantos mais fitas forem utilizadas no processo de ordenação, menor será o esforço α pelo programa.

Além disso, notamos o crescimento do valor de α em função de r, porém esse crescimento é feito por blocos, mantendo-se estável nesses blocos até o próximo salto.

3.2 Ordenação Polifásica

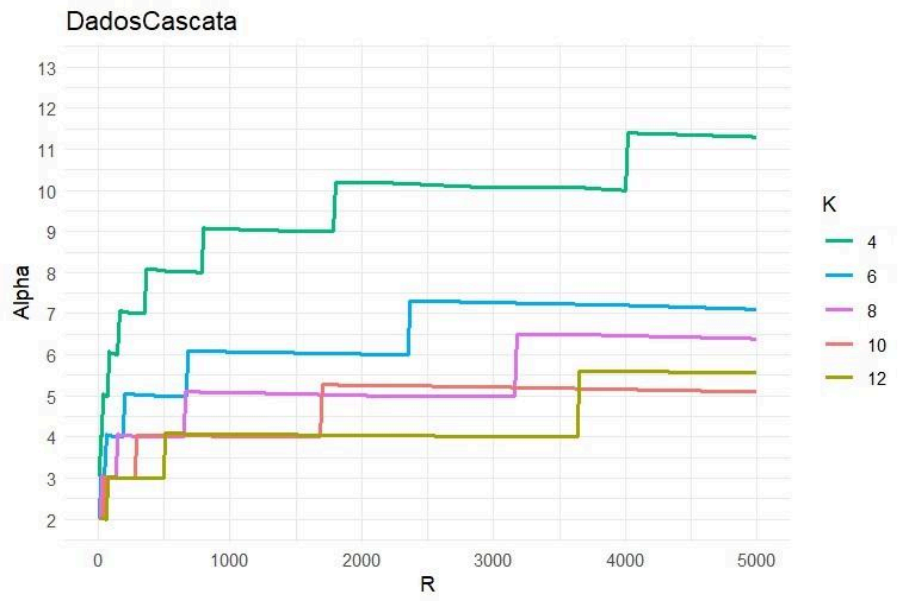
Os dados colhidos do teste executado no item 2.3 está representado no gráfico a seguir:



Pelo Gráfico 1 vemos claramente que há uma relação inversa entre o esforço α e a quantidade de arquivos: no geral, quanto mais fitas disponíveis menor o α. Além disso, percebemos claramente que a quantidade de sequências iniciais aumenta o esforço do algoritmo em intervalos, o que indica que a quantidade de fases para completar a ordenação aumenta em pontos específicos, precisando de cada vez mais aumento em r para sofrer outro "salto".

3.3 Ordenação por Intercalação em Cascata

No gráfico a seguir, temos os resultados observados nos testes executados conforme descrito na seção 2.4:



Podemos observar no gráfico o crescimento do fator alpha em função de r. Observemos que, no geral, um maior número de arquivos resulta num esforço

menor por parte do algoritmo. O alpha se mantém relativamente estável durante um intervalo de quantidade de sequências ordenadas no qual o número de fases necessárias para completar a execução do algoritmo é o mesmo. Entretanto, é possível notar saltos em alpha nos pontos em que passamos a ter sequências ordenadas o suficiente para precisarmos de mais uma fase do algoritmo para realizar a ordenação. Vemos que, quanto maior o valor de k, maior é o número de sequências necessárias antes de realizar cada salto.

