Árvores de Segmentos

Tiago Reis

11 de agosto de 2017

Introdução

- Estrutura que permite consultas eficientes em intervalos de um vetor.
- Exemplo: Range Sum Query (RSQ). O algoritmo trivial consome tempo linear no tamanho do intervalo para cada consulta.
- A Árvore de Segmentos faz esse tipo de consulta sobre um vetor de tamanho N em $O(\log N)$ operações.

Construção da Árvore

- A raiz da árvore armazena o intervalo $[0, N-1]$.
- Um intervalo $[L, R]$ armazenado no índice p é particionado em dois subintervalos $[L, (L+R)/2]$ e $[(L+R)/2+1, R]$ nos filhos esquerdo e direito de p, respectivamente.
- Podemos construir uma solução recursiva para as operações na árvore, sendo o caso base quando $L == R$.
- Vamos ver alguns códigos retirados do Hackerearth.

Construção da árvore

```
void build(int node, int start, int end)
{
    if(start == end)
    {
        // Leaf node will have a single element
        tree[node] = A[start];
    }
    else
    {
        int mid = (start + end) / 2;
        // Recurse on the left child
        build(2*node, start, mid);
        // Recurse on the right child
        build(2*node+1, mid+1, end);
        // Internal node value calculated
        tree[node] = tree[2*node] + tree[2*node+1];
    }
}
```

Consulta na árvore

```
int query(int node, int start, int end, int l, int r)
{
    if(r < start or end < l)
    {
        // node range is outside the queried range
        return 0;
    }
    if(l <= start and end <= r)
    {
        // node range is inside the queried range
        return tree[node];
    }
    // node range node is partially inside
    // and partially outside the given range
    int mid = (start + end) / 2;
    int p1 = query(2*node, start, mid, l, r);
    int p2 = query(2*node+1, mid+1, end, l, r);
    return (p1 + p2);
}
```

Atualização de um ponto

```
void update(int node, int start, int end, int idx, int val)
{
    if(start == end)
    {
        // Leaf node
        A[idx] += val;
        tree[node] += val;
    }
    else
    {
        int mid = (start + end) / 2;
        if(idx <= mid)
            update(2*node, start, mid, idx, val);
        else
            update(2*node+1, mid+1, end, idx, val);

        tree[node] = tree[2*node] + tree[2*node+1];
    }
}
```

Atualização em um intervalo

```
void updateRange(int node,int start,int end,int l,int r,int val)
{
    if(lazy[node] != 0)
    {
        // This node needs to be updated
        tree[node] += (end - start + 1) * lazy[node];
        if(start != end)
        {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if(start > r or end < l)
        return;
    if(start >= l and end <= r)
    {
        tree[node] += (end - start + 1) * val;
        if(start != end)
        {
            lazy[node*2] += val;
            lazy[node*2+1] += val;
        }
        return;
    }
    int mid = (start + end) / 2;
    updateRange(node*2, start, mid, l, r, val);
    updateRange(node*2 + 1, mid + 1, end, l, r, val);
    tree[node] = tree[node*2] + tree[node*2+1];
}
```

Problemas que podem ser resolvidos

- Aplicações diretas: range sum/product/minimum/maximum query, soma máxima de um subvetor, ...
- Às vezes, as aplicações não são diretas: contagem de inversões.

Exemplo: Encontrar o k-ésimo elemento

- É dada uma permutação de n elementos.
- Temos dois tipos de operações: deletar um elemento e consultar qual é o k-ésimo menor dos que restaram.
- É possível fazer a atualização em $O(\log n)$ e a consulta em $O(\log^2 n)$.

Exemplo: Codeforces 813E

- Problema Army Creation.
- São dadas q consultas online, sobre uma fila de n soldados, para um número k fixo. Cada soldado tem um tipo.
- Consulta: qual é o maior exército possível usando do l-ésimo ao r-ésimo soldado, sendo que não pode haver mais que k do mesmo tipo?
- Todos os números da entrada podem variar de 1 a 10^5.