Trabalho Prático 3 - Árvore de Segmentação

João Correia Costa (2019029027)

Dezembro de 2023, Belo Horizonte

1 Introdução

Árvores de Segmentação são estruturas de dados versáteis utilizadas em ciência da computação para lidar eficientemente com várias tarefas de consulta de intervalo, em inglês, range-query. É particularmente útil em cenários nos quais você precisa realizar funções agregadas ou operações de atualização em um subvetor de elementos.

A ideia central por trás de uma árvore de segmentos é representar um vetor dado como uma árvore binária, onde cada nó folha corresponde a um elemento individual no vetor. Os nós internos da árvore armazenam informações agregadas (como a soma, mínimo, máximo, etc.) de seus respectivos nós filhos. Essa estrutura hierárquica permite consultas eficientes de intervalo e atualizações.

A construção de uma árvore de segmentos envolve a divisão recursiva do vetor em segmentos menores até que cada segmento represente um único elemento. Durante operações de consulta, a árvore é percorrida para encontrar os segmentos relevantes que contribuem para o resultado final, levando a soluções eficientes.

Essa abordagem é particularmente útil em cenários nos quais o vetor de entrada é estático, e a maioria das operações envolve consultas ou atualizações em intervalos específicos, pois oferece um equilíbrio entre complexidade de tempo e espaço.

No contexto mencionado, o presente trabalho busca implementar uma Árvore de Segmentação construída sobre um vetor de entrada que contém matrizes $A_{2\times2}$. Cada matriz representa uma transformação vetorial no espaço $2D$, e um subvetor específico do vetor de entrada representa uma sequência de transformações vetoriais que pode ser sintetizada em uma única matriz $R_{2\times2}$, resultante do produto de todas as matrizes no intervalo considerado. Em outras palavras, a transformação vetorial associada ao subvetor do índice i ao j é dada por $A_i \times \ldots \times A_j = R_{ij}$. Neste sentido, os nós internos da árvore de segmentação contém matrizes resultantes R_{ij} do produto de uma sequência de matrizes do segmento, e os nós folha contém matrizes simples, associada a uma transformação vetorial apenas.

Estamos buscando calcular a matriz de transformação R_{ij} associada a um determinado subvetor de matrizes, do índice i ao j, que será aplicada a um vetor 2D (cord_x, cord_y) fornecido. Essa matriz nos permitirá prever qual será o novo vetor resultante após a aplicação da transformação.

2 Método

A entrada de dados é composta por dois inteiros: n, que representa o tamanho do vetor de matrizes (ou seja, o número de matrizes de transformação), e q, indicando o número de operações a serem realizadas. Em seguida, as operações são descritas, uma por linha. Para consultas, é lido um caractere q seguido por quatro inteiros t_0, t_1, $cord_x$, $cord_y$, indicando os índices do subvetor de matrizes a ser considerado na consulta e as coordenadas do ponto (x, y) a ser transformado. Apenas os 8 dígitos menos significativos do ponto resultante após a transformação são impressos. Para operações de atualização, é lido um caractere u, seguido por um inteiro a, representando a posição no vetor de matrizes a ser alterada. As próximas linhas leem uma matriz 2D que deve substituir a matriz na posição a do vetor. A operação de atualização não possui nenhuma saída esperada.

Duas soluções foram implementadas. A primeira consiste em armazenar cada uma das transformações em um arranjo indexado. A segunda estratégia envolve a implementação de uma Árvore de Segmentação que armazena matrizes de transformação pré-computadas em seus nós.

2.1 Vetor Simples de Matrizes

Uma abordagem simplificada para o problema seria armazenar cada uma das transformações em um vetor indexado de matrizes. Para a operação de consulta, considerando os instantes (índices) i e j, seria suficiente multiplicar todas as matrizes nos índices de i a j. No pior caso, essa abordagem tem complexidade $O(n)$. Para as operações de atualização, bastaria modificar a matriz na posição i, o que é feito em tempo constante $O(1)$. Ver Figura 1.

2.2 Árvore de Segmentação

A segunda abordagem adotada para resolver o problema consiste na implementação de uma Árvore de Segmentação que armazena matrizes de transformação pré-computadas em seus nós. Essa estratégia tem como objetivo otimizar a execução das operações de consulta.

