Algoritmos e Estrutura de Dados

Deque

Prof. Nilton Luiz Queiroz Jr.

Deque

- Um deque (Double Ended Queue) é conjunto de itens a partir do qual podem ocorrer inserções e remoções em suas duas extremidades;
- São mais genéricos que filas e pilhas;
- Possuem algumas propriedades similares a um baralho de cartas;
- Podem ter:
  - Entrada restrita:
    - Somente uma extremidade permite entrada;
  - Saída restrita;
    - Somente uma extremidade permite a saída;

Deques

- Deques podem ser aplicados em:
  - Histórico de páginas visitadas em um navegador de internet;
    - Por mais que o histórico funcione como uma pilha, quando ele começa a crescer, pode ser interessante descartar os elementos que estão na base, dessa forma, temos um deque com entrada restrita;
  - Mecanismos de desfazer alterações em documentos;
    - Ctrl+Z;
  - Escalonamento para multiprocessadores;
    - Algoritmo A-Steal (Adaptive stealing thread scheduler);

Deques

- São implementadas quatro principais operações em deques:
  - Inserção na extremidade direita;
  - Remoção na extremidade direita;
  - Inserção na extremidade esquerda;
  - Remoção na extremidade esquerda;

Deques

- Deques podem ser implementados:
  - Estaticamente;
  - Dinamicamente;

Implementação estática

- As operações a serem implementadas em um deque estático são:
  - Inicializar;
  - Verificar se está vazio;
  - Verificar se está cheio;
  - Inserir na extremidade direita;
  - Inserir na extremidade esquerda;
  - Remover na extremidade direita;
  - Remover na extremidade esquerda;

Implementação estática

- A ideia de um deque estático é semelhante a de uma fila estática circular;
  - Manter um valor para a extremidade esquerda e um para a extremidade direita;
    - Quando ambos são iguais o deque está vazio;
    - Quando o “índice” sucessor ao da extremidade direita for o índice da extremidade esquerda o deque está cheio;
  - Porém, no deque tanto “a esquerda” quanto “a direita” podem andar para os “dois lados”;
    - Dessa forma precisamos de funções para:
      - Obter o índice do próximo;
      - Obter o índice do anterior;

Implementação estática

- As estruturas e constantes usadas no deque estático serão:

```
#define TAM 100

struct tipo_item{
    int chave;
};

struct tipo_deque{
    int esquerda,direita;
    struct tipo_item dados[TAM];
};
```

Implementação estática

- Obter próximo:

```
int proximo(int pos){
    return (pos+1)%TAM;
}
```

- Obter anterior:

```
int anterior(int pos){
    return (pos+(TAM-1))%TAM;
}
```

Implementação estática

- Inicializar o deque:

```
void inicializa(struct tipo_deque *d){
    d->esquerda=0;
    d->direita=0;
}
```

- Verificar se o deque está vazio:

```
int vazio(struct tipo_deque *d){
    return d->esquerda==d->direita;
}
```

- Verificar se o deque está cheio:

```
int cheio(struct tipo_deque *d){
    return proximo(d->direita)==d->esquerda;
}
```

Implementação estática

- Inserção na extremidade direita;

```
int insere_dir(struct tipo_deque *d, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação estática

- Inserção na extremidade direita;

```
int insere_dir(struct tipo_deque *d, struct tipo_item x){
    if(!cheio(d)){
        d->dados[d->direita]=x;
        d->direita=proximo(d->direita);
        return 1;
    }else{
        return 0;
    }
}
```

Implementação

- Inserção na extremidade esquerda

```
int insere_esq(struct tipo_deque *d, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação estática

- Inserção na extremidade esquerda

```
int insere_esq(struct tipo_deque *d, struct tipo_item x){
    if(!cheio(d)){
        d->esquerda=anterior(d->esquerda);
        d->dados[d->esquerda]=x;
        return 1;
    }else{
        return 0;
    }
}
```

Implementação estática

- Remoção na extremidade direita

```
int remove_dir(struct tipo_deque *d, struct tipo_item *x){
    /*algoritmo*/
}
```

Implementação estática

- Remoção na extremidade direita

```
int remove_dir(struct tipo_deque *d, struct tipo_item *x){
    if(!vazio(d)){
        d->direita=anterior(d->direita);
        *x=d->dados[d->direita];
        return 1;
    }else{
        return 0;
    }
}
```

Implementação estática

- Remoção na extremidade esquerda

```
int remove_esq(struct tipo_deque *d, struct tipo_item *x){
    /*algoritmo*/
}
```

