UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO SUPERIOR DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Dionatan Eduardo Correa Rodrigues
Mateus Fernandes De Quadros

FILA DE PRIORIDADE

Santa Maria, RS
2023

LISTA DE FIGURAS

SUMÁRIO

1 APRESENTAÇÃO DE CONCEITOS

1.1 FILAS DE PRIORIDADE

A fila de prioridade é uma estrutura de dados essencial em ciência da computação que permite o gerenciamento eficiente de elementos com diferentes níveis de prioridade. Em muitos problemas e algoritmos, é crucial poder acessar e processar elementos de acordo com sua importância relativa.

Uma fila de prioridade é semelhante a uma fila convencional, mas a principal diferença é que cada elemento na fila de prioridade possui uma prioridade associada. Isso significa que elementos com maior prioridade têm precedência sobre aqueles com menor prioridade. Dessa forma, quando é necessário recuperar um elemento da fila, ele é escolhido com base em sua prioridade, não apenas na ordem em que foi inserido.

Uma das implementações mais comuns da fila de prioridade é utilizando a chamada "heap". O heap é uma estrutura de dados completa e balanceada, ele é uma árvore binária com uma propriedade especial chamada "propriedade do heap". Em um heap máximo, a chave de cada nó é maior ou igual às chaves de seus filhos. Em um heap mínimo, a chave de cada nó é menor ou igual às chaves de seus filhos. Com essa propriedade, é possível organizar eficientemente os elementos em uma fila de prioridade.

As operações comuns em filas de prioridade são:

- Inserção: Adiciona um elemento à fila de prioridade.
- Remoção: Remove e retorna o elemento com a maior (ou menor) prioridade da fila de prioridade.
- Impressão/Consulta: Exibe ou retorna o elemento com a maior (ou menor) prioridade sem removê-lo da fila.

1.2 HEAP BINÁRIO

Um heap binário é uma estrutura de dados em forma de árvore binária completa, em que cada nó tem um valor associado chamado de chave. A estrutura de um heap binário é definida da seguinte maneira:

- A árvore é completa, o que significa que todos os níveis estão preenchidos, exceto possivelmente o último, que é preenchido da esquerda para a direita.
- Para qualquer nó i, o valor da chave no nó pai de i é sempre maior ou igual ao valor das chaves dos seus nós filhos (propriedade de heap máximo) ou sempre menor ou igual (propriedade de heap mínimo). Essa propriedade é chamada de propriedade de heap.

A representação do heap binário pode ser feita em um array unidimensional, onde os elementos são armazenados de acordo com a ordem de nível da árvore. Por exemplo, o

nó raiz é armazenado no índice 1, os filhos do nó raiz são armazenados nos índices 2 e 3, os filhos dos nós 2 e 3 são armazenados nos índices 4, 5, 6 e 7, e assim por diante.

O heap binário é a forma mais comum de heap, em que cada nó pode ter no máximo dois filhos. Ele pode ser implementado usando um array, onde os elementos são organizados de forma que o elemento presente na posição "i"do array se torna pai das posições "2*i+1"(filho da esquerda) e "2*i+2"(filho da direita). Sendo assim, exploraremos os conceitos básicos relacionados à fila de prioridade utilizando heap. Veremos como realizar operações de inserção, remoção e acesso ao elemento de maior prioridade.

2 EXEMPLOS

A fim de ilustrar os conceitos de fila de prioridade utilizando heap, vamos explorar alguns exemplos práticos. Esses exemplos nos ajudarão a compreender como a fila de prioridade baseada em heap pode ser aplicada.

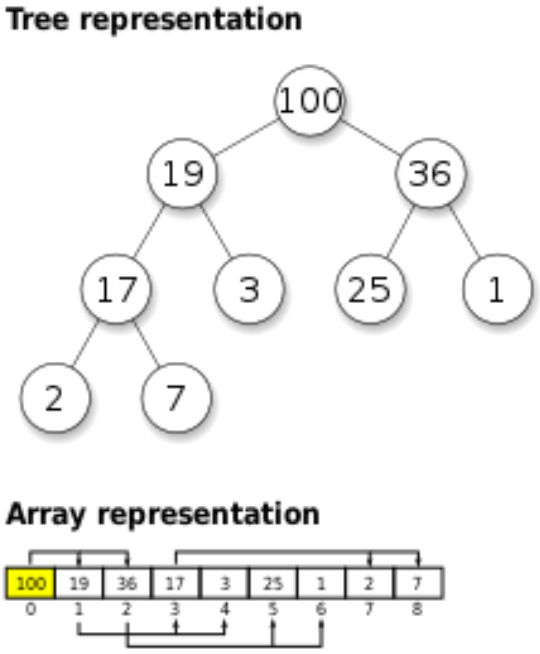

Figura 2.1 – Representação gráfica 01.

