Estruturas de dados temporais

Anahí Coimbra Maciel

Março de 2024

---

1 Introdução

Em estruturas de dados (EDs) convencionais, ao se executar uma operação de modificação, o estado anterior da estrutura é perdida, tendo o usuário acesso apenas à versão atual. As estruturas de dados temporais são uma forma de contornar esse problema, permitindo que o usuário consulte versões anteriores e execute operações em instantes do passado. Essas estruturas têm aplicações em diversos contextos, como por exemplo, em sistemas de controle de versão, sistemas de bancos de dados, linguagens funcionais, etc.

Dentre as EDs temporais, podemos distinguir dois tipos, as EDs retroativas e as persistentes. EDs persistentes, formalmente introduzidas por Driscoll, Sarnak, Sleator e Tarjan [4], mantêm diversas versões da estrutura,e operações podem ser executadas em uma versão de modo a produzir uma nova versão. Por outro lado, no paradigma da retroatividade, introduzido por Demaine, Iacono e Langerman [2], a sequência de operações executadas na estrutura não é fixo, e o usuário pode inserir e deletar operações em qualquer instante.

As EDs retroativas e persistentes têm em comum o fato de ambas introduzirem a noção de tempo, permitindo consultas a versões anteriores da ED. No entanto, enquanto nas EDs persistentes cada versão é tratada como um repositório imutável, nas EDs retroativas as operações podem ser executadas diretamente na versão anterior da ED, afetando o estado da ED em todas as versões posteriores.

2 Retroatividade

Numa estrutura de dados tradicional, há operações de modificação (update), que promovem a inserção ou remoção de elementos no instante presente, e de consulta (query), que consultam o estado da estrutura também no momento presente, ou seja, após todas as modificações feitas. Por outro lado, dentro do paradigma da retroatividade, as operações são indexadas por instantes de tempo, e é possível fazer inserções, remoções e consultas em instantes diferentes do presente. Dessa forma, uma alteração em um instante t altera o estado da estrutura de dados em todos os instantes posteriores.

Primeiramente, deve-se fazer uma distinção entre retroatividade parcial e retroatividade total. No caso da retroatividade parcial, as inserções e remoções podem ocorrer em qualquer instante, enquanto as consultas ocorrem apenas no presente. Já no caso da retroatividade total, as consultas também podem ocorrer em qualquer instante.

2.1 A lista de operações

Como mencionado anteriormente, em estruturas de dados retroativas, as operações passam a ser indexadas por um instante de tempo. Nesse contexto, faz sentido pensar em uma lista de operações $U = [u(t_1), \ldots, u(t_m)]$, onde $u(t_i)$ é uma operação de modificação realizada no instante t_i e $t_1 < t_2 < \cdots < t_m$. Dessa forma, ao adicionar ou remover um elemento da estrutura estamos adicionando ou removendo operações de U.

2.2 Fila parcialmente retroativa

A fila parcialmente retroativa consiste em uma fila - estrutura em que o primeiro elemento a ser inserido é o primeiro a ser removido - parcialmente retroativa, ou seja, inserções e remoções ocorrem em qualquer instante e consultas (de primeiro e último elementos), apenas no presente. No geral, essa estrutura possui as seguintes funções:

```
void add_enqueue(int time, int value);

void add_dequeue(int time);

void remove_enqueue(int time);

void remove_dequeue(int time);

int query_first();

int query_last();
```

Respectivamente, as funções:

1. Adiciona na lista de operações um enqueue() no instante time com valor value;
2. Adiciona na lista de operações um dequeue() no instante time;
3. Remove da lista o enqueue() no instante time;
4. Remove da lista o dequeue() no instante time;
5. Consulta o primeiro da fila no instante atual;
6. Consulta o último da fila no instante atual.

2.2.1 Estrutura interna

A fila parcialmente retroativa pode ser implementada a partir de uma lista duplamente ligada e uma árvore auxiliar. Guarda-se também um ponteiro *front*, que aponta para a primeira célula da fila. Dessa forma, responde-se à consulta *query_first()* simplesmente retornando o valor desse nó.

