ACH2023 – turmas 04 e 94 – EP1 2º Semestre de 2020

Fila de Prioridade

Filas de prioridades são estruturas de dados que gerenciam um conjunto de elementos, cada um com uma prioridade associada.

Dentre as operações previstas numa fila de prioridade estão:

- inserção de um elemento;
- exclusão do elemento de prioridade máxima;
- aumento de prioridade de um elemento;
- redução de prioridade de um elemento;
- consulta da prioridade de um elemento;
- consulta à quantidade de elementos (tamanho) da fila.

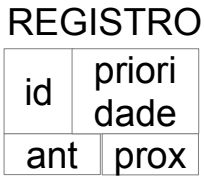
Em filas de prioridade, é desejável que todas estas operações sejam realizadas de maneira eficiente (mas isto estará fora do contexto deste EP). A eficiência dessas operações dependerá da maneira que a fila de prioridade foi implementada (e de sua estrutura subjacente).

Apesar de serem chamadas de filas (assunto que será visto no futuro dentro desta disciplina), a implementação que utilizaremos neste EP é baseada em **listas duplamente ligadas ordenadas e com nó cabeça** e também utilizaremos um **arranjo auxiliar**. Assim os dois conceitos principais necessários para implementação deste EP são: listas duplamente ligadas ordenadas e um arranjo de elementos. Observação a lista utilizada será ordenada de forma **decrescente**.

A seguir serão apresentadas as estruturas de dados envolvidas nesta implementação e como elas serão gerenciadas.

A estrutura básica será o *ELEMENTO*, que contém quatro campos: *id* (identificador inteiro do elemento), *prioridade* (número do tipo *float* com a prioridade do elemento), *ant* (ponteiro para o elemento anterior, isto é, o que possui prioridade imediatamente maior do que a do elemento atual), e *prox* (ponteiro para o elemento posterior, isto é, o que possui prioridade imediatamente menor do que a do elemento atual).

```
typedef struct aux {
   int id;
   float prioridade;
   struct aux* ant;
   struct aux* prox;
} ELEMENTO, * PONT;
```



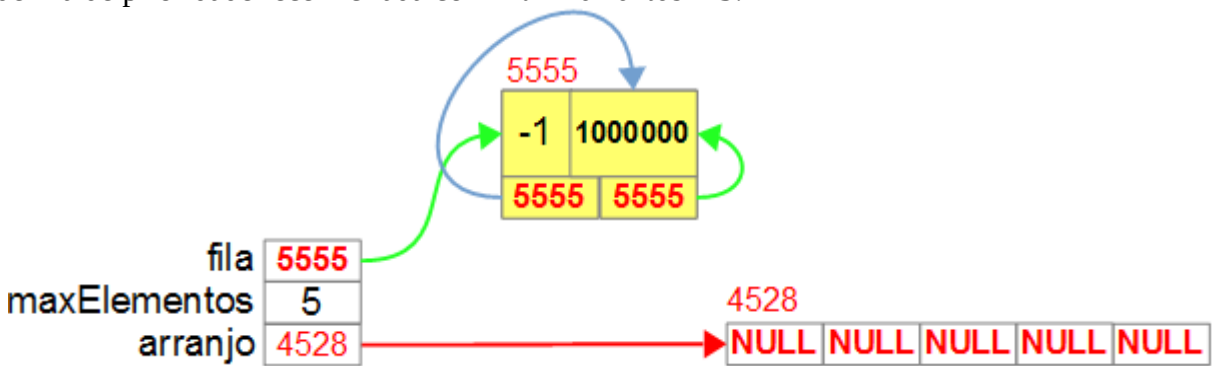
A estrutura *FILADEPRIORIDADE* possui três campos: *fila* é um ponteiro para elementos do tipo *ELEMENTO* e corresponde ao ponteiro para o nó cabeça da lista duplamente ligada e ordenada de elementos, ordenados da maior prioridade para a menor **(esta lista de elementos possuirá nó-cabeça e será circular)**; *maxElementos* é um campo do tipo inteiro que representa a quantidade máxima de elementos válidos permitidos na fila de prioridade atual (os *ids* válidos dos elementos valerão de 0 [zero] até *maxElementos-1*) [o nó-cabeça não conta como elemento válido]; *arranjo* corresponde a um ponteiro para um arranjo de ponteiros para elementos do tipo *ELEMENTO*. Na inicialização de uma fila de prioridades este arranjo é criado com todos seus valores valendo *NULL*. Já que os *ids* válidos variam de 0 a *maxElementos*-1 então há uma posição específica para guardar o endereço de cada ELEMENTO (quando ele for criado) neste arranjo, permitindo o acesso rápido a um elemento qualquer a partir de seu *id*. O nó-cabeça não é considerado um elemento válido (do ponto de vista do usuário) e não é apontado por este arranjo. Você pode considerar, neste EP, que **nenhum elemento válido terá prioridade maior do que 999999**.

```
typedef struct {
  PONT fila;
  int maxElementos;
  PONT* arranjo;
} FILADEPRIORIDADE, * PFILA;
```

A função criarFila é responsável por criar uma nova fila de prioridade que poderá ter até *max elementos* e deve retornar o endereço dessa fila de prioridades. Observe que o arranjo de ponteiros para elementos já é criado e tem seus valores inicializados nessa função.

```
PFILA criarFila(int max){
   PFILA res = (PFILA) malloc(sizeof(FILADEPRIORIDADE));
   res->maxElementos = max;
   res->arranjo = (PONT*) malloc(sizeof(PONT)*max);
   int i;
   for (i=0;i<max;i++) res->arranjo[i] = NULL;
   PONT cabeca = (PONT) malloc(sizeof(ELEMENTO));
   res->fila = cabeca;
   cabeca->ant = cabeca;
   cabeca->prox = cabeca;
   cabeca->id = -1;
   cabeca->prioridade = 1000000;
   return res;
}
```

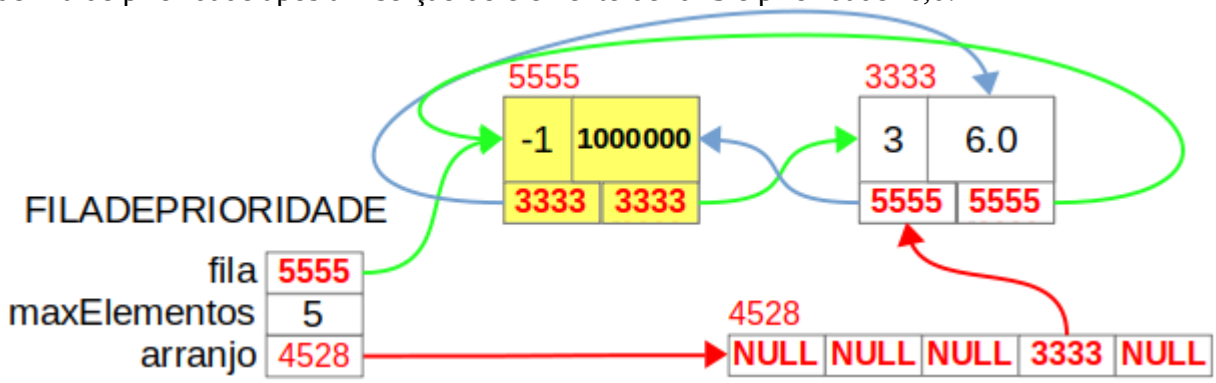
Exemplo de fila de prioridade recém criada com *maxElementos* = 5:



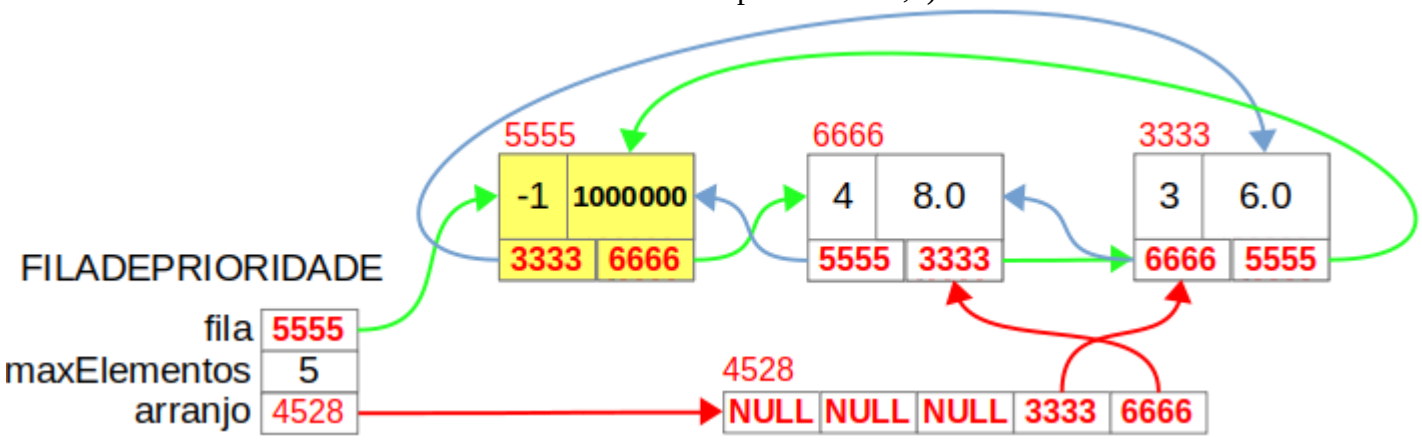
Ao se inserir um novo elemento na estrutura, este deverá ser incluindo na lista duplamente ligada ordenada de forma decrescente (cujo nó-cabeça é apontado pelo campo *fila*) e também deverá ter seu endereço armazenado na respectiva posição do arranjo (apontado pelo campo *arranjo*).

Observação: se houver dois elementos com a mesma **prioridade**, tanto faz qual dos dois ficará na frente do outro na lista duplamente ligada ordenada de forma decrescente (nos testes deste EP não serão inseridos elementos com prioridade repetida, mas isso costuma ser permitido em filas de prioridade).

Exemplo de fila de prioridade após a inserção do elemento de id=3 e prioridade=6,0:



Exemplo de fila de prioridade após a inserção do elemento de id=4 e prioridade=8,0 (depois da inserção do elemento de id=3 e prioridade=6,0).



Ao se modificar a prioridade de um elemento da fila de prioridade, este deverá ser reposicionado, caso necessário, dentro da lista duplamente ligada ordenada de forma decrescente.

Funções que deverão ser implementadas no EP

int tamanho(PFILA f): função que recebe o endereço de uma fila de prioridade e retorna o número de elementos válidos na fila (ou mais precisamente, da lista duplamente ligada ordenada, sem contar o nó-cabeça).

bool inserirElemento(PFILA f, int id, float prioridade): função que recebe o endereço de uma fila de prioridade, o identificador do novo elemento e o valor de sua prioridade.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxElementos*);
- o identificador seja válido, mas já houver um elemento com esse identificador na fila.

Caso contrário, a função deverá alocar memória para esse novo elemento, colocar o endereço dele no arranjo de elementos e inseri-lo na posição correta da fila (de acordo com sua prioridade), acertando todos os ponteiros necessários e retornar *true*.

bool aumentarPrioridade(PFILA f, int id, float novaPrioridade): função que recebe o endereço de uma fila de prioridade, o identificador do elemento e o novo valor de sua prioridade.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxElementos*);
- o identificador seja válido, mas não haja um elemento com esse identificador na fila.
- o identificador seja válido, mas sua prioridade já seja maior ou igual à prioridade passada como parâmetro da função.

Caso contrário, a função deverá trocar a prioridade do elemento, reposicionar (se necessário) o elemento na fila de prioridade (lista duplamente ligada e ordenada de forma decrescente) e retornar *true*. Observação: esta função não deverá criar um novo elemento.

bool reduzirPrioridade(PFILA f, int id, float novaPrioridade): função que recebe o endereço de uma fila de prioridade, o identificador do elemento e o novo valor de sua prioridade.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a *maxElementos*);
- o identificador seja válido, mas não haja um elemento com esse identificador na fila.
- o identificador seja válido, mas sua prioridade já seja menor ou igual à prioridade passada como parâmetro da função.

Caso contrário, a função deverá trocar a prioridade do elemento, reposicionar (se necessário) o elemento na fila de prioridade (lista duplamente ligada e ordenada de forma decrescente) e retornar *true*. Observação: esta função não deverá criar um novo elemento.

PONT removerElemento(PFILA f): esta função recebe como parâmetro o endereço de uma fila de prioridade e deverá retornar *NULL* caso a fila não possua nenhum elemento válido (ou seja, possua apenas o nó-cabeça). Caso contrário, deverá retirar o primeiro elemento válido da lista duplamente ligada (acertando os ponteiros necessários), colocar o valor *NULL* na posição correspondente desse elemento no *arranjo* e retornar o endereço do respectivo elemento. A memória desse elemento **não deverá ser liberada**, pois o usuário pode querer usar esse elemento para alguma coisa.

bool consultarPrioridade(PFILA f, int id, float resposta)*: função que recebe o endereço de uma fila de prioridade, o identificador do elemento e um endereço para uma memória do tipo *float*.
Esta função deverá retornar *false* caso:

- o identificador seja inválido (menor que zero ou maior ou igual a maxElementos);
- o identificador seja válido, mas não haja um elemento com esse identificador na fila.

Caso contrário, a função deverá colocar na memória apontada pela variável *resposta* o valor da prioridade do respectivo elemento e retornar *true*.

Informações gerais:

Os EPs desta disciplina são trabalhos individuais que devem ser submetidos pelos alunos via sistema TIDIA até às 23:55h (com margem de tolerância de 60 minutos).

Vocês receberão três arquivos para este EP:

- filaDePrioridade.h que contém a definição das estruturas, os *includes* necessários e o cabeçalho/assinatura das funções. Vocês não deverão alterar esse arquivo.
- filaDePrioridade.c que conterá a implementação das funções solicitadas (e funções adicionais, caso julguem necessário). Este arquivo já contém o esqueleto geral das funções e alguns códigos implementados.
- usaFilaDePrioridade.c

Você deverá submeter **apenas** o arquivo filaDePrioridade.c, porém renomeie este arquivo para seuNúmeroUSP.c (por exemplo, **1234567.c**) antes de submeter.

Não altere a assinatura de nenhuma das funções e não altere as funções originalmente implementadas (*exibirLog* e *criarFila*).

Nenhuma das funções que vocês implementarem deverá imprimir algo. Para testa/*debugar* o programa, você pode imprimir coisas, porém, na versão a ser entregue ao professor, suas funções não deverão imprimir nada (exceto pela função *exibirLog* que já imprime algumas informações).

Você poderá criar novas funções (auxiliares), mas não deve alterar o arquivo filaDePrioridade.h. eu código será testado com uma versão diferente do arquivo usaFilaDePrioridade.c. Suas funções serão testadas individualmente e em grupo.

Todos os trabalhos passarão por um processo de verificação de plágios. Em caso de plágio, todos os alunos envolvidos receberão nota zero.