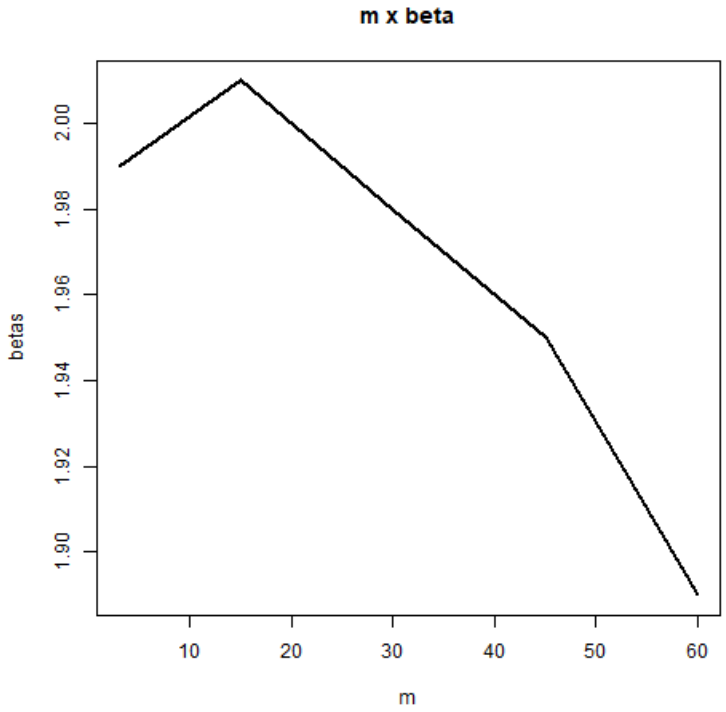
É possível notar que, no caso do cascata, a valor de alpha tende a se estabilizar na proximidade de um número inteiro, e então realizar o próximo salto de aproximadamente 1 unidade. Especulamos que isso se deva à natureza do algoritmo, que é feito de forma que, em cada fase, passamos por todos os registros exatamente uma vez, então a tendência é que o número de operações de escrita envolvendo cada registro seja o mesmo: igual ao número de fases da execução do algoritmo.

Vemos que o esforço realizado por esse método, em comparação com o método de intercalação polifásica, foi, em média, maior. Entretanto, à medida que aumentamos k e r, o fator alpha se aproxima cada vez mais ao fator alpha do método anterior para um mesmo par (k, r), e a tendência observada é que, para k e r suficientemente grandes, o algoritmo de ordenação por intercalação em cascata supere a intercalação polifásica no quesito eficiência (menor número de operações de escrita por registro).

3.4 Seleção Natural

Para avaliar a variação do tamanho das sequências iniciais em função da memória disponível, foi criado o gráfico abaixo (fixamos o n em 1000 e geramos 10 testes para cada valor de m, e registramos o resultado médio):

Aqui podemos observar que $\beta(m, 0)$ é inversamente proporcional a m, (apesar de nesse caso o valor de beta cair lentamente, possivelmente pela escolha de n grande em comparação aos valores m observados), indicando que o crescimento do tamanho médio das sequências iniciais não acompanha o crescimento de m. A tendência é que o fator beta caia até atingir o valor 1 quando o tamanho médio das sequências for igual ao tamanho de m, o que acontece com certeza quando $m = n$, pois teremos uma única sequência ordenada inicial do tamanho de n. Portanto, temos que $\beta(n, 0) = 1$. Então nesse caso, poderíamos estimar que $\beta(1000, 0) = 1$, pois fixamos o tamanho da entrada em 1000. Ademais, se continuarmos aumentando o tamanho da memória principal para além do tamanho da entrada, não obteremos acréscimo no tamanho da única sequência inicial, que continuará sendo n (o tamanho da entrada). Assim, para $m \geq n$, $\beta(m, 0) = \frac{n}{m}$ e, portanto, $\beta(m, 0)$, deve tender a 0 quando m tende a $+\infty$. Entretanto, é importante ressaltar que, para $m \geq n$, o método da Seleção Natural já ordena toda a entrada, sendo equivalente ao HeapSort. Como o objetivo dos algoritmos estudados nessa disciplina é realizar a ordenação nos casos em que a memória principal não comporta toda a entrada, não faz muito sentido tratarmos dos casos em que $m \geq n$.



4. Considerações Finais

Todos os métodos estudados são muito mais eficientes no quesito de minimizar as os acessos à memória secundária se comparado com os métodos tradicionais estados na disciplina de EDA I, como o Quicksort. Entretanto, podemos observar, pelos resultados obtidos através da avaliação experimental, que a Ordenação Balanceada por P-caminhos realiza, no geral, um maior número de operações de escrita por registro quando comparado aos demais métodos de ordenação externa estudados na disciplina. Os métodos de ordenação por intercalação Polifásica e em Cascata fazem melhor proveito da quantidade de arquivos disponíveis, ocupando $k - 1$ arquivos com as sequências, enquanto o primeiro método visto distribui as sequências em apenas $\frac{k}{2}$ dos k arquivos disponíveis.

Ademais, dentre o Polifásica e o Cascata, observamos que, para a maior parte dos valores observados de r e k, o Cascata realiza mais operações de escrita por registro. Entretanto, à medida que os valores de r e k aumentaram, essa diferença foi diminuindo, e estima-se que a tendência é que para r e k suficientemente grandes, o Cascata atinja o mesmo desempenho que o Polifásica e, possivelmente, ultrapasse.