Práctica 5: Pila y cola

Estructuras de Datos y Algoritmos I

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno revisará las definiciones, características, procedimientos y ejemplos de las estructuras lineales Pila y Cola para comprender sus estructuras y funcionamiento a fin de ser capaz de implementarlas.

2. Material

Se asume que el alumno cuenta con un una computadora con arquitectura Intel x86 o compatible, sistema operativo Linux/Unix, y compilador `gcc` instalado.

3. Instrucciones

1. Lea detenidamente los antecedentes teóricos sobre la construcción de estructuras de datos tipo *FIFO* y tipo *LIFO* en la Sección 4.
2. Realice cada una de las actividades detalladas en la Sección 5 apoyándose en el código de ejemplo del Apéndice A y responda las preguntas de las mismas.

4. Antecedentes[1]

Las pilas y las colas son conjuntos dinámicos en los que el elemento eliminado del conjunto por la operación DELETE está preespecificado. En una pila, el elemento eliminado del conjunto es el que se ha insertado más recientemente: la pila implementa una política de *último en entrar, primero en salir* o LIFO (***L**ast **I**n, **F**irst **Out***). De manera similar, en una cola, el elemento eliminado es siempre el que ha estado en el conjunto durante más tiempo: la cola implementa una política de *primero en entrar, primero en salir* o (***F**irst **I**n, **F**irst **Out***). Hay varias formas eficientes de implementar pilas y colas en una computadora.

4.1. Pilas (Stacks)

La operación INSERT en una pila a menudo se llama PUSH, y la operación DELETE, que no necesita un argumentos, a menudo se llama POP. Estos nombres son alusiones a pilas físicas, como las pilas de platos con resorte que se usan en las cafeterías. El orden en el que se extraen los platos de la pila es el orden inverso al que se empujaron hacia la pila, ya que solo se puede acceder a la placa superior.

Como muestra la Figura 1, podemos implementar una pila de máximo n elementos utilizando un arreglo. El arreglo tiene un atributo $S.top$ que indexa al último elemento insertado. La pila consta de los elementos $S[1 \ldots S.top]$, donde $S[1]$ es el elemento en la parte inferior de la pila y $S[S.top]$ es el elemento en la parte superior.

Cuando $S.top = 0$ la pila no contiene elementos y está vacía. Se puede probar si la pila está vacía mediante la operación de consulta STACK-EMPTY. Si se intenta sacar un elemento de una pila vacía con la operación POP, se dice que la pila no fluye (***underflow***), lo que normalmente es un error. Por otro lado, si $S.top > n$, la pila se desborda (en la implementación de pseudocódigo, no nos preocupamos por el desbordamiento de la pila).

[1] *Esta sección, en su totalidad, es una traducción del capítulo 10 y sección 10.1 del libro «Introduction to Algorithms» 3ª edición de Thomas Cormen.*

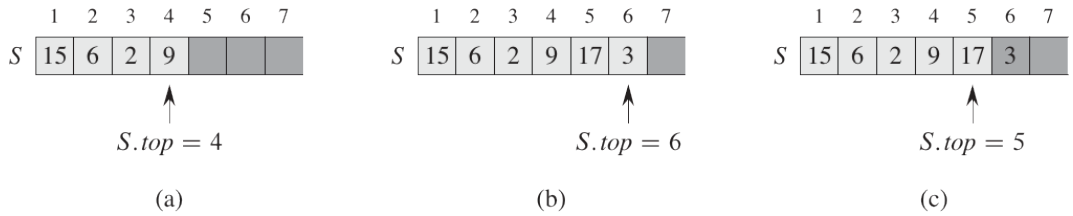

Figura 1: Implementación una pila S usando arreglos. Los elementos de pila aparecen sombreados. (a) La pila S tiene 4 elementos. El elemento superior es el 9. (b) La pila S después de las llamadas PUSH $(S, 17)$ y PUSH $(S, 3)$. (c) La pila S después de que la llamada POP (S) haya devuelto el elemento 3, que es el último ingresado. Aunque el elemento 3 todavía aparece en el arreglo, ya no está en la pila; la parte superior es el elemento 17.

Podemos implementar cada una de las operaciones de la pila con solo unas pocas líneas de código:

Algoritmo 1 Verificar si la pila está vacía

```
1: procedure STACK-EMPTY(S)
2:     if S.top == 0 then
3:         return TRUE
4:     end if
5:     return FALSE
6: end procedure
```

Algoritmo 2 Insertar un elemento en la pila

```
1: procedure STACK-PUSH(S, x)
2:     S.top = S.top + 1
3:     S[S.top] = x
4: end procedure
```

Algoritmo 3 Sacar (eliminar) un elemento de la pila

```
1: procedure STACK-POP(S)
2:     if STACK-EMPTY(S) then
3:         error «underflow»
4:     else
5:         S.top = S.top − 1
6:         return S[S.top + 1]
7:     end if
8: end procedure
```

4.2. Colas (Queues)

Llamamos a la operación INSERT en una cola ENQUEUE, y llamamos a la operación DELETE, DEQUEUE. Al igual que la operación POP de una pila, DEQUEUE no recibe ningún argumento como parámetro. La propiedad FIFO de una cola hace que funcione como una fila de clientes esperando pagarle a un cajero. La cola tiene cabeza y cola. Cuando un elemento se coloca en la cola, ocupa su lugar al final de la cola, al igual que un cliente recién llegado ocupa un lugar al final de la fila. El elemento eliminadp de la cola es siempre el que está al principio de la cola, como el cliente al principio de la fila que ha esperado más tiempo.

La figura Figura 2 muestra una forma de implementar una cola de a lo sumo $n - 1$ elementos usando un arreglo $Q[1 \ldots n]$. La cola tiene un atributo $Q.head$ que indexa o apunta a su cabeza. El atributo $Q.tail$ indexa la siguiente ubicación en la que se insertará un elemento recién llegado a la cola. Los elementos de la cola residen en

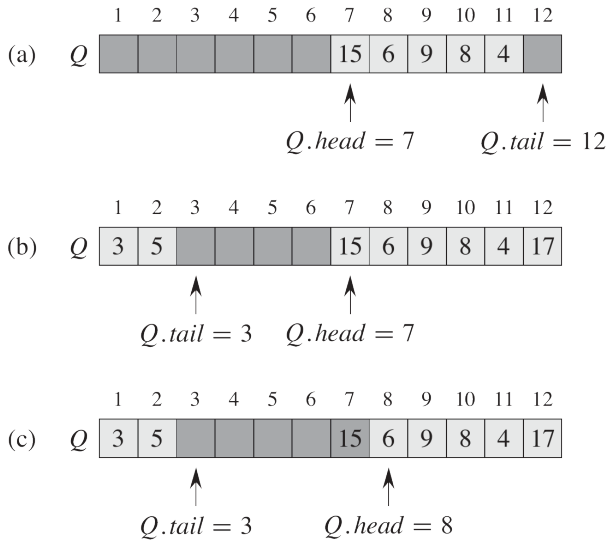

Figura 2

las locaciones $Q.head, Q.head + 1, \ldots, Q.tail - 1$, donde son «envueltos», en el sentido de que la locación 1 sigue inmediatamente a la locación n en un orden circular. Cuando $Q.head = Q.tail$, la cola está vacía. Inicialmente, tenemos $Q.head = Q.tail = 1$. Si intentamos sacar un elemento de una cola vacía, la cola no fluye (***underflow***).

Cuando $Q.head = Q.tail + 1$, la cola está llena, y si intentamos insertar un elemento en la cola, ésta se desborda (***overflow***). En los procedimientos ENQUEUE y DEQUEUE, se han omitido la comprobación de errores para *overflow* y *underflow*. En el pseudocódigo se asume que $n = Q.length$.

Algoritmo 4 Insertar un elemento en la cola

```
1: procedure ENQUEUE(Q, x)
2:     Q[Q.tail] = x
3:     if Q.tail == Q.length then
4:         Q.tail = 1
5:     else
6:         Q.tail = Q.tail + 1
7:     end if
8: end procedure
```

Algoritmo 5 Sacar (eliminar) un elemento de la cola

```
1: procedure DEQUEUE(Q)
2:     x = Q[Q.head]
3:     if Q.head == Q.length then
4:         Q.head = 1
5:     else
6:         Q.head = Q.head + 1
7:     end if
8:     return x
9: end procedure
```

5. Desarrollo de la práctica

Lea cuidadosamente cada una de las actividades propuestas antes de realizar el programa o las modificaciones indicadas.

5.1. Actividad 1: Pila

El programa de la Apéndice A.2 implementa parcialmente una estructura de datos tipo LIFO.

Implemente las funciones PUSH (Q, x) y POP (Q), compile, ejecute y pruebe su código ingresando diez números, por ejemplo:

```
1 3 5 7 11 13 17 23 29 42 69
```

¿En qué orden se imprimen los números? Explique [1 puntos]: _____

¿Cómo implementó las funciones PUSH (Q, x) y POP (Q, x)? Anote su código a continuación [2 puntos]:

5.2. Actividad 2: Cola

El programa de la Apéndice A.3 implementa parcialmente una estructura de datos tipo FIFO.

Implemente las funciones ENQUEUE (Q, x) y DEQUEUE (Q, x), compile, ejecute y pruebe su código ingresando diez números, por ejemplo:

```
1 3 5 7 11 13 17 23 29 42 69
```

¿En qué orden se imprimen los números? Explique [1 puntos]: _____

¿Cómo implementó las funciones ENQUEUE (Q, x) y DEQUEUE (Q)? Anote su código a continuación [2 puntos]:

5.3. Actividad 3: Apilando cadenas

Tome como base la implementación desarrollada en la Actividad 1 y modifique el código para que la pila acepte cadenas en lugar de enteros. ¿Qué cambios se realizaron? Explique y anote su código a continuación [2 puntos]:

5.4. Actividad 4: Encolando cadenas

Tome como base la implementación desarrollada en la Actividad 2 y modifique el código para que la cola acepte cadenas en lugar de enteros. ¿Qué cambios se realizaron? Explique y anote su código a continuación [2 puntos]:

A. Código de ejemplo

A.1. Archivo **src/Makefile**

src/Makefile

```
CC       = gcc
CFLAGS   = -Wall -O3 -std=c99 -pedantic
SILENT   = @
PROGRAMS = stack queue #stacks queues
.PHONY: all clean $(PROGRAMS)

all: $(PROGRAMS)

$(PROGRAMS): %: %.c
  $(SILENT) $(CC) $(CFLAGS) -o $@ $<

clean:
  rm -f *.o
```

A.2. Archivo `src/stack.c`

src/stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#define STACK_SIZE 20
struct {
  int* array;
  size_t top;
  size_t asize;
  size_t count;
} typedef stack;

stack* stack_init(void);

int stack_empty(stack *s);
void stack_push(stack *s, int value);
int stack_pop(stack *s);
void stack_print(stack *s);

int main(void){
  char opc = 0;
  int value;
  stack *s = stack_init();

  printf("STACK!!!\n\n");
  while(1){
    printf("S tiene %lu elementos\n", s->count);
    printf("  u: pUsh\n");
    printf("  o: pOp\n");
    printf("  v: Vaciar\n");
    printf("  p: imPrimir\n");
    printf("  q: salir\n\n");
    printf("Opcion: ");
    if(!scanf(" %c", &opc)) continue;
    switch(opc){
      case 'q': case 'Q':
        return 0;
      case 'u': case 'U':
        printf("Valor a ingresar: ");
        if(scanf("%d", &value)){
          printf("push(s, %d)\n\n", value);
          stack_push(s, value);
        }
        break;
      case 'o': case 'O':
        printf("%d <- pop(s)\n\n", stack_pop(s)); break;
      case 'p': case 'P':
        stack_print(s); break;
      case 'v': case 'V':
        printf("S:");
        while(s->top) printf(" %d", stack_pop(s));
        printf("\n");
        break;
    }
  }
  return 0;
}

int stack_empty(stack *s){
  return !s->top;
}

stack* stack_init(void){
  stack *s = (stack*)calloc(1, sizeof(stack));
  s->array = (int*)calloc(STACK_SIZE, sizeof(int));
  s->asize = STACK_SIZE;
  s->count = 0;
  return s;
}

void stack_push(stack *s, int value){
  // Código del estudiante
}

int stack_pop(stack *s){
  // Código del estudiante
  return 0;
}

void stack_print(stack *s){
  printf("S:");
  for(size_t i = 0; i < s->top; ++i)
    printf(" %d ->", s->array[i]);
  printf("<-TOP\n\n");
}
```

A.3. Archivo `src/queue.c`

src/queue.c

```c
#include <stdio.h>
#include <stdlib.h>
#define Q_SIZE 10
struct {
  int* array;
  size_t head;
  size_t tail;
  size_t asize;
  size_t count;
} typedef queue;

queue* queue_init(void);

void enqueue(queue *q, int value);
int dequeue(queue *q);
void queue_print(queue *q);

int main(void){
  char opc = 0;
  int value;
  queue *q = queue_init();

  printf("QUEUE!!!\n\n");
  while(1){
    printf("Q tiene %lu elementos\n", q
    ->count);
    printf("  e: encolar\n");
    printf("  d: desencolar\n");
    printf("  v: Vaciar\n");
    printf("  p: imPrimir\n");
    printf("  q: salir\n\n");
    printf("Opcion: ");
    if(!scanf(" %c", &opc)) continue;
    switch(opc){
      case 'q': case 'Q':
        return 0;
      case 'e': case 'E':
        printf("Valor a ingresar: ");
        if(scanf("%d", &value)){
          printf("enqueue(q, %d)\n\n",
    value);
          enqueue(q, value);
        }
        break;
      case 'd': case 'D':
        printf("%d <- dequeue(q)\n\n",
    dequeue(q)); break;
      case 'p': case 'P':
        queue_print(q); break;
      case 'v': case 'V':
        printf("Q:");
        while(q->count) printf(" %d",
    dequeue(q));
        printf("\n");
        break;
    }
  }
  return 0;
}

int queue_empty(queue *q){
  return !q->count == 0;
}

queue* queue_init(void){
  queue *q = (queue*)calloc(1, sizeof(
    queue));
  q->array = (int*)calloc(Q_SIZE,
    sizeof(int));
  q->asize = Q_SIZE;
  q->count = 0;
  return q;
}

void enqueue(queue *q, int value){
  // Código del estudiante
}

int dequeue(queue *q){
  // Código del estudiante
  return 0;
}

void queue_print(queue *q){
  printf("Q:\n[Tail] ->");
  size_t i = q->head;
  size_t j = 0;
  while(j < q->count){
    printf(" %d ->", q->array[i++]);
    i %=q->asize;
    ++j;
  }
  printf("<- [Head]\n\n");
}
```

B. Reporte Escrito

El reporte de la práctica deberá ser entregada en un archivo en formato `PDF` siguiendo las siguientes especificaciones:

- La primera página del documento deberá ser la carátula oficial para prácticas de laboratorio disponible en `lcp02.fi-b.unam.mx/`
- El nombre del documento `PDF` deberá ser `nn-XXXX-L05.pdf`, donde:
  - `nn` es el número de lista del alumno a dos dígitos forzosos (ej. 01, 02, etc.).
  - `XXXX` corresponden a las dos primeras letras del apellido paterno seguidas de la primera letra del apellido materno y la primera letra del nombre, en mayúsculas y evitando cacofonías; es decir, los cuatro primeros caracteres de su RFC o CURP.
- El reporte consiste en un documento de redacción propia donde el alumno explica de forma concisa y a detalle las actividades realizadas en la práctica, y reportando los resultados obtenidos.
- La longