Árvores de segmentos

Eduardo Marinho e Thiago Raquel

Junho 2024

1 Resumo

Esse relatório tem como objetivo explicar a aplicação prática da Árvore de Segmentos, suas informações teóricas, assim como seus algoritmos e suas utilidades e aplicações na área de escolha geoespacial.

2 Introdução

A árvore de segmentos é uma estrutura de dados usada para armazenar invervalos ou segmentos de dados, permitindo consultas e atualizações rápidas em intervalos em relação à outras estruturas de dados para aplicações semelhantes. Ela consiste em algumas características principais:

1. Estrutura hierárquica: É uma árvore binária onde cada nó representa um intervalo.
2. Eficiência: Permite consultas e atualizações em tempo logarítmico, O(log n).
3. Flexibilidade: Suporta várias operações como soma, mínimo, máximo e outras entre intervalos.
4. Aplicações: Usada em problemas de intervalos como consultas de intervalo e problemas de processamento de sinais.

3 Informações teóricas

A árvore de segmento é uma árvore binária balanceada. Cada nó na árvore representa um intervalo da lista original. A árvore é construída de modo que a raiz represente todo o intervalo da lista, as folhas representem os elementos da lista individualmente e os nós internos representem a combinação de seus filhos, por meio da operação especificada. A árvore de segmento é uma estrutura de dados eficiente e versátil para operações de intervalo.

3.1 Implementação

Todas as funções foram feitas de maneira recursiva, por isso, foi necessário utilizar variáveis de controle como parametros da função, essas variáveis não são do interesse do usúario, portanto, foram definidas funções públicas e outras privadas, para facilitar o uso para o usuário. Note tambem que, as variáveis que começam com 'm' são variáveis do objeto e que a função "Operacao" é a função que realiza a operação definida pelo usúario para ser realizada na árvore, comummente sendo a operação de soma, de mínimo ou máximo ou de multiplicação. A árvore é armazenada como um array, na qual o filho da esquerda do nó atual tem índicie 2*p e o filho da direita 2*p+1, sendo p o índicie do pai. A raiz da árvore é armazenada do índicie 1.

3.1.1 Construção

O algoritmo de construção constroi a árvore de segmento a partir de uma lista de elementos iniciais, a operação a ser realizada na árvore e o elemento neutro da operação. O algoritmo funciona da seguinte forma.

1. Caso Base (Nó Folha): Se o intervalo atual cobre um único elemento, define o valor do nó folha como o valor correspondente da lista inicial.

2. Divisão e Recursão: Divide o intervalo atual em dois e chama recursivamente a função construir para os filhos da esquerda e da direita. Após construir os filhos, combina os resultados usando a função Operacao, definindo assim, o valor do nó atual.

3.1.2 Complexidade

A complexidade do algoritmo de construção é O(n), na qual n é o número de elementos na lista original. Isso porque, cada elemento da lista é processado uma vez e cada nó da árvore de segmento é visitado uma vez durante o processo de construção.

3.1.3 Pseudocódigo

```
public construir(lista, elemento_neutro) {
  m_numero_de_elementos = lista.size();
  m_elemento_neutro = elemento_neutro;
  construir(lista, 1, 0, m_numero_de_elementos - 1);
}
private construir(lista, indicie, inicio, final) {
  // Se inicio é igual a final, significa que o intervalo atual cobre
  // apenas um único elemento, sendo esse um nó folha na árvore de
  // segmento. O valor dos nós folhas são definidos como o valor
  // correspondente da lista inicial (lista[inicio]).
  if (inicio == final)
    return m_arvore_segmentos[indicie] = lista[inicio];
```

```
  meio = (inicio + final) / 2; // indicie do meio do intervalo.

  // Chama a função para construir os filhos do nó atual e, após isso,
  // combinamos os resultados usando a função Operacao para definir o
  // valor do nó atual (m_arvore_segmentos[indicie]).
  return m_arvore_segmentos[indicie] = Operacao(
            construir(lista, 2 * indicie, inicio, meio), // inicio ate o meio
            construir(lista, 2 * indicie + 1, meio + 1, final) // meio ate o fim
            );
}
```

3.1.4 Atualização

O algoritmo altera o valor do nó que representa o indície da lista desejado e todos os seus pais de maneira recursiva, de modo a alterar todos os nós da raiz até o nó sendo alterado da árvore, atualizando todos os intervalos que contém o nó sendo atualizado, de modo a refletir a mudança do valor, recalculando todos os intervalos.

3.1.5 Complexidade

Esse algorítmo de atualização percorre a árvore de segmento do nó raiz até o nó a ser alterado (nó folha, no pior caso), realizando um número constante de operações em cada nível da árvore. Como a altura da árvore é O(log n), a complexidade da operação de atualização é O(log n).

3.1.6 Pseudocódigo

```
public atualizar(indicie_lista, valor) {
  // Executa o algoritmo começando da raiz, com os intervalos limites da árvore
  atualizar(indicie_lista, valor, 1, 0, m_numero_de_elementos - 1);
}
private atualizar(indicie_lista, valor, indicie_arvore, inicio, fim) {
  // Se o indicie_lista está fora dos limites do intervalo atual
  // [inicio, fim], a função retorna o valor atual do nó
  // (m_arvore_segmentos[indicie_arvore]). Isso significa que esse nó
  // e seus filhos não precisam ser atualizados.
  if (indicie_lista < inicio || indicie_lista > fim)
    return m_arvore_segmentos[indicie_arvore];

  // Se inicio é igual a fim, significa que o intervalo atual cobre
  // apenas um único elemento. Esse é o nó folha correspondente ao
  // índice indicie_lista na lista original. Portanto, atualizamos
  // o valor deste nó com o novo valor fornecido e retornamos o valor
  // atualizado.
```

```
  else if (inicio == fim)
    return m_arvore_segmentos[indicie_arvore] = valor;

  meio = (inicio + fim) / 2;
  return m_arvore_segmentos[indicie] = Operacao(
             atualizar(indicie_lista, valor, 2 * indicie_arvore, inicio, meio),
             atualizar(indicie_lista, valor, 2 * indicie_arvore + 1, meio + 1, fim)
             );
}
```

3.1.7 Remoção

O valor do nó é atualizado para ser o elemento neutro da operação, o que, na prática, é como se o nó não existisse.

3.1.8 Complexidade

Mesma da Atualização, por ser exatamente o mesmo algoritmo.

3.1.9 Pseudocódigo

```
public remover(indicie_lista) {
  atualizar(indicie_lista, m_elemento_neutro, 1, 0, m_numero_de_elementos -1);
}
```

3.1.10 Consulta

O algoritmo de consulta na árvore de segmentos funciona da seguinte forma:

1. Verificação de Intervalo: Verifica se o intervalo de consulta se sobrepõe ao intervalo coberto pelo nó atual. Se não houver sobreposição, retorna o elemento neutro da operação.

2. Intervalo Totalmente Dentro: Se o intervalo coberto pelo nó atual está completamente dentro do intervalo de consulta, retorna o valor armazenado no nó atual.

3. Divisão e Recursão: Divide o intervalo atual em dois subintervalos e chama recursivamente a função consultar para os filhos da esquerda e da direita. Combina os resultados das consultas dos filhos usando a função Operacao.

3.1.11 Complexidade

A complexidade do algoritmo de consulta é O(log n), onde n é o número de elementos na lista original. Isso se deve ao fato de que a altura da árvore de segmentos ser O(log n), e ao fato de que cada nó é visitado uma vez por nível da árvore durante a consulta.

