MAT A40 - Estrutura de Dados e Algoritmos I [1]

Dr. George Lima
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade Federal da Bahia

[1] Este módulo foi baseado em conteúdos presentes em "A. V. Aho, J. D. Ullman, *Foundations of Computer Science* (C edition). Computer Science Press (W.H. Freeman), 1995." e "Paulo Feofiloff. *Algoritmos em linguagem C*. Campus/Elsevier, 2009.".

Módulo 3

Fila, Pilha e Aplicações

O tipo Fila

Definição

Uma fila é uma sequência ordenada e finita de zero ou mais elementos, não necessariamente distintos. O primeiro elemento da sequência é aquele que foi inserido a mais tempo enquanto o último é o mais recentemente inserido (ordem FIFO). A remoção retira da fila o primeiro elemento enquanto a inserção inclui o novo elemento na última posição. Se todos os seus elementos são do mesmo tipo T, a fila é dita do tipo T.

O tipo Fila

Possível conjunto de operações sobre listas:

- Inserção: Se $Q = (a_1, a_2, \ldots, a_n)$, `enqueue`$(x, L)$ insere x em Q após a_n.

O tipo Fila

Possível conjunto de operações sobre listas:

- Remoção: Se $Q = (a_1, a_2, \ldots, a_n)$, então `dequeue`$(L)$ remove a_1 de Q.

O tipo Fila

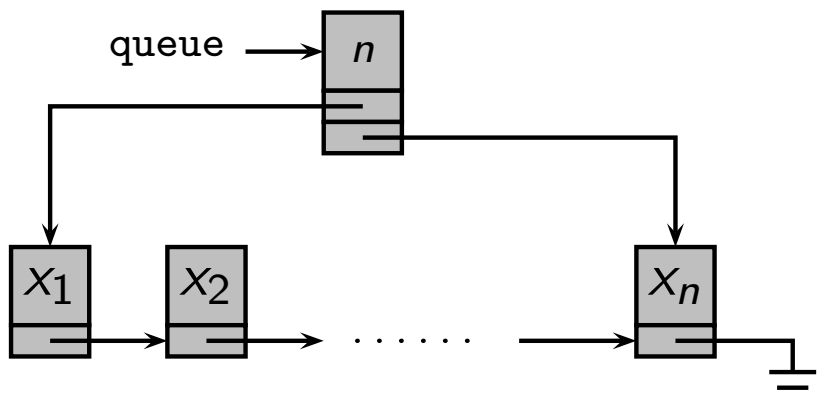
Possível conjunto de operações sobre listas:

- Condições limites: A operação `isempty(`Q`)` retorna verdade se Q não está vazia e falso caso contrário. A operação `isfull(`Q`)` retorna verdade se Q não há mais espaço para novas inserções em Q e falso caso contrário.

Implementação de fila através de lista encadeada

Descrição

Uma fila pode ser construída através de uma lista encadeada para a qual inserções ocorrem no final da lista e remoções ocorrem no início da lista.



TAD: Fila (com encadeamento) em C

```c
typedef struct cell cell_t;
typedef int element_t;
typedef struct queue queue_t;
typedef enum {FALSE, TRUE} bool;

struct queue { // queue TAD
  cell_t *first, *last;
  unsigned int
    size, //  no. de elementos na fila
  maxsize; // max. no. de elementos permitidos
};

struct cell { // para lista encadenada
      element_t * element;
      cell_t *next;
};
```

- Lista sem cabeça: `first` = `last` = `NULL` se fila vazia.
- Estrutura queue guarda informações relevantes da fila.

TAD: Fila (com encadeamento) em C

Funções básicas:

```
1 cell_t *newcell(void);  // Cria nova celula na lista
2 queue_t *newQueue(unsigned int); // Cria nova fila
3
4 bool enqueue(element_t *, queue_t *); // insere elem.
5 element_t *dequeue(queue_t *); // remove elemento
```

Funções auxiliares:

```
1 bool isempty(queue_t *) // Verdade sse fila vazia
2 bool isfull(queue_t *) // Verdade sse fila cheia
3 void display(queue_t *);
```

Instanciando uma fila

```
1 queue_t *newQueue(unsigned int n) {
2   queue_t *p = malloc ( sizeof (queue_t));
3   if (!p) return NULL;
4
5   p -> first = p -> last = NULL;
6   p -> size  = 0;
7   p -> maxsize = n;
8   return p;
9 }
```

Incluindo um elemento na fila

```
1  bool enqueue(element_t *e, queue_t *q) {
2
3    if (isfull(q)) return FALSE;
4
5    cell_t *p = newCell();
6    if (!p) return FALSE;
7
8    if (isempty(q)) q -> first = p;
9    else q -> last -> next = p;
10
11   q -> last = p;
12   q -> size++;
13   p -> element = e;
14
15   return TRUE;
16 }
```

Removendo um elemento na fila

Nesta versão, a função dequeue devolve (ponteiro para) o elemento contido na primeira célula da fila.

```
1  element_t *dequeue(queue_t *q) {
2
3    if (isempty(q)) return NULL;
4
5    cell_t *p = q -> first;
6    element_t *e = q -> first -> element;
7
8    q -> first = q -> first -> next;
9    q -> size--;
10
11   if (isempty(q)) q -> first = q -> last = NULL;
12
13   free(p);
14   return (e);
15 }
```

Exercícios de fixação

Redefina as funções anteriormente vistas para lidar com os seguintes cenários:

1. Considere que o elemento a ser armazenado na fila é uma estrutura contendo alguns campos. Cada célula da fila deve conter um ponteiro para cada elemento. Nesta implementação, considere que `dequeue` libera o espaço referente à célula retirada da fila. Defina o tipo `element_t` para representar a estrutura que contém as informações de cada elemento na fila.
2. Em muitos cenários, configura-se a fila como circular. Nesta configuração, o primeiro elemento, quando lido, pode ser novamente colocado na fila. Isto é muito comum em aplicações que processam a fila em ciclos, "servindo" cada elemento da fila, um por vez, com uma quantidade constante de unidades de serviço. Construa uma fila circular. Cada elemento da fila deve conter seu identificador e as unidades de serviço necessárias. Em cada ciclo de serviço, o elemento só pode obter, no máximo, k unidades de serviço. O elemento deve ser novamente enfileirado caso não seja possível servi-lo completamente. Neste caso, suas unidades de serviços restantes ficarão para serem a ele oferecidas no próximo ciclo de serviço. Execute sua implementação considerando como exemplo a seguinte fila e $k = 5$: $(1, 12)$, $(4, 45)$, $(2, 27)$, $(3, 234)$, $(10, 55)$, $(14, 99)$, $(8, 76)$. Cada tupla (i, k_i) representa o elemento de identificação i e as unidades de serviços k_i por ele requisitadas.

Implementação de fila através de vetor

Quando sabemos o tamanho máximo de uma fila, a sua implementação num vetor é mais eficiente.
TAD:

```
1  typedef enum {FALSE, TRUE} bool;
2  typedef int * element_t; // elementos como ponteiros
3  typedef struct queue queue_t;
4
5  struct queue {
6    unsigned int first,
7      last, maxsize;
8
9    element_t *qvector; // vetor de elementos
10 };
```

Instanciando uma nova fila

```
1  queue_t *newQueue(unsigned int n) {
2
3    queue_t *p =  malloc (sizeof (queue_t));
4    if (!p) return NULL;
5
6    p->qvector = malloc (n* sizeof (element_t));
7
8    if (!p->qvector) return NULL;
9
10   p -> first = p -> last = 0;
11   p -> maxsize = n;
12   return p;
13 }
```

Inserção e remoção de elementos

```
1  bool enqueue(element_t e, queue_t *q) {
2
3    if (isfull(q)) return FALSE;
4    q -> qvector[q -> last++] = e;
5    if (q -> last == q -> maxsize) q -> last = 0;
6
7    return TRUE;
8  }
9
10 element_t dequeue(queue_t *q) {
11
12   if (isempty(q)) return NULL;
13
14   int f = q -> first++;
15   if (q -> first == q -> maxsize) q -> first = 0;
16
17   return ((q -> qvector[f]));
18 }
```

Funções auxiliares

```
1 bool isempty(queue_t *q) {
2   return (q -> first == q -> last);
3 }
```

```
1 bool isfull(queue_t *q)  {
2   return ((q->last+1) % (q -> maxsize) == q -> first);
3   }
```

O tipo Pilha

Definição

Uma pilha é uma sequência ordenada e finita de zero ou mais elementos, não necessariamente distintos. O primeiro elemento da sequência é aquele que foi inserido a menos tempo enquanto o último é o menos recentemente inserido (ordem LIFO). A inserção insere e a remoção retira o primeiro elemento da pilha. Se todos os seus elementos são do mesmo tipo T, a pilha é dita do tipo T.

O tipo Pilha

Possível conjunto de operações sobre listas:

- Inserção: Se $S = (a_n, a_{n-1}, \ldots, a_1)$, `push`$(x, L)$ insere x em S antes de a_n.

O tipo Pilha

Possível conjunto de operações sobre listas:

- Remoção: Se $S = (a_n, a_{n-1}, \ldots, a_1)$, então $\mathtt{pop}(L)$ remove a_n de S.

O tipo Pilha

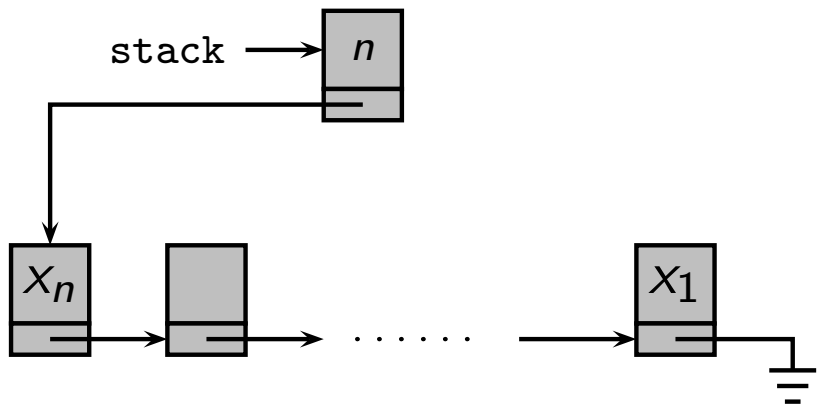
Possível conjunto de operações sobre listas:

- Condições limites: A operação `isempty`(S) retorna verdade se S está vazia e falso caso contrário. A operação `isfull`(S) retorna verdade se S não há mais espaço para novas inserções em S e falso caso contrário.

Implementação de pilha através de lista encadeada

Descrição

Uma pilha pode ser construída através de uma lista encadeada para a qual inserções e remoções ocorrem no início da lista.



TAD: Pilha (com encadeamento) em C

```c
typedef struct cell cell_t;
typedef int *element_t;
typedef struct stk stk_t;
typedef enum {FALSE, TRUE} bool;

struct stk {
  cell_t *top;
  unsigned int size;
};

struct cell {
      element_t element;
      cell_t *next;
};
```

- Lista sem cabeça: `first` = `last` = `NULL` se fila vazia.
- Estrutura `stack` guarda informações relevantes da pilha.

TAD: Pilha (com encadeamento) em C

Funções básicas:

```
1 bool     push(element_t , stk_t *); // insere elemento
2 element_t pop(stk_t *); // remove elemento
3 element_t top(stk_t *); // devolve elemento topo
4 cell_t *newcell(void); // Cria nova celula na lista
5 stk_t *newStk(void); // Cria nova pilha
```

Funções auxiliares:

```
1 void display(queue_t *);
2 unsigned int currentSize(stk_t *);
```

Instanciando uma pilha

```
1 stk_t *newStack() {
2   stk_t *p = malloc ( sizeof (stk_t));
3   if (!p) return NULL;
4
5   p -> top = NULL;
6   p -> size = 0;
7   return p;
8
9 }
```

Inserção e remoção

```
1  bool push(element_t e, stk_t *s) {
2
3    cell_t *p = newCell();
4    if (!p) return FALSE;
5
6    p -> next = s -> top = p;
7    s -> size++;
8    p -> element = e;
9    return TRUE;
10 }
11 element_t pop(stk_t *s) {
12
13   if (!s || !(s -> top)) return NULL;
14
15   cell_t *p = s -> top;
16   element_t e = s -> top -> element;
17   s -> top = s -> top -> next;
18   s -> size--;
19   free(p);
20   return (e);
21 }
```

TAD: Pilha em vetor

```
1 typedef int * element_t;
2 typedef struct stk stk_t;
3
4 struct stk {
5   unsigned int
6     top,
7     maxsize;
8   element_t *svector;
9 };
```

Funções adicionais:

```
1 bool isempty(stk_t *s) {
2   return (s -> top == 0);
3 }
4 bool isfull(stk_t *s)  {
5   return ( s -> top == s -> maxsize);
6 }
```

Instanciando uma pilha (em vetor)

```
1  stk_t *newStack(unsigned int n) {
2
3    stk_t *p =  malloc (sizeof (stk_t));
4    if (!p) return NULL;
5
6    p->svector = malloc (n* sizeof (element_t));
7    if (!p -> svector) return NULL;
8
9    p -> top = 0;
10   p -> maxsize = n;
11   return p;
12 }
```

Inserção e remoção

```
1  bool push(element_t e, stk_t *s) {
2
3    if (isfull(s)) return FALSE;
4
5    s -> svector[s -> top++] = e;
6
7    return TRUE;
8  }
9
10 element_t pop(stk_t *s) {
11
12   if (isempty(s)) return NULL;
13
14   element_t e = s -> svector[--(s -> top)];
15
16   return (e);
17 }
```