Implementação estática

- Remoção na extremidade esquerda

```
int remove_esq(struct tipo_deque *d, struct tipo_item *x){
    if(!vazio(d)){
        *x=d->dados[d->esquerda];
        d->esquerda=proximo(d->esquerda);
        return 1;
    }else{
        return 0;
    }
}
```

Implementação estática

- Escrever os valores do deque da esquerda para a direita;

```
void escreve_ed(struct tipo_deque *d){
    int i;
    printf("==================\n");
    for(i=d->esquerda; i!=d->direita; i=proximo(i)){
          printf("%d ",d->dados[i].chave);
    }
    printf("\n");
    printf("==================\n");
}
```

Implementação dinâmica

- As operações a serem implementadas em um deque dinâmico são:
  - Inicializar;
  - Verificar se está vazio;
  - Inserir na extremidade direita;
  - Inserir na extremidade esquerda;
  - Remover na extremidade direita;
  - Remover na extremidade esquerda;
  - Escrever o deque
- Um deque dinâmico pode ser implementado em qualquer tipo de lista;
  - Porém, quando implementado em listas duplamente ligadas todas as remoções e inserções têm a mesma complexidade assintótica (constante);
    - Desde que se mantenha ponteiros para o acesso ao início e ao fim;
  - Já quando implementados em listas simplesmente encadeadas a remoção em uma extremidade sempre terá um custo mais alto (linear);

Implementação dinâmica

- Para implementar dinamicamente iremos considerar 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;
};

struct tipo_deque{
    struct tipo_celula *sentinela;
};
```

Implementação dinâmica

- Inicializar o deque

```
void inicializa(struct tipo_deque *d){
    d->sentinela=(struct tipo_celula *)malloc(sizeof(struct tipo_celula ));
    d->sentinela->prox=d->sentinela;
    d->sentinela->ant=d->sentinela;
}
```

- Verificar se o deque está vazio

```
int vazio(struct tipo_deque *d){
    return d->sentinela->prox == d->sentinela;
}
```

Implementação dinâmica

- Inserção na extremidade direita

```
void insere_dir(struct tipo_deque *d, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Inserção na extremidade direita

```
void insere_dir(struct tipo_deque *d, struct tipo_item x){
    struct tipo_celula *novo;
    novo=(struct tipo_celula *)malloc(sizeof(struct tipo_celula));
    novo->item=x;
    novo->prox=d->sentinela;
    novo->ant=d->sentinela->ant;
    d->sentinela->ant->prox = novo;
    d->sentinela->ant=novo;
}
```

Implementação dinâmica

- Inserção na extremidade esquerda

```
void insere_esq(struct tipo_deque *d, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Inserção na extremidade esquerda

```
void insere_esq(struct tipo_deque *d, struct tipo_item x){
    struct tipo_celula *novo;
    novo=(struct tipo_celula *)malloc(sizeof(struct tipo_celula));
    novo->item=x;
    novo->prox = d->sentinela->prox;
    novo->ant = d->sentinela;
    d->sentinela->prox->ant = novo;
    d->sentinela->prox = novo;
}
```

Implementação dinâmica

- Remoção na extremidade direita

```
int remove_dir(struct tipo_deque *d, struct tipo_item *x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Remoção na extremidade direita

```
int remove_dir(struct tipo_deque *d, struct tipo_item *x){
    struct tipo_celula *ptr;
    if(!vazio(d)){
        ptr=d->sentinela->ant;
        *x=ptr->item;
        ptr->ant->prox=ptr->prox;
        ptr->prox->ant=ptr->ant;
        free(ptr);
        return 1;
    }else{
        return 0;
    }
}
```

Implementação dinâmica

- Remoção na extremidade esquerda

```
int remove_esq(struct tipo_deque *d, struct tipo_item *x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Remoção na extremidade esquerda

```
int remove_esq(struct tipo_deque *d, struct tipo_item *x){
    struct tipo_celula *ptr;
    if(!vazio(d)){
        ptr=d->sentinela->prox;
        *x=ptr->item;
        ptr->prox->ant = ptr->ant;
        ptr->ant->prox = ptr->prox;
        free(ptr);
        return 1;
    }else{
        return 0;
    }
}
```

Exercícios

1. Implemente uma operação para escrever o deque da direita para a esquerda, considerando uma implementação:
   a. Estática;
   b. Dinâmica;
2. Considerando uma implementação de deque estática, implemente uma operação para exibir o valor da extremidade:
   a. Direita;
   b. Esquerda;
3. Considerando uma implementação de deque dinâmica, implemente uma operação para exibir o valor da extremidade:
   a. Direita;
   b. Esquerda;