Algoritmos e Estrutura de Dados

Filas de prioridade

Baseado nos materiais do prof:
Dr. Wesley Romão

Prof. Nilton Luiz Queiroz Jr.

Filas de prioridade

- Filas comuns não permitem tratar de modo privilegiado alguns elementos;
  - Toda inserção é feita no fim, e toda remoção é feita no início;
- Algumas vezes isso se faz necessário;
- Numa fila de prioridade, usa-se um valor prioridade para cada elemento da fila;
  - As inserções e/ou remoções levam em conta a prioridade;

Filas de prioridade

- Uma lista de tarefas que devemos fazer muitas vezes é estruturada como uma fila de prioridades;
  - Algumas tarefas, como assistir um episódio de uma série não podem ser adiadas;
  - Outras como estudar para a avaliação da próxima semana tem que ser feitas com mais urgência;
- Dessa forma, as tarefas podem ser efetuadas pela ordem de sua importância, e escolhemos qual a mais urgente/importante a ser realizada;

Filas de prioridade

- Várias áreas da computação usam filas de prioridade:
  - Sistemas operacionais;
  - Algoritmos de ordenação;
  - Cálculos de distância em grafos;
  - Etc;

Filas de prioridade

- Existem 3 principais maneiras para implementar filas de prioridade:
  - Árvores binárias de busca;
  - Heaps binários;
  - Listas;

Filas de prioridade

- Apesar do nome fila, uma fila de prioridade não obedece as políticas de inserção e remoção de uma fila comum;
  - Ou seja, a política FIFO não é válida em uma fila de prioridade;
- Apesar disso, o nome já está associado a estrutura;

Filas de prioridade

- As filas de prioridade são construídas sobre dois principais operadores:
  - Inserção de um elemento;
  - Remoção de um elemento;
- A maneira que esses operadores são construídos depende de como a lista é organizada;
  - Fila ordenada;
    - A remoção deve ser sempre no início da fila;
    - A inserção deve ser feita de maneira ordenada;
  - Fila não ordenada;
    - A remoção ocorre em qualquer ponto da fila;
    - A inserção é feita sempre no final da fila;

Filas de prioridade

- Todos nós podem conter um atributo chave que pode ser usado para ordenar a fila;
  - A escolha de qual chave (maior ou menor) irá corresponder a maior prioridade é uma questão de implementação da fila;
- Não é necessário esse atributo chave ser algo “externo” ao tipo de dado da fila;
  - A chave pode ser parte dos atributos dos dados armazenados na fila;

Tipos de fila de prioridade

- Em geral podemos dividir as filas de prioridade em dois tipos:
  - Fila de prioridade ascendente;
    - Os itens são inseridos arbitrariamente;
    - O item com menor chave é o removido;
  - Fila de prioridade descendente;
    - Os itens são inseridos arbitrariamente;
    - O item com maior chave é o removido;

Exemplo

- Supondo uma fila de atendimento:
  - A prioridade poderia ser a idade da pessoa que está na fila;
  - Em caso de empate pode-se adotar a política de uma fila normal (FIFO);
- Exemplo:

Ordem de chegada
1. Gisele, prioridade 15
2. Natália, prioridade 20
3. Manoel, prioridade 18
4. Giovana, prioridade 20
5. Carolina, prioridade 18

Ordem de atendimento
1. Natália, prioridade 20
2. Giovana, prioridade 20
3. Manoel, prioridade 18
4. Carolina, prioridade 18
5. Gisele, prioridade 15

Implementação

- Para implementar uma fila de prioridade iremos adotar as seguintes estruturas:

```
struct tipo_item{
    int chave;
    /*outros campos*/
};

struct tipo_celula{
    struct tipo_item item;
    struct tipo_celula *prox;
    struct tipo_celula *ant;
    int prioridade;
};

struct tipo_fila_p{
    struct tipo_celula *primeiro;
};
```

Implementação

- Para implementar uma fila de prioridade iremos implementar as seguintes operações:
  - Inicializar fila;
  - Verificar se a fila está vazia;
  - Inserir elemento;
  - Obter o elemento de maior prioridade;
    - Será uma fila descendente;
  - Remover elemento;
  - Exibir a fila;
- Por questões de simplicidade do código a fila será implementada em uma lista circular, duplamente ligada com sentinela;
- A fila será implementada de maneira não ordenada;

Implementação

- Inicializar fila:

```
void inicializa(struct tipo_fila_p *pq){
    pq->primeiro=(struct tipo_celula *)malloc(sizeof(struct tipo_celula ));
    pq->primeiro->prox=pq->primeiro;
    pq->primeiro->ant=pq->primeiro;
}
```

- Verificar se a fila está vazia:

```
int vazia(struct tipo_fila_p *pq){
    return pq->primeiro->prox == pq->primeiro;
}
```

Implementação

- Exibir a fila:
  - Observe que o procedimento não mostra a fila ordenada pelas prioridades;

```
void escreve(struct tipo_fila_p *pq){
    printf("======================\n");
    struct tipo_celula *ptr=pq->primeiro->prox;
    while(ptr!=pq->primeiro){
        printf("item:%d\n",ptr->item.chave);
        printf("prioridade:%d\n",ptr->prioridade);
        ptr=ptr->prox;
        printf("--------------------\n");
    }
    printf("\n");
    printf("======================\n");
}
```

Implementação

- Inserir elemento;
  - A inserção deve ser uma inserção no fim;

```
void insere(struct tipo_fila_p *pq, struct tipo_item x, int prioridade){
    /*algoritmo*/
}
```

Implementação

- Inserir elemento;
  - A inserção deve ser uma inserção no fim;

```
void insere(struct tipo_fila_p *pq, struct tipo_item x, int prioridade){
    struct tipo_celula *novo;
    novo=(struct tipo_celula *)malloc(sizeof(struct tipo_celula));
    novo->item=x;
    novo->prioridade=prioridade;
    novo->prox=pq->primeiro;
    novo->ant=pq->primeiro->ant;
    pq->primeiro->ant->prox = novo;
    pq->primeiro->ant=novo;
}
```

Implementação

- Obter o elemento com maior prioridade;
  - Esse função deve retornar um ponteiro para o elemento com maior prioridade;
  - Em caso de elementos com a mesma prioridade, o primeiro a entrar na fila deve ser o obtido;

```
struct tipo_celula *encontra_maior(struct tipo_fila_p *pq){
    /*algoritmo*/
}
```

Implementação

- Obter o elemento com maior prioridade;
  - Esse função deve retornar um ponteiro para o elemento com maior prioridade;
  - Em caso de elementos com a mesma prioridade, o primeiro a entrar na fila deve ser o obtido;

```
struct tipo_celula *encontra_maior(struct tipo_fila_p *pq){
    struct tipo_celula *ptr,*maior;
    ptr=pq->primeiro->prox;
    maior=pq->primeiro->prox;
    while(ptr!=pq->primeiro){
        if(ptr->prioridade> maior->prioridade)
            maior=ptr;
        ptr=ptr->prox;
    }
    if(maior != pq->primeiro)
        return maior;
    else
        return NULL;
}
```

Implementação

- Remover um elemento da fila;
  - A remoção deve encontrar o elemento com maior prioridade e o excluir;
    - Essa implementação retorna a prioridade do elemento, e também o elemento;

```
int remove_fila_p(struct tipo_fila_p *pq, struct tipo_item *x, int *prioridade){
    /*algoritmo*/
}
```

Implementação

- Remover um elemento da fila;
  - A remoção deve encontrar o elemento com maior prioridade e o excluir;
    - Essa implementação retorna a prioridade do elemento, e também o elemento;

```
int remove_fila_p(struct tipo_fila_p *pq, struct tipo_item *x, int *prioridade){
    struct tipo_celula *del;
    del = encontra_maior(pq);
    if (del!=NULL){
          del->ant->prox=del->prox;
          del->prox->ant=del->ant;
          *prioridade = del->prioridade;
          *x=del->item;
          return 1;
    }else{
          return 0;
    }
}
```

Exercícios

1. Faça uma operação que identifique quantos elementos de uma determinada prioridade existem na fila.
2. Faça uma operação que receba uma chave de um item e permita a alteração de prioridade do elemento com aquela chave. Assuma que não existem chaves repetidas.
3. Faça uma função que permita descobrir quantos elementos estão a frente de um determinado elemento na fila.

Exercícios

2. Uma agência de banco precisa de um sistema para o atendimento de seus clientes, tanto os que possuem conta no banco, quanto os que não possuem. Para cada tipo de cliente existem tipos de prioridade diferentes, exibidos na tabela abaixo:

Tipo de cliente	Prioridade
Acima de 60 anos e com conta na agência	Muito Alta
Acima de 60 anos e sem conta na agência	Alta
Até 60 anos e com conta na agência	Média
Até 60 anos e sem conta na agência	Baixa

Faça um programa com as opções de emitir senhas e chamar os clientes para o atendimento