3.1.12 Pseudocódigo

```
public consultar(inicio_intervalo, final_intervalo) {
  // Executa o algoritmo começando da raiz, com os intervalos limites
  // da árvore. Chama com o final_intervalo - 1, pois o usuário espera
  // que o intervalo passado por ele seja um intervalo fechado, aberto,
  // mas o algoritmo espera que o intervalo seja fechado, fechado.
  return consultar(inicio_intervalo, final_intervalo - 1, 1, 0,
                   m_numero_de_elementos - 1);
}
private consultar(inicio_intervalo, final_intervalo, indicie, incio, final) {
  // Se o intervalo de consulta [inicio_intervalo, final_intervalo] não se
  // sobrepõe ao intervalo coberto pelo nó atual [incio, final], a função
  // retorna o elemento neutro da operação
  if (final_intervalo < incio || inicio_intervalo > final)
    return m_elemento_neutro;

  // Se o intervalo coberto pelo nó atual está completamente dentro do
  // intervalo de consulta, retorne o valor armazenado no nó atual
  // (m_arvore_segmentos[indicie]).
  else if (inicio_intervalo <= incio && final <= final_intervalo)
    return m_arvore_segmentos[indicie];

  meio = (incio + final) / 2;

  // retornar o resultado da operação para o valor do intervalo da esquerda
  // e da direita, ou seja, combina os intervalos da direita e da esquerda.
  return Operacao(
      consultar(inicio_intervalo, final_intervalo, 2 * indicie, incio, meio),
      consultar(inicio_intervalo, final_intervalo, 2 * indicie + 1, meio + 1, final)
      );
}
```

4 Aplicação

4.1 Apresentação

A aplicação prática escolhida na área geoespacial foi a avaliação de potencial agrícola de uma região a partir de coleta de dados referentes à topografia, tipo de solo, inclinação e umidade do solo e outros fatores que afetam na deposição de matérial orgânico da região inspecionada.

Para isso, deve-se usar um sistema de posicionamento para identificar com precisão a topografia analisada. O GPS (Global Position System) usado em celulares ou automóveis fornecem um posicionamento de precisão de 5 a 10 metros, isso se dá devido à interferência do sinal de satélite para o receptor no caminho pela atmosfera (satélites orbitam a aproximadamente 20.000 km acima

da superfície). Isso resulta em diferenças de precisão de condições de céu claro e de dia nublado, além de baixa exatidão para curtas distâncias.

A solução disso é utilizar o método RTK (Real-Time Kinetic), similar ao GPS também utiliza dois receptores, um deles é o estacionário e fica em um ponto com coordenadas conhecidas fixas, e o outro é o "rover" que fica junto ao usuário executando a coleta de dados. A estação base fica constantemente calculando correções em relação às coordenadas do ponto em que está estacionada, mede erros e tais correções são transmitidas para o rover em tempo real. Isso resulta em alta precisão proporcionada (aprox. 2cm) e trabalhos que demoravam dias para ser executados com outros métodos podem ser feitos em algumas horas com o sistema RTK.

4.2 Aplicação da Segment Tree no problema

Podemos utilizar a Segment Tree no problema em questão para armezanar informação de concavidade, topografia, ph, umidade e fertilidade do solo e outros fatores em níveis ou intervalos na área inspecionada, e calcular a soma desses fatores com seus respectivos índices de influência no potencial agrícola para avaliar as áreas da fazenda, facilitando o trabalho do agricultor em decidir onde realizar as plantações de alta produtividade.

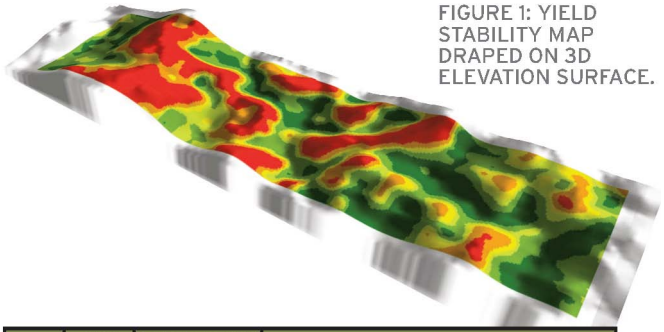
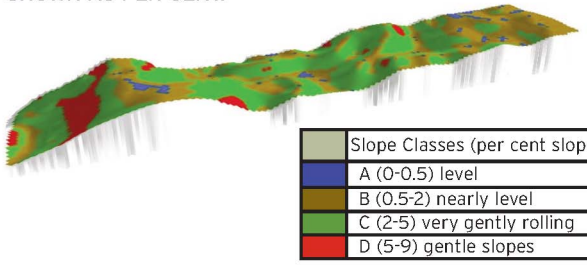


FIGURE 1: YIELD STABILITY MAP DRAPED ON 3D ELEVATION SURFACE.