A lista duplamente ligada é composta por células que armazenam os valores *time* (instante da operação), *value* (valor associado à operação), *next* (ponteiro para a próxima célula) e *past* (ponteiro para a célula anterior). Para tornar a inserção e remoção mais eficientes, uma árvore de busca binária balanceada ordenada por tempo é utilizada. Cada nó tem como valor uma referência à célula da lista ligada correspondente ao seu tempo. Desse modo, nas rotinas de

inserção e remoção da lista, as buscas são feitas na árvore, já que buscas em árvores balanceadas são mais eficientes que buscas em listas ligadas. A lista duplamente ligada também possui as funções:

```
cell* forward(cell* ptr);
cell* backward(cell* ptr);
```

As quais, respectivamente, deslocam um ponteiro para a célula seguinte e anterior da lista. Por fim, estão presentes também os ponteiros *head* e *last*, que apontam para a primeira e última célula da lista. O ponteiro *last* é utilizado na consulta *query_last*.

2.2.2 Implementação

Na fila parcialmente retroativa, apenas as operações de *enqueue()* são guardadas, já que a adição ou remoção de um *dequeue()* tem como único efeito a mudança do elemento no topo da lista, e esse controle pode ser feito com o ponteiro *front*.

Dessa forma, ao se adicionar ou remover um *dequeue()*, move-se o ponteiro *front* respectivamente para a célula sucessora ou antecessora.

Já no caso de uma inserção de *enqueue()*, insere-se o elemento na lista ligada e na árvore no instante correspondente. Além disso, o ponteiro *front* é atualizado da seguinte maneira: se anteriormente a fila estava vazia, ele passa a apontar para o elemento adicionado. Se o instante do elemento adicionado é menor do que o instante do elemento apontado pelo ponteiro, move-se o *front* para a célula anterior.

Por fim, no caso de uma remoção de *enqueue()*, o elemento do instante correspondente é removido da lista e da árvore, e o ponteiro *front* é movido para a célula sucessora se o instante do elemento removido for menor ou igual ao instante do ponteiro.

2.3 Fila totalmente retroativa

A fila parcialmente retroativa consiste em uma fila - estrutura em que o primeiro elemento a ser inserido é o primeiro a ser removido - parcialmente retroativa, isto é, inserções, remoções e consultas ocorrem em qualquer instante. No geral, essa estrutura possui as seguintes funções:

```
void add_enqueue(int time, int value);

void add_dequeue(int time);

void remove_enqueue(int time);

void remove_dequeue(int time);

int query_first(int time);

int query_kth(int time, int k);
```

A única função que difere em relação à fila parcialmente retroativa é a consulta *query_kth(int time, int k)*, que retorna o k-ésimo elemento da fila. Percebe-se, também, que as consultas agora recebem o parâmetro *time*, já que se trata da versão totalmente retroativa da estrutura.

2.3.1 Estrutura interna

A estrutura utilizada para implementar a fila totalmente retroativa é uma árvore binária balanceada ordenada por tempo com dois tipos de nó, o nó interno e a folha. As folhas armazenam as informações das operações inseridas ou deletadas (instante de inserção/deleção e valor, no caso das inserções). Os nós internos carregam atributos que permitem que as buscas sejam feitas de forma mais eficiente, a saber, o número de folhas em sua subárvore e o menor instante em uma folha de sua subárvore. A árvore possui duas funções que a diferem de uma árvore binária de busca comum:

```
int count(int time);
int Kth(int time, int k);
```

As quais, respectivamente, retornam o número de folhas com instante $\leq time$ e o k-ésimo elemento da árvore, caso seu instante seja $\leq time$. Essas funções são necessárias para responder às consultas, como ficará claro na seção a seguir.

2.3.2 Implementação

A fila totalmente retroativa é implementada por meio de duas árvores, uma para armazenar as operações de inserção e outra para armazenar as operações de deleção. Dessa forma, as operações são adicionadas ou removidas simplesmente adicionando-as ou removendo-as da árvore correspondente.

Para responder às consultas, leva-se em conta o fato que o primeiro elemento após x remoções será aquele que foi inserido na $(x+1)$-ésima inserção. Portanto, no caso de uma consulta no instante t, conta-se quantas remoções de elementos ocorreram até esse instante. Sendo d esse número, o primeiro elemento da lista em t é aquele que ocupa a $(d+1)$-ésima posição na árvore de inserções, e o n-ésimo elemento da lista em t é aquele que ocupa a $(d+n)$-ésima posição na árvore de inserções.

2.4 Pilha totalmente retroativa

A pilha totalmente retroativa consiste em uma pilha - estrutura em que o primeiro elemento a ser inserido é o último a ser removido - totalmente retroativa. No geral, essa estrutura possui as seguintes funções:

```
void add_push(int time, int value);

void add_pop(int time);

void remove_push(int time);

void remove_pop(int time);

int query_top(int time);

int query_kth(int time,int k);
```

Respectivamente, as funções:

1. Adiciona na lista de operações um push() no instante time com valor value;

2. Adiciona na lista de operações um pop() no instante time;
3. Remove da lista o push() no instante time;
4. Remove da lista o pop() no instante time;
5. Consulta o primeiro da pilha no instante *time*;
6. Consulta o k-ésimo da pilha no instante *time*.

2.4.1 Estrutura interna

Assim como no caso da fila totalmente retroativa, a pilha totalmente retroativa também utiliza uma árvore binária balanceada ordenada por tempo com nós internos e folhas. No entanto, é utilizada apenas uma árvore, então associa-se um atributo *weight* (peso) a cada folha, sendo que esse valor é $+1$ no caso de uma inserção e -1 no caso de uma remoção. O peso de nós internos corresponde à soma dos pesos das folhas de sua subárvore. Além disso, os nós também têm o atributo *leftover*, que nas folhas de peso negativo é 0, nas folhas de peso positivo é 1 e nos nós internos é o sufixo (soma dos pesos das folhas, retrocedendo nas folhas da árvore) de sua subárvore de maior soma. Novamente, a árvore possui as funções:

```
int count(int time);
int Kth(int time, int k);
```

As quais, respectivamente, retornam o número de elementos no instante *time* e a k-ésima folha de peso positiva não-anulada, retrocedendo na árvore a partir do instante *time*. Por não-anulada, queremos dizer que não há, até o instante *time*, nenhuma folha de peso negativo que torne o prefixo começando na folha em questão igual a 0.

2.4.2 Implementação

Como dito anteriormente, a implementação da pilha totalmente retroativa utiliza uma árvore binária. Para inserir ou remover uma operação, insere-se ou remove-se essa operação na árvore. Para fazer uma consulta do k-ésimo elemento no instante *t*, busca-se a k-ésima folha positiva não-anulada com instante menor que *t*. Ou seja, queremos a inserção que, retrocedendo nas folhas da árvore a partir desse instante, torna a soma dos pesos das operações $+k$.

2.5 Fila de prioridade parcialmente retroativa

Uma fila de prioridade é uma estrutura de dados que contém um conjunto de elementos ordenados por prioridade. Aqui apresentamos uma fila de prioridade parcialmente retroativa ordenada por valor mínimo, isto é, um elemento com valor menor tem prioridade maior. Essa estrutura possui as funções:

```
void add_insert(int time, int value);

void add_delete_min(int time);

void remove_insert(int time);
```

```
void remove_delete_min(int time);

int query_min();
```

As quais, respectivamente:

1. Adiciona na lista de operações um insert() no instante time com valor value;
2. Adiciona na lista de operações um delete_min() no instante time;
3. Remove da lista o insert() no instante time;
4. Remove da lista o delete_min() no instante time;
5. Consulta o mínimo do fila no momento atual.

2.5.1 Estrutura interna

A fila de prioridade parcialmente retroativa usa para guardar as operações uma árvore binária balanceada ordenada similar à da pilha retroativa, mas com dois novos atributos: max_out, que corresponde ao maior valor de uma folha da subárvore desse nó que não está na fila no instante atual, e min_in, que corresponde ao menor valor de uma folha da subárvore desse nó que está na fila no instante atual. O uso do atributo *weight* também muda: as folhas que correspondem às operações *insert()* cujos elementos estão na fila no instante atual recebem peso 0, enquanto aquelas cujos elementos não estão na fila recebem peso 1. As folhas que correspondem à operação *delete_min()* recebem peso -1.

Essas mudanças são necessárias devido ao fato de que toda inserção na fila de prioridade pode desencadear uma cascata de alterações, devido à operação *delete_min*, alterando assim quais elementos estão e quais elementos não estão na fila após cada operação.

A fila de prioridade parcialmente retroativa usa ainda uma segunda árvore binária para guardar os elementos presentes da fila no instante atual, de forma a facilitar as consultas de mínimo, que só ocorrem no instante atual. Essa árvore é uma árvore binária balanceada comum, ordenada por valor.

2.5.2 Implementação

Na implementação da fila de prioridade parcialmente retroativa são utilizadas duas árvores de busca, a q_now, que guarda os elementos que estão na fila no momento atual, e a *updates*, que contém as operações inseridas. Para responder à operação de consulta, basta retornar o elemento mínimo da árvore q_now. Para adicionar ou remover operações, são utilizadas rotinas inspiradas em resultados demonstrados por Demaine em [2].

2.6 Fila de prioridade totalmente retroativa

Aqui apresentamos uma fila de prioridade totalmente retroativa obtida a partir da técnica de *hierarchical checkpointing*, como descrito em [3]. Essa estrutura apresenta as mesmas funções da fila de prioridade parcialmente retroativa.

Em primeiro lugar, chamamos uma estrutura de dados parcialmente retroativa de tempo-fusível se, dadas duas estruturas representando duas linhas de tempo diferentes e contidas em intervalos de tempo disjuntos, é possível gerar uma nova estrutura de dados representando a concatenação dessas linhas de tempo. Esse conceito será importante para a implementação da estrutura apresentada.

2.6.1 Estrutura interna

A estrutura interna utilizada é a *checkpoint tree*, uma árvore binária de busca balanceada em que cada nó interno contém uma estrutura de dados parcialmente retroativa tempo-fusível que inclui todos as operações de modificação da subárvore com raiz naquele nó. As folhas contêm apenas uma operação de modificação e são ordenadas por tempo.

2.6.2 Implementação

Cada nó interno da árvore interna é associado como uma instância de uma fila de prioridade parcialmente retroativa tempo-fusível. Essa estrutura corresponde à fila de prioridade parcialmente retroativa introduzida na seção anterior com duas estruturas auxiliares, Q_{now} e Q_{del}, árvores binárias de busca que respectivamente

1. contém todas as chaves presentes na fila no instante presente;
2. contém todas as chaves que foram removidas da fila em algum instante do passado.

Por convenção, se a fila está vazia no instante atual, uma operação de *deleteMin* tem como resultado a inserção de uma chave de peso infinito em Q_{del}.

Para manter essas estruturas auxiliares, são utilizadas usando B-árvores balanceadas com fator de balanceamento $d = 8$, como descrito em [1].

Adicionalmente, a fila de prioridade parcialmente retroativa tempo-fusível suporta uma função, *Fuse* (d_1, d_2), que funde as instâncias d_1 e d_2 da estrutura, retornando uma instância que reflete as operações de modificação aplicadas a d_1 e d_2, contando que as estruturas fundidas representem intervalos de tempo disjuntos e adjacentes. Antes de descrever o algoritmo, consideremos o lema

Lema. *Considere duas filas de prioridade parcialmente retroativas Q_1 e Q_2 contendo operações nos intervalos $[a, b)$ e $[b, c)$ respectivamente. Segue que a fila de prioridade parcialmente retroativa Q_3 contendo todos as operações de Q_1 e Q_2 no intervalo $[a, c)$ tem as propriedades*

$$Q_{3,now} = Q_{2,now} \cup max - A\{Q_{1,now} \cup Q_{2,del}\}$$
$$Q_{3,del} = Q_{1,del} \cup min - D\{Q_{1,now} \cup Q_{2,del}\}$$

onde $A = |Q_{1,now}|$ e $D = |Q_{2,del}|$.

Representamos $Q_{3,now}$ e $Q_{3,del}$ como listas de árvores obtidas por operações de divisão de árvore (tree-split operations). Além disso, dizemos que uma fila de prioridade parcialmente retroativa tempo-fusível Q^k tem ordem k se ambas Q^k_{now} e Q^k_{del} são representadas como listas de no máximo k filas.

Abaixo enumeramos os passos que compõem o algoritmo *Fuse* (Q^k_1, Q^k_2).

1. Computar A conforme o lema anterior;
2. Concatenar as listas representando $Q^k_{1,now}$ e $Q^k_{2,del}$, formando uma lista L com **no máximo** $2k$ árvores;
3. Computar x, valor maior do que D elementos contidos nas árvores de L;
4. Dividir cada árvore T_i de L em uma árvore $T_{i,<}$ contendo as chaves menores que x e uma árvore $T_{i,>}$ contendo as chaves maiores;
5. Computar $Q_{3,now}$ a partir da união de $Q_{2,now}$ e de todas as árvores $T_{i,>}$;

6. Computar $Q_{3,del}$ a partir da união de $Q_{1,del}$ e de todas as árvores $T_{i,<}$;

7. Retornar Q_3.

No terceiro passo, é utilizada a função *GetSplitKey* $(s, T_1, \ldots, T_k)$, que retorna um valor x maior que s elementos nas árvores $T_1, \ldots, T_k$.

Para executar uma operação de modificação, a operação é inserida como folha na árvore interna e aplicada a todas as instâncias da estrutura contidas em nós no caminho da raiz até a folha inserida.

Para executar a operação de consulta no instante *t*, percorre-se a árvore interna para identificar as subárvores contidas no intervalo de tempo $(-\infty, t]$. As estruturas associadas a essas subárvores são então fundidas, resultando em uma instância representando o estado da fila de prioridade totalmente retroativa no instante *t*. Finalmente, pode-se aplicar a operação de consulta na estrutura obtida.

3 Conclusão

Neste trabalho, foram apresentadas diversas estruturas de dados retroativas, a saber,

1. Fila parcialmente retroativa;
2. Fila totalmente retroativa;
3. Pilha totalmente retroativa;
4. Fila de prioridade parcialmente retroativa;
5. Fila de prioridade totalmente retroativa.

A principal contribuição do trabalho é complementar [5] ao aprofundar a exposição da fila de prioridade parcialmente retroativa e expor a fila de prioridade totalmente retroativa.

Referências

[1] Michael A Bender, Erik D Demaine, and Martin Farach-Colton, *Cache-oblivious b-trees*, Proceedings 41st Annual Symposium on Foundations of Computer Science, IEEE, 2000, pp. 399–409.

[2] Erik D. Demaine, John Iacono, and Stefan Langerman, *Retroactive data structures*, ACM Transactions on Algorithms (TALG) **3** (2007), no. 2, 13–es.

[3] Erik D. Demaine, Tim Kaler, Quanquan Liu, Aaron Sidford, and Adam Yedidia, *Polylogarithmic fully retroactive priority queues via hierarchical checkpointing*, Algorithms and Data Structures: 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings 14, Springer, 2015, pp. 263–275.

[4] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan, *Making data structures persistent*, Proceedings of the eighteenth annual ACM symposium on Theory of computing, 1986, pp. 109–121.

[5] Beatriz Figueiredo Marouelli, *Um estudo sobre estruturas de dados retroativas*, Universidade de São Paulo (2019).