Lista 9 - MAC0122 Princípios de Desenvolvimento de Algoritmos - POLI

Prof. Ronaldo Fumio Hashimoto

1 Exercícios

Os exercícios de 18 a 25 correspondem, respectivamente, aos exercícios de 4.18 a 4.25 do livro de Robert Sedgewick [1].

18. Abaixo são exibidos um programa (Programa 4.4 do livro) e uma figura (Figura 4.1 do livro). Dê os valores de `s[0] ..., s[4]` depois da execução das operações ilustradas na figura, usando o programa.

    Programa 4.4

```
#include <stdlib.h>
#include "Item.h"
#include "STACK.h"
static Item *s;
static int N;
void STACKinit(int maxN)
   { s = malloc(maxN*sizeof(Item)); N = 0; }
int STACKempty()
   { return N == 0; }
void STACKpush(Item item;)
   { s[N++] = item; }
Item STACKpop()
   { return s[--N]; }
```

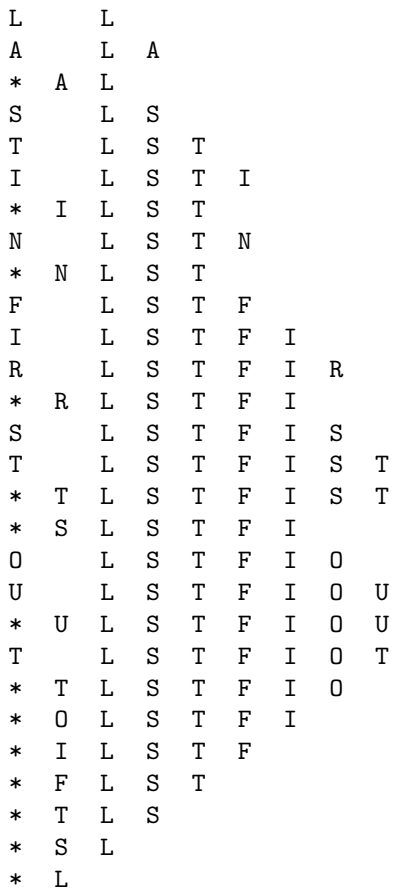

Figura 1: Figura adpatada de [2].

Exemplo de Pilha *PushDown* (fila LIFO)

Essa lista exibe o resultado da sequência de operações na coluna da esquerda (de cima para baixo), onde uma letra denota um empilhamento e um asterisco um desempilhamento. Cada linha exibe: a operação, a letra que foi desempilhada para a operação de desempilhamento, e o conteúdo da pilha depois da operação, na ordem do menos recente inserido para o mais recente inserido da esquerda para direita.

19. Suponha que você troque a interface da pilha *pushdown*, removendo o teste de pilha vazia pela contagem de elementos, que deve retornar o número de itens que estão na estrutura de dados. Forneça a implementação para a contagem de elementos utilizando a representação por vetor (*array*) do Programa do exercicio 18 (Programa 4.4 do livro) e a representação da lista ligada do programa abaixo (Programa 4.5 do livro).

Programa 4.5

```
#include <stdlib.h>
#include "Item.h"
typedef struct STACKnode* link;
struct STACKnode { Item item, link next; };
static link head;
link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
void STACKinit(int maxN)
   { head = NULL; }
int STACKempty()
   { return head == NULL; }
STACKpush(Item item)
   { head = NEW(item, head); }
Item STACKpop()
   { Item item = head->item;
     link t = head->next;
     free(head); head = t;
     return item;
   }
```

20. Modifique a implementação da pilha *pushdown* que utiliza vetor (programa do exercicio 18) para chamar a função `STACKerror` se o cliente tentar desempilhar em uma pilha vazia ou empilhar em uma pilha cheia.

21. Modifique a implementação da pilha que utiliza a lista ligada (programa do exercicio 19) para chamar a função `STACKerror` se o cliente tentar desempilhar uma pilha vazia ou se não existir mais espaço para um `malloc` ao executar um empilhamento.

22. Modifique a implementação da pilha que utiliza lista ligada (programa do exercicio 19) para utilizar a implementação da lista que utiliza índices de vetor, tal com na figura abaixo (Figura 3.6 do livro)

	0	1	2	3	4	5	6	7	8
item	1	2	3	4	5	6	7	8	9
next	1	2	3	4	5	6	7	8	0
5	1	2	3	4	5	6	7	8	9
	1	2	3	5	5	6	7	8	0
1	1	2	3	4	5	6	7	8	9
	1	2	3	5	5	6	7	8	1
7	1	2	3	4	5	6	7	8	9
	1	2	3	5	5	7	7	8	1
4	1	2	3	4	5	6	7	8	9
	1	2	5	5	5	7	7	8	1
3	1	2	3	4	5	6	7	8	9
	1	5	5	5	5	7	7	8	1
6	1	2	3	4	5	6	7	8	9
	1	7	5	5	5	7	7	8	1
9	1	2	3	4	5	6	7	8	9
	1	7	5	5	5	7	7	1	1
2	1	2	3	4	5	6	7	8	9
	1	7	5	5	5	7	7	7	1

Figura 2: Figura adaptada de [3].

Representação de lista ligada como vetor.

Essa sequência exibe a lista ligada para o problema de Josephus, implementada como índices de vetores em vez de ponteiros. O índice do item seguinte ao item com índice 0 na lista é `next[0]`, e assim por diante. Inicialmente (as três linhas do topo), o item da pessoa i tem índice $i-1$, e nós formamos uma lista circular estabelecendo `next[i]` para $i+1$ para i de 0 a 8 e `next[8]` para 0. Para simular o processo de remoção, nós trocamos os *links* (entrada de vetor `next`) mas não movemos os itens. Cada par de linhas exibe o resultado de andar pela lista quatro vezes através de `x = next[x]`. Então, apaga-se o quinto item (exibida ao lado esquerdo) ao modificarmos `next[x]` para `next[next[x]]`.

23. Escreva uma implementação de pilha *pushback* utilizando lista ligada que mantém os itens na lista na ordem do menos recente para o mais recente inserido. Você precisará de uma lista duplamente ligada.

24. Desenvolva um ADT (*Abstract Data Type*) que forneça duas pilhas *push-back* diferentes. Use a implementação que utiliza vetor. Mantenha uma pilha no começo do vetor e outra no final. (se o programa cliente mantém uma pilha que cresce enquanto a outra diminui, essa implementação usa menos espaço que outra alternativas.)

25. Implemente uma função que calcula expressões infixas para inteiros que incluam os programas abaixo (Programa 4.2 e Programa 4.3 do livro), usando sua ADT do exercicio 24.

    Programa 4.2

```
#include <stdio.h>
#include <string.h>
#include "Item.h"
#include "STACK.h"
main(int argc, char *argv[])
  { char *a = argv[1]; int i, N = strlen(a);
    STACKinit(N);
    for (i = 0; i < N; i++)
      {
         if (a[i] == '+')
            STACKpush(STACKpop() + STACKpop());
         if (a[i] == '*')
            STACKpush(STACKpop() * STACKpop());
         if ((a[i] >= '0') && (a[i] <= '9'))
            STACKpush(0);
         while ((a[i] >= '0') && (a[i] <= '9'))
            STACKpush(10*STACKpop() + (a[i++]-'0'));
      }
    printf("%d \n", STACKpop());
}
```

Programa 4.3

```
#include <stdio.h>
#include <string.h>
#include "Item.h"
#include "STACK.h"
main(int argc, char *argv[])
  { char *a = argv[1]; int i, N = strlen(a);
    STACKinit(N);
    for (i = 0; i < N; i++)
      {
         if (a[i] == ')')
            printf("%c", STACKpop());
         if ((a[i] == '+') || (a[i] <= '*'))
            STACKpush(a[i]);
         if ((a[i] >= '0' && (a[i] <= '9'))
            printf("%c", a[i]);
      }
    printf("\n");
  }
```

Referências

[1] R. Sedgewick, "Algorithms in c—third edition," pp. 148–149, 1998.

[2] R. Sedgewick, "Algorithms in c—third edition," p. 136, 1998.

[3] R. Sedgewick, "Algorithms in c—third edition," p. 96, 1998.