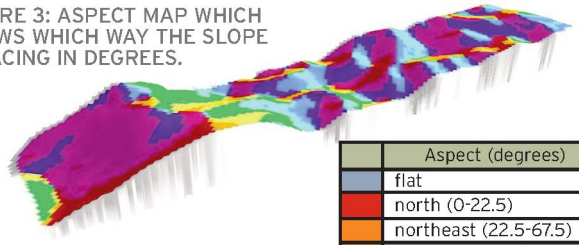
Zone	Area (acres)	Area (per cent of total)	Yield Index All Crops
1	5.6	20.2	stable, low yields
2	4.5	16.3	somewhat stable, below average yields
3	2.3	8.4	not stable, slightly below average yields
4	2.7	9.6	not stable, slightly above average yields
5	6.5	23.7	somewhat stable, above average yields
6	6	21.6	stable, high yields
Total	27.6	100.0	

FIGURE 2: EXAMPLE OF A SLOPE MAP WHICH IS RISE (HEIGHT) OVER RUN (DISTANCE DOWN THE FIELD) MULTIPLIED BY 100 AND SHOWN AS PER CENT.



	Slope Classes (per cent slope)
	A (0-0.5) level
	B (0.5-2) nearly level
	C (2-5) very gently rolling
	D (5-9) gentle slopes

FIGURE 3: ASPECT MAP WHICH SHOWS WHICH WAY THE SLOPE IS FACING IN DEGREES.



	Aspect (degrees)
	flat
	north (0-22.5)
	northeast (22.5-67.5)
	east (67.5-112.5)
	southeast (112.5-157.5)
	south (157.5-202.5)
	southwest (202.5-247.5)
	west (247.5-292.5)
	northwest (292.5-337.5)
	north (337.5-359.9434)

FIGURE 4: WETNESS INDEX CALCULATED FROM GRADIENT (% SLOPE) AND THE AMOUNT OF UPSLOPE AREA.

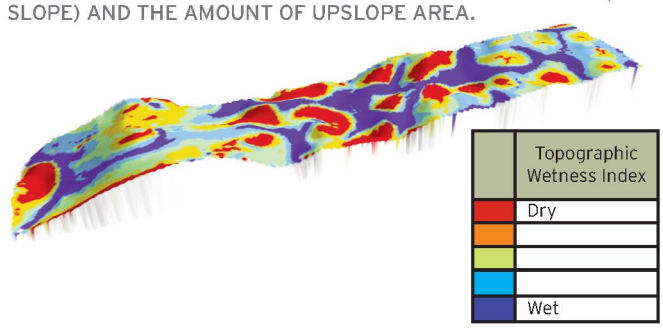


FIGURE 7: CURVATURE MAP THAT SHOWS CONVEX VS. CONCAVE AREAS ACROSS THE FIELD.

5 Conclusão

O uso de árvores de segmentos é vantajoso em aplicações reais devido à sua eficiência em consulta e atualizações de intervalos em relação a outras estruturas de dados, provendo complexidade temporal $O(\log n)$ em ambas as situações. Essa eficiência é especialmente útil em contextos como a análise geoespacial para a agricultura, onde grandes volumes de dados precisam ser processados rapidamente. A capacidade de realizar consultas e atualizações de forma ágil permite uma avaliação precisa e em tempo real das condições do solo, auxiliando os agricultores na tomada de decisões informadas sobre o manejo das culturas. Além disso, a flexibilidade da árvore de segmentos em suportar múltiplas operações em intervalos torna-a uma ferramenta poderosa e adaptável para diversas necessidades analíticas. Em suma, a adoção dessa estrutura de dados pode melhorar significativamente a precisão e a eficiência das operações agrícolas, resultando em um melhor uso dos recursos e aumento da produtividade.

6 Repositório

Esse repósitorio contém uma implementação genérica de uma arvore de segmentos, assim como esse relatório e o meterial de apresentação desse trabalho. Link: https://github.com/enfmarinho/segtree

7 Referências bibliográficas

wikipédia

cp-algorithms

geeksforgeeks

ontariograinfarmer

cpetecnologia

livro Competitive Programmer's Handbook