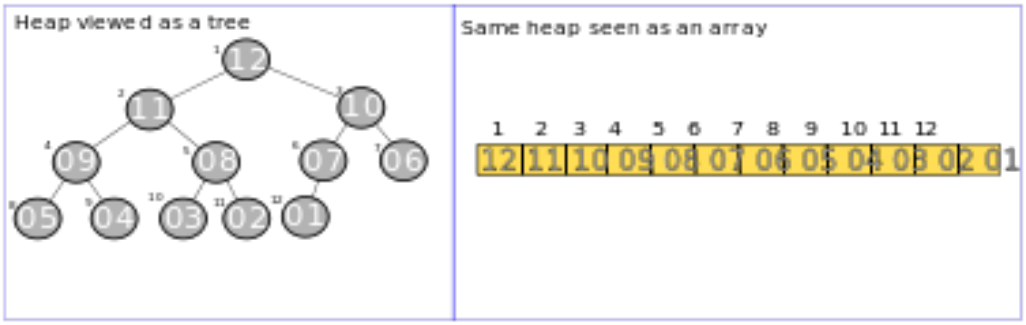

Figura 2.2 – Representação gráfica 02.

2.1 INSERÇÃO NO HEAP

O novo elemento é inserido na próxima posição disponível no final da árvore e, em seguida, é comparado com seu nó pai. Se o novo elemento ter prioridade maior que a do nó pai, ocorre uma troca entre eles. Essa comparação e troca é repetida recursivamente até que o elemento esteja em uma posição adequada no heap ou até que seja alcançada a raiz da árvore.

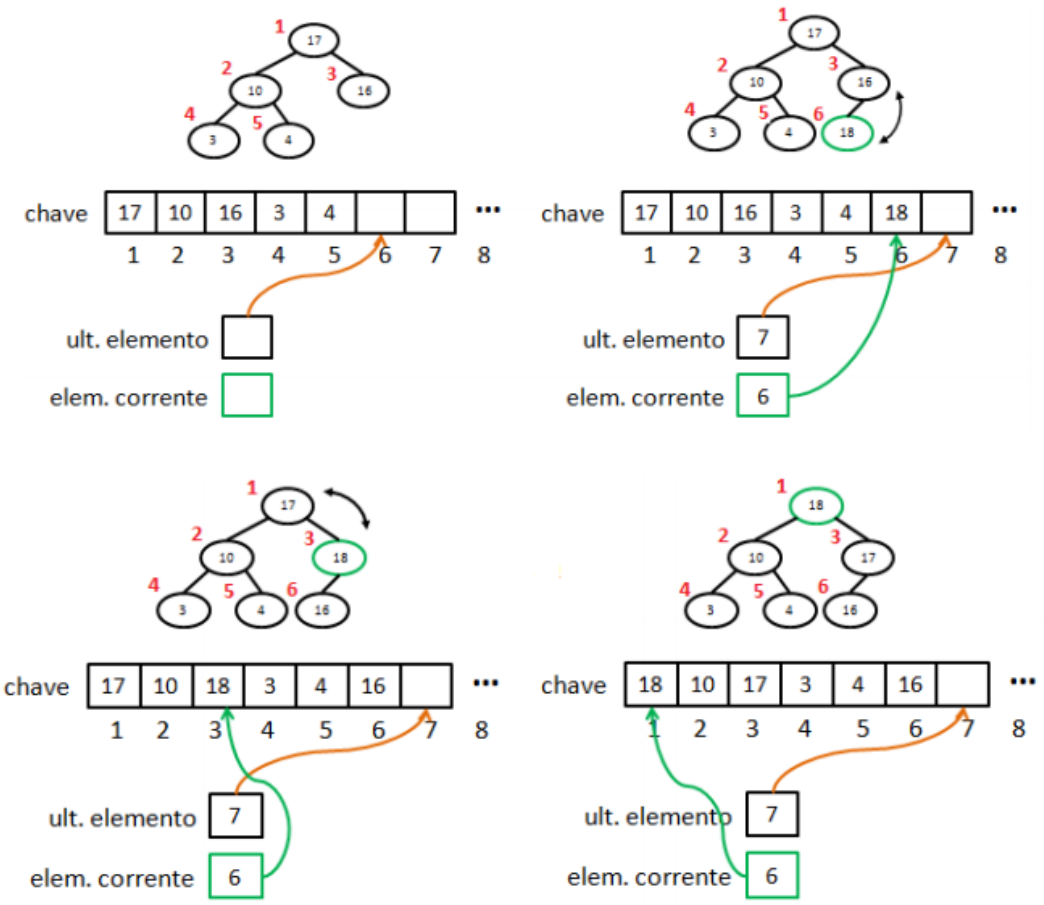

Figura 2.3 – Inserção no max-heap

2.2 REMOÇÃO NO HEAP

Removemos a raiz e substituímos pelo último elemento da árvore, preservando a estrutura completa do heap. Em seguida, comparamos o novo elemento raiz com seus nós filhos e trocamos de posição com o filho que tiver o valor maior. Esse processo é repetido até que o novo elemento raiz esteja em uma posição adequada no heap, respeitando a propriedade de max-heap.

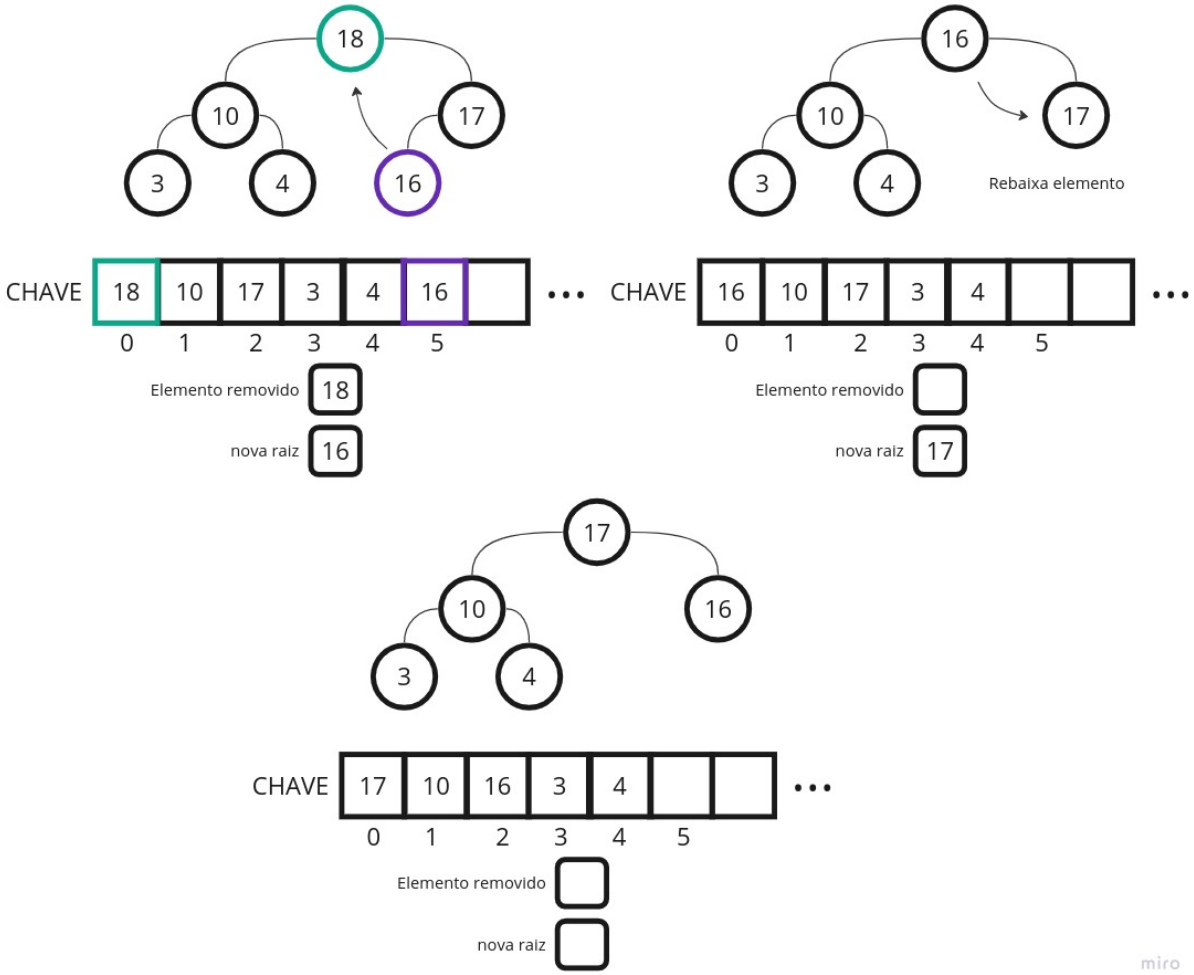

Figura 2.4 – Remoção no max-heap

3 ALGORÍTMOS USADOS

3.1 INSERÇÃO

Para fazer a inserção foram usadas duas funções, a primeira função insert() coloca o novo elemento na última posição do vetor e logo em seguida chama a função promove_elemento(), na qual tem a função de verificar as prioridades dos elementos do vetor e reorganiza-los se necessário.

```
function insert(filaPrioridade *ponteiro, int prioridade){
    se (ponteiro == NULL)
        escreva ("ERRO")

    se (ponteiro->tamanho == ponteiro->capacidade)
        escreva("Fila cheia!")

    ponteiro->vetor[ponteiro->tamnho].prioridade = prioridade;
    promove_elemento(ponteiro,ponteiro->tamanho)
    ponteiro->tamanho ++;
}

function promove_elemento(filaPrioridade *ponteiro, int filho){
    //primeiro é calculado a posição do pai do novo elemento inserido
    int pai = (filho-1)/2

    while ( (filho>0) && //testa até chegar na raiz
         ponteiro->vetor[pai].prioridade
            <                          // prioPai < prioFilho
        ponteiro->vetor[filho].prioridade){

        swap(ponteiro,filho,pai)  //faz a troca necessária
        filho = pai         //atualiza posição do filho
        pai = (pai-1)/2     //atualiza posição do pai


    } //fim laço
} //fim função
```

3.2 REMOÇÃO

Assim como na inserção para fazer a remoção foram usadas duas funções.A primeira função remocao() remove o primeiro elemento da fila e coloca o último elemento na primeira posição para que em seguida chame a função rebaixa_elemento(), a qual verifica as prioridades dos elementos do vetor e reorganiza-os quando necessário.

```
function remocao(filaPrioridade *ponteiro){
    se (ponteiro == NULL)
        escreva ("ERRO")

    se (ponteiro->tamanho == 0)
        escreva("Fila vazia!")

    ponteiro->tamanho--
    ponteiro->vetor[0] = ponteiro->vetor[ponteiro->tamanho]
    rebaixa_elemento(ponteiro,0);

}

function rebaixa_elemento(filaPrioridade *ponteiro, int pai){
    //primeiro é calculado a posição do filho do nó raiz
    int filho = (2*pai)+1

    //percorre a árvore no maximo até as folhas
    while(filho < ponteiro->tamanho){

        //testa se tem um irmão
        if(filho < ponteiro->tamanho-1){

            //testa qual irmão tem maior prioridade
            if(ponteiro->vetor[filho].prioridade <
                            ponteiro->vetor[filho+1].prioridade){
                filho++;
            }
        } //fim teste se tem irmão

        //pai ja tem prioridade >= , não precisa swap
        if(ponteiro->vetor[pai].prioridade  >=
                        ponteiro->vetor[filho].prioridade)
            break;

        swap(ponteiro,pai,filho) //faz troca necessária
        pai = filho              // atualiza posição do pai
        filho = 2*pai+1          //atualiza posição do filho

    } //fim laço
} //fim função
```

3.3 IMPRESSÃO

A função impressao() recebe um ponteiro para uma estrutura filaPrioritaria, ela verifica se a fila está vazia e imprime "Fila vazia!"caso seja verdadeiro, em caso de a fila não vazia

ela imprime cada nó com seus respectivos filhos da direita e esquerda.Imprime vazio em caso de não haver algum dos filhos.

```
function impressao (filaPrioritaria *ponteiro) {
    se (ponteiro->tamanho == 0 )
        escreve ("fila vazia!")

    para (i=0; i < ponteiro->tamanho ;i++){
        escreva(ponteiro->vetor[i].prioridade) // pai na posição i

        se (2*i+1 >= tamnho )  // testa se filho da esquerda existe
            escreva ("vazio")
        else
            escreva ("ponteiro->vetor[2*i+1].prioridade")

        se (2*i+2 >= tamnho )  // testa se filho da direita existe
            escreva ("vazio")
        else
            escreva ("ponteiro->vetor[2*i+2].prioridade")
    } //fim do laço

} //fim da função
```

REFERÊNCIAS BIBLIOGRÁFICAS

CHEN FEI HUA, Y. J. Y.; ZHANG, E. Z. Bgpq: A heap-based priority queue design for gpus. **ACM Digital Library**, v. 1, 2021.

CORMEN, T. H. **Introduction to Algorithms**. 3. ed. Nova Iorque, Estados Unidos: The MIT Press, 2009.

Cormen, T.H.; et al. **Heap**. 2001. Acesso em 27 jun. 2023. Disponível em: <https://pt.wikipedia.org/wiki/Heap>.

WEISS, M. **Data Structures Algorithm Analysis in C++**. 4. ed. Nova Iorque, Estados Unidos: Pearson Education, 2013.

(CORMEN, 2009) (WEISS, 2013) (CHEN FEI HUA; ZHANG, 2021) (Cormen, T.H.; et al., 2001)