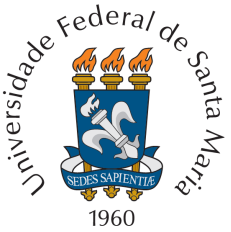


Ministério da Educação
Universidade Federal de Santa Maria
Gabinete do Reitor

FILAS DE PRIORIDADE COM HEAPS BINÁRIOS

GIULIA RODRIGUES DE ARAÚJO
VITÓRIA LUIZA CAMARA

Santa Maria - RS

2023

SUMÁRIO

1. Filas de Prioridade

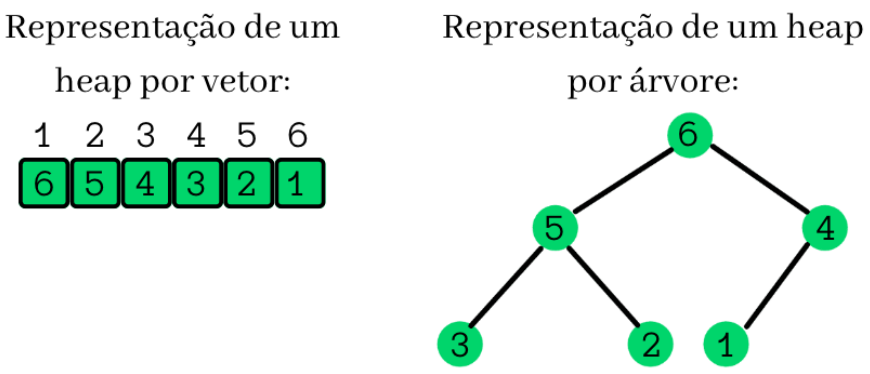
Uma fila de prioridade é um tipo de fila em que os elementos são armazenados e reorganizados de acordo com uma determinada prioridade. A prioridade determina a ordem em que os elementos são processados e, usualmente, os elementos com maior valor prioridade são processados antes dos elementos com menor prioridade.

2. Heap Binário

Um heap binário é uma árvore binária completa que é utilizada para armazenar dados eficientemente, visando coletar o elemento máximo ou mínimo de uma estrutura. Cada nó possui um valor associado, chamado de chave ou prioridade. Além disso, o heap binário deve obedecer à propriedade do heap, que pode ser de dois tipos:

- **Heap mínimo:** Em um heap mínimo, a chave do nó raiz deve ser a menor, ou igual, quando comparado aos outros nós presente no heap binário, e essa regra se aplica recursivamente para todos os nós.
- **Heap máximo:** Em um heap máximo, a chave do nó raiz deve ser a maior ou igual quando comparado ao restante dos nós, funcionando de maneira semelhante ao heap mínimo

Além disso, a forma principal que essa estrutura é implementada, na maioria dos algoritmos, é com a utilização de vetores, definindo todo valor com três propriedades: um pai, um filho da esquerda e um filho da direita. O valor do pai corresponde à (x-1)/2, do filho da esquerda à 2x + 1 e o filho da direita a 2x + 2.



Exemplo:

a = [12, 8, 11, 7, 5, 10, 6]

Posições do vetor: 0, 1, 2, 3, 4, 5, 6, 7

O pai 8 tem como filhos 7 (esquerda) e 5 (direita), e tem como índice do vetor 1, portanto:

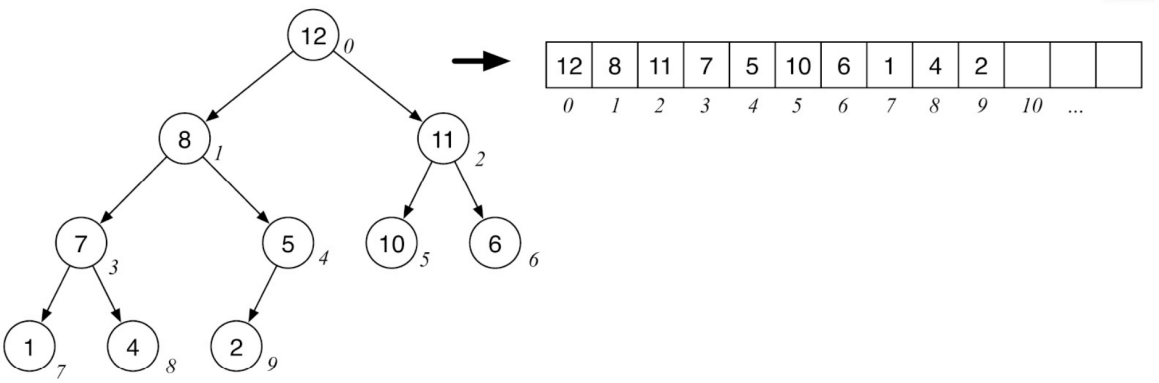
x = 1

Pai: (1-1)/2 = 0

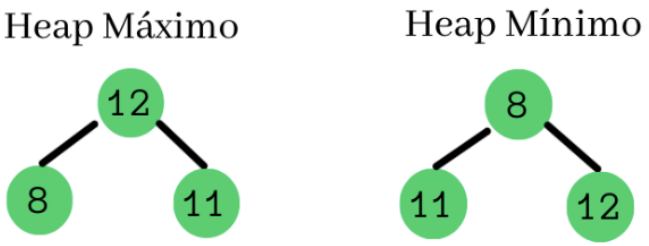
Filho da esquerda: [(2.1) + 1] = 3

Filho da direita: [(2.1) + 2] = 4

Dessa forma, no vetor a = [12, 8, 11, 7, 5, 10, 6], o pai do elemento 8 está na posição 0 (valor 12), o filho da esquerda está na posição 3 (valor 7) e o filho da direita está na posição 4 (valor 5).



Como comentado anteriormente, existem dois principais meios de hierarquia dentro de um heap binário, podendo ser eles de mínimo e máximo. Ou seja, é a forma de organização do nós da árvore de forma que o pai seja maior ou menor que seus filhos. Nesse documento, iremos explicar a utilização do heap máximo.



3. Algoritmos para Filas de Prioridade

```
static void troca (Heap* h, int i, int j){

   float tmp = h->v[i]; h->v[i] =
   h->v[j];
   h->v[j] = tmp;

}
```

```
static void sobe (Heap* h, int i){

   while (i > 0) {
      int p = pai(i);
      if (h->v[p] > h->v[i])
         break ;
      troca(h,p,i);
      i = p;
   }

}
```

```
void heap_insere (Heap* h, float value){

   h->v[h->n++] = value;
   sobe(h,h->n-1);

}
```

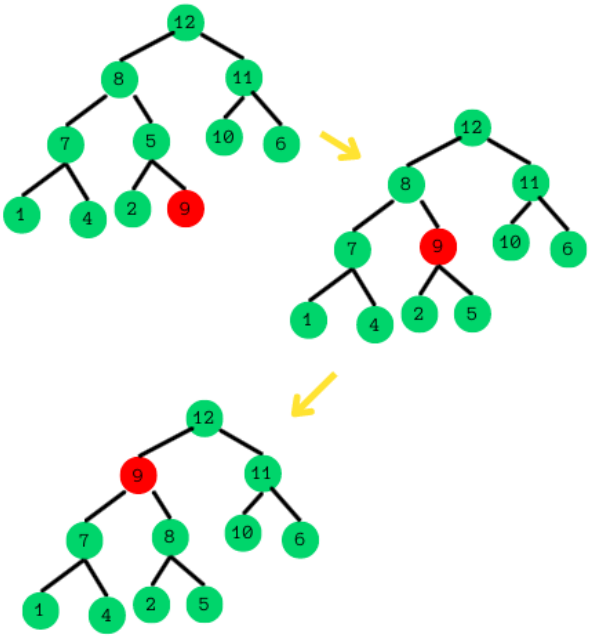
Existem vários algoritmos importantes para a implementação de filas de prioridade usando heaps binários. Os principais são:

• **Inserção:** Para inserir um elemento em uma fila de prioridade, você precisa adicionar o elemento como uma nova folha na árvore e, em seguida, ajustar a estrutura do heap para garantir que a propriedade do heap seja preservada. Como a árvore binária da lista de prioridade é sempre cheia, é possível armazená-la em um vetor. Caso seja preciso incluir um novo nó na árvore, é adicionado um elemento ao final do vetor, que equivale a mais um nó no último nível. Por exemplo:

Ao lado, temos um exemplo de funções responsáveis pela inserção de nós no heap. A função troca é responsável por trocar a posição de dois elementos dentro do vetor do Heap. Ela recebe como parâmetros um ponteiro para a estrutura Heap (h), e os índices dos elementos a serem trocados (i e j). A função realiza a troca de valores entre h->v[i] e h->v[j] usando uma variável temporária tmp. Já a função sobe é responsável por realizar o ajuste necessário após a inserção de um novo elemento no Heap. Ela recebe como parâmetros um ponteiro para a estrutura Heap (h) e o índice (i) do elemento que foi inserido. A função realiza um loop enquanto o índice não chega à raiz do Heap. Em cada iteração, o elemento é comparado com seu pai (calculado através da função pai(i)) e, se o valor do pai for menor que o valor do elemento, ocorre a troca entre eles usando a função troca. Após a troca, o índice é atualizado para o pai e o processo se repete até que o elemento esteja na posição correta no Heap. Finalmente, a função heap_insere é

responsável por adicionar um novo valor ao Heap. Ela recebe como parâmetros um ponteiro para a estrutura Heap (h) e o valor (value) a ser inserido. Primeiro, o valor é adicionado ao vetor do Heap na posição h->n (que corresponde ao final do vetor, antes da inserção). Em seguida, a função sobe é chamada passando o Heap e o índice do elemento inserido (h->n-1). Dessa forma, o novo elemento é inserido corretamente no Heap e o Heap é ajustado para manter a propriedade de max-heap.

Veja abaixo um exemplo de inserção do heap binário com o elemento 9. Note que o valor sobe na estrutura.




```
static void desce (Heap* h, int i){

  int c = esq(i);
  while (c < h->n) {
    int c2 = dir(i);
    if (c2 < h->n && h->v[c2] > h->v[c])
      c = c2;
    if (h->v[c] < h->v[i])
      break ;
    troca(h,i,c);
    i = c;
    c = esq(i);
  }

}
```

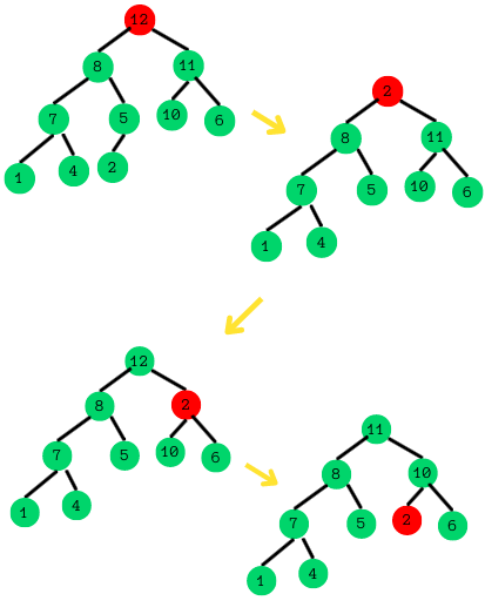
```
static void troca (Heap* h, int i, int j){

  float tmp = h->v[i]; h->v[i] =
  h->v[j];
  h->v[j] = tmp;

}
```

```
float heap_retira (Heap* h){

  float r = h->v[0];
  h->v[0] = h->v[--h->n];
  desce(h,0);
  return r;

}
```

• **Remoção:** A remoção em uma fila de prioridade envolve a exclusão do elemento de maior prioridade (no caso de um heap máximo) ou menor prioridade (no caso de um heap mínimo). Após a remoção, é necessário reorganizar a estrutura do heap para restaurar a propriedade do heap.

As três funções ao lado são responsáveis pela operação de remoção de um elemento do heap. Inicialmente, a função “desce” procura ajustar a posição de

um elemento do heap para baixo, de forma a manter a propriedade de heap máximo. Ela recebe como parâmetro um ponteiro para a estrutura Heap e um índice que representa a posição do elemento a ser ajustado. Logo após, compara o elemento com seus filhos e troca com o maior filho, se necessário, para manter a propriedade do heap máximo. O laço while auxilia para que o programa fique executando até encontrar o maior filho de toda a árvore. Já a função “troca”, é responsável por trocar a posição de dois elementos no heap. Ela recebe como parâmetro um ponteiro para a estrutura heap e dois índices que representam as posições dos elementos a serem trocados. Finalmente, a função “heap_retira” remove e retorna o elemento de maior prioridade do heap, recebendo como parâmetro um ponteiro para o heap e substituindo o elemento raiz pelo último elemento do heap.



• **Impressão:** Para que seja possível fazer uma impressão, o heap deve estar preenchido. Dessa forma, é feito uma verificação para caso esteja vazio. A partir disso, o heap sempre percorrerá até a posição atual e verificando se há um filho da esquerda ou da direita nesse determinado nó. Após o print da fila em ordem do de maior prioridade para o menor. Veja abaixo um exemplo de implementação:

```
int isEmpty(Heap* heap) {
   return heap->tail == -1;
}
```

```
void printHeap(Heap* heap) {
   if (heap == NULL || isEmpty(heap)) {
      printf("Heap vazio.\n");
      return;
   }

   printf("Fila em ordem de prioridade maxima: ");
   for (int i = 0; i <= heap->tail; i++) {
      printf("%d ", heap->heap[i]);
   }
```

A função isEmpty verifica se o heap está vazio, ou seja, se não contém nenhum elemento. Ela recebe como parâmetro um ponteiro para a estrutura do heap (Heap* heap) e verifica se o atributo tail desse heap é igual a -1. O atributo tail representa o índice do último elemento inserido no

heap. Se o tail for -1, significa que o heap está vazio, e a função retorna 1 (verdadeiro), caso contrário, retorna 0 (falso). Já a função printHeap imprime os elementos do heap em ordem de prioridade máxima. Ela também recebe um ponteiro para a estrutura do heap como parâmetro (Heap* heap). A função verifica se o heap é nulo (heap == NULL) ou se está vazio usando a função isEmpty(heap). Se o heap for nulo ou vazio, a função imprime a mensagem "Heap vazio." e retorna. Caso o heap não seja vazio, a função itera sobre os elementos do heap, começando do índice 0 até o índice tail. A cada iteração, ela imprime o elemento correspondente no heap usando heap->heap[i]. Essa abordagem percorre o array do heap e imprime os elementos na ordem em que estão armazenados. Como os heaps têm a propriedade de que o elemento de maior prioridade está na raiz, percorrer o array dessa forma resulta na impressão dos elementos em ordem de prioridade máxima. Ao final, a função imprime uma quebra de linha para melhor formatação.

4. Referências Bibliográficas

- BM, João Arthur. Heap - Estrutura de Dados. Disponível em: https://joaoarthurbm.github.io/eda/posts/heap/. Acesso em: 16 jun. 2023.

- GeeksforGeeks. "Priority Queue Set 1 (Introduction)." Disponível em: https://www.geeksforgeeks.org/priority-queue-set-1-introduction/. Acesso em: 26 de junho de 2023.

- CELES, Waldemar. Introdução a Estruturas de Dados - Com Técnicas de Programação em C. [Digite o Local da Editora]: Grupo GEN, 2016. E-book. ISBN 9788595156654. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595156654/. Acesso em: 27 jun. 2023.