2.2.1 Construção da Árvore

A construção da Árvore de Segmentação é um processo realizado de maneira recursiva, onde cada nó interno da árvore representa um intervalo do vetor de matrizes. O método utilizado para implementar essa construção é denominado `SegTree::build`.

No início do processo, a raiz da árvore é configurada para representar o intervalo completo do vetor de matrizes. Cada nó interno da árvore armazena o produto das matrizes dentro do intervalo correspondente. A construção ocorre de forma recursiva até que cada nó folha contenha uma única matriz de transformação.

No código fornecido, a função `SegTree::build` recebe quatro parâmetros: o índice do nó atual p, os limites esquerdo e direito do intervalo l e r, e um ponteiro para um array de matrizes `array`.

A construção da árvore é realizada da seguinte maneira:

- Se o intervalo l é igual a r, ou seja, estamos em um nó folha, a função retorna a matriz correspondente a esse índice no array.
- Caso contrário, calcula-se o ponto médio m do intervalo e continua a construção para os filhos esquerdo e direito de forma recursiva.
- As matrizes obtidas dos filhos são multiplicadas para obter a matriz resultante representando o intervalo atual.
- A matriz anteriormente armazenada no nó atual é liberada, e o nó atual é atualizado com a nova matriz resultante.

Essa abordagem garante que a árvore seja construída de forma eficiente, armazenando produtos de matrizes nos nós internos e mantendo a consistência ao longo da estrutura da árvore.

2.2.2 Consulta na Árvore

Na etapa de consulta na Árvore de Segmentação de Matrizes, o procedimento segue uma lógica sequencial.Ao iniciar a consulta, verifica-se se o nó atual na árvore possui alguma interseção com o intervalo desejado. Se não houver interseção, a matriz identidade é retornada como resultado, indicando que não é necessário continuar a exploração nesse ramo específico da árvore.

Em seguida, examina-se se o nó atual está completamente contido no intervalo da consulta. Caso afirmativo, a matriz armazenada nesse nó é retornada integralmente como resultado, eliminando a necessidade de buscar mais informações nesse ramo da árvore.

Se o nó atual apresentar apenas uma interseção parcial com o intervalo da consulta, o processo se ramifica. A consulta é repetida para o filho esquerdo e para o filho direito do nó atual. As respostas obtidas desses filhos são então combinadas multiplicando as matrizes correspondentes.

Por fim, a resposta final é representada pela matriz resultante da combinação das respostas dos filhos. Dessa maneira, a consulta na Árvore de Segmentação de Matrizes é conduzida de forma eficiente, explorando apenas os ramos relevantes e utilizando a matriz identidade como um elemento neutro para casos específicos. A consulta na árvore foi implementada no método `SegTree::query`.

2.2.3 Atualização na Árvore

Na etapa de atualização na Árvore de Segmentação de Matrizes, o processo inicia-se pela localização do nó correspondente à posição a no vetor de matrizes. Esse nó é identificado para representar a matriz que será atualizada. Posteriormente, realiza-se a substituição da matriz armazenada nesse nó pela nova matriz fornecida como entrada, efetuando a atualização na posição específica do vetor de matrizes.

O próximo passo envolve uma propagação recursiva em direção aos nós pais. Em cada nó pai ao longo do caminho até a raiz da árvore, recalcula-se a matriz armazenada no nó pai com base nas novas matrizes dos filhos. Essa propagação recursiva é executada até alcançar a raiz da árvore, assegurando a consistência das matrizes nos nós pais.

Dessa forma, a operação de atualização garante a substituição adequada da matriz na posição a do vetor e mantém a integridade da representação das transformações ao longo da estrutura da árvore. A atualização na árvore foi implementada no método `SegTree::update`.

Com essa implementação, busca-se melhorar o desempenho, especialmente em cenários nos quais as operações de consulta são frequentes em comparação com as operações de atualização.

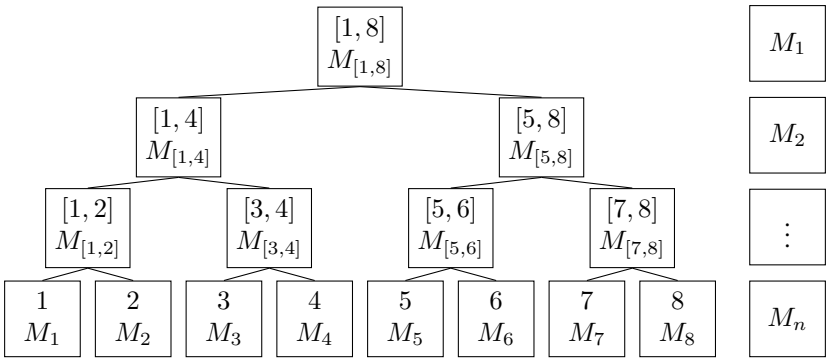

Figure 1: *Árvore de Segmentação à esquerda e Vetor Ingênuo à direita.*

3 Análise de Complexidade

Essa seção se dedica a analisar a complexidade assintótica, em termos de espaço e de tempo, das funções de consulta, atualização e construção na árvore de segmentação e no vetor simples de matrizes.

3.1 `SegTree::build`

- A operação de construção da árvore de segmentação (`SegTree::build`) pode ser convenientemente descrita de forma recursiva, percorrendo a árvore a partir do vértice raiz até os vértices folha.
- O procedimento de construção, quando chamado em um vértice não folha, realiza o seguinte:
  1. Constrói recursivamente as matrizes dos dois vértices filhos.
  2. Multiplica a matriz dos vértices filhos para obter a matriz do nó.
- O início da construção ocorre no vértice raiz, permitindo a computação da árvore de segmentação completa.
- A complexidade de tempo dessa construção é $O(n)$, onde n é o número de nós na árvore, considerando que a operação de multiplicação (`Matrix* operator*(const Matrix& other)`) é de tempo constante (a operação de multiplicação é chamada n vezes, que é igual ao número de nós internos na árvore de segmentação).

3.2 `SegTree::query`

A consulta opera dividindo o segmento de entrada em vários subsegmentos, para os quais todas as matrizes já foram previamente calculadas e armazenadas na árvore. A eficiência característica da Árvore de Segmentação consiste na possibilidade de interromper a consulta no ramo sempre que o nó coincide com o range de busca ou está contido nele. No pior caso temos custo $O(\log n)$, onde n é o número de nós na árvore.

3.3 `SegTree::update`

Quando desejamos modificar um elemento específico no vetor, por exemplo, realizar a atribuição $vetor[i] = M$, é necessário reconstruir a Árvore de Segmentação de modo a corresponder ao novo array modificado. Todo o ramo associado ao índice i é afetado, então caminhamos da folha até a raiz recalculando as matrizes nos nós, temos custo $O(\log n)$.

3.4 `SimpleVector::build`

Aloca-se espaço para o vetor e percorre cara índice sequenciamente adicionando uma matriz identidade, tem-se custo linear $O(n)$.

3.5 `SimpleVector::update`

Apenas deletamos a matriz no índice i de atualização e substituímos por uma nova matriz. Temos custo constante $O(1)$.

3.6 `SimpleVector::query`

Varremos o vetor sequencialmente entre os índices de busca i e j, multiplicando as matrizes e armazenando em uma matriz resultante, tem-se custo $O(n)$.

3.7 `SimpleVector::build`

Varremos o vetor sequencialmente entre os índices de busca i e j, multiplicando as matrizes e armazenando em uma matriz resultante, tem-se custo $O(n)$.

4 Estratégias de Robustez

Com o objetivo de tornar o programa mais robusto e evitar problemas com entradas inválidas, foram criadas classes de exceção `MatrixMultiplicationException` . Essa exceção é disparada, com uma mensagem de erro descritiva, caso a função de multiplicação seja aplicada em matrizes com shape inconsistentes.

Para manter a integridade do programa e evitar vazamentos de memória, os Tipos Abstratos de Dados (TADs) implementam destrutores apropriados. Além disso, os métodos `SegTree::update` e `SegTree::build` garantem que as matrizes a serem substituídas são deletadas. Foram realizados testes com o Valgrind, e nenhum erro relacionado à alocação de memória foi observado.

Entretanto, é importante destacar que o programa ainda possui limitações, uma vez que não cobre um amplo espectro de possíveis entradas de dados, presumindo que o usuário fornecerá entradas corretas.

5 Análise Experimental

O presente experimento foi conduzido com o propósito de avaliar o custo computacional das operações de atualização e consulta para a Árvore de Segmentação e para o Vetor Simples de Matrizes. Dois conjuntos de testes foram gerados, sendo que, no primeiro, empregou-se um vetor de matrizes com tamanhos variando de 100 a 20 mil, e o número de operações correspondia sempre à metade do tamanho do vetor.

Os resultados evidenciaram uma notável discrepância no desempenho entre os dois métodos implementados. A Árvore de Segmentação apresentou um tempo de consulta significativamente inferior ao vetor simples, corroborando a análise teórica de complexidade. Conforme esperado, a `SegTree` demonstrou uma complexidade de consulta $O(\log n)$, enquanto o `SimpleVector` exibiu uma complexidade $O(n)$. No entanto, a visualização da curva em formato logarítmico tornou-se desafiadora devido às oscilações e à disparidade nas magnitudes de tempo entre os dois métodos.

Em relação à operação de atualização, o `SimpleVector` destacou-se, pois basta acessar uma posição no vetor e modificar a matriz, resultando em uma complexidade $O(1)$. Em contraste, na Árvore de Segmentação, é necessário modificar todo um ramo da árvore, resultando em uma complexidade $O(\log n)$.

No segundo conjunto de testes, foi empregado um vetor de matrizes de tamanho variando de 100 a 1000, com o número de operações correspondendo a 90% do tamanho. Observou-se a mesma comparação entre os métodos do conjunto anterior. No entanto, o formato de curva logarítmica na consulta da Árvore de Segmentação pôde ser melhor visualizado.

Em síntese, os resultados reforçam a eficácia da Árvore de Segmentação em operações de consulta, enquanto o `SimpleVector` se destaca em operações de atualização. A escolha entre essas estruturas de dados deve considerar os requisitos específicos da aplicação em questão. Ver Figuras no Apêndice B.

Os experimentos foram executados com as especificações abaixo:

Software:

- **System Version:** macOS 14.1.1 (23B81)
- **Kernel Version:** Darwin 23.1.0

Hardware:

- **Model Name:** MacBook Pro
- **Chip:** Apple M1 Pro
- **Total Number of Cores:** 10 (8 performance and 2 efficiency)
- **Memory:** 16 GB

---

6 Conclusão

Em resumo, a implementação da Árvore de Segmentação sobre um vetor de matrizes $A_{2\times 2}$ se revelou uma abordagem eficaz para lidar com transformações vetoriais em um espaço $2D$. A estrutura da árvore permitiu realizar consultas de maneira eficiente, sintetizando sequências de transformações em matrizes resultantes R_{ij}.

Ao construir a Árvore de Segmentação, a divisão recursiva do vetor em segmentos menores possibilitou consultas rápidas e atualizações eficientes. Essa abordagem hierárquica ofereceu um equilíbrio entre complexidade de tempo e espaço, tornando-a especialmente adequada para cenários em que o vetor de entrada é estático e as operações mais frequentes envolvem consultas ou atualizações em intervalos específicos.

A capacidade de calcular a matriz de transformação R_{ij} associada a um subvetor de matrizes permitiu prever de maneira eficaz o resultado da aplicação de transformações vetoriais a um vetor 2D fornecido. Essa funcionalidade é valiosa em diversas aplicações, como gráficos computacionais, simulações físicas e processamento de imagens.

Em conclusão, a implementação bem-sucedida desta Árvore de Segmentação abre portas para uma gama de aplicações em que a manipulação eficiente de transformações vetoriais em um espaço bidimensional é crucial.

7 Bibliografia

1. Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

2. Introduction to Algorithms, Thomas H. Cormem, Charles E. Leiserson, Ronald L. Rivest.

A Instruções para Compilação e Execução

Observação: Certifique-se de que você tenha o compilador GCC (g++) instalado em seu sistema para a compilação.

A.1 Compilação do Projeto

Para compilar o projeto, siga as instruções abaixo:

1. Abra um terminal e navegue até o diretório raiz do projeto.
2. Certifique-se de que o projeto contenha a seguinte estrutura de diretórios:

   ```
   - src/
   - obj/
   - bin/
   - include/
   ```

3. Utilize o seguinte comando para compilar o projeto:

   ```
   make ou make all
   ```

   Isso irá compilar o projeto e gerar o executável `bin/tp2.out`.

A.2 Execução do Projeto

Para executar o projeto compilado, utilize o seguinte comando:

```
./bin/tp3.out
```

Este comando executará o programa principal.

A.3 Limpeza dos Arquivos Compilados

Para limpar os arquivos compilados e executáveis, utilize o seguinte comando:

```
make clean
```

Isso removerá os arquivos objetos e executáveis.

B Figuras

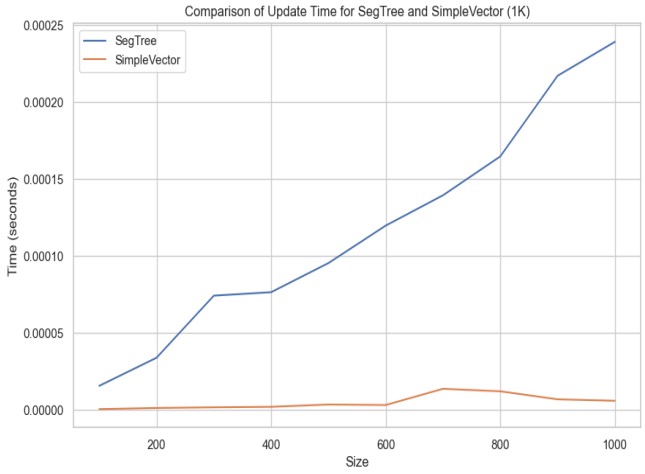

Figure 2: *Update Time 1k*

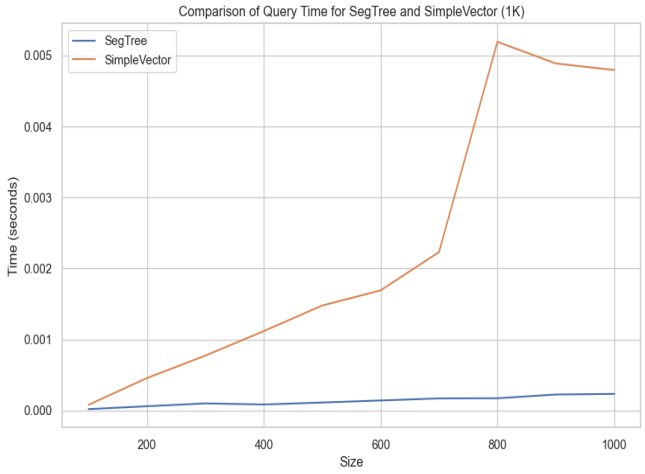

Figure 3: *Query Time 1k*

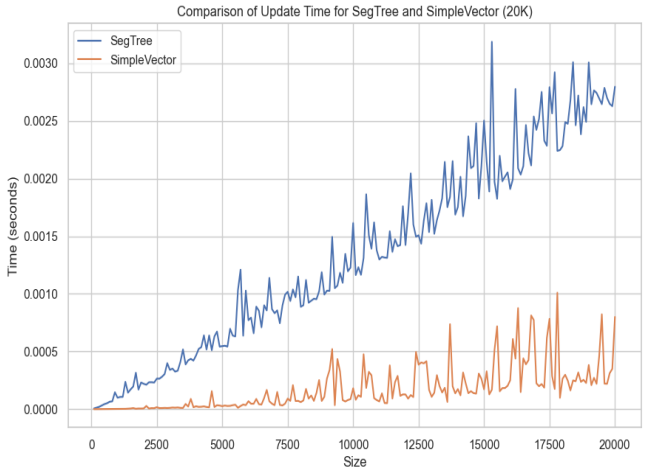

Figure 4: *Update Time 20k*

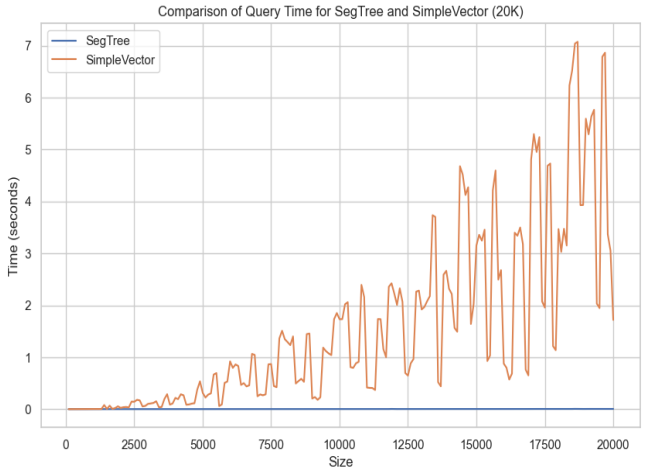

Figure 5: *Query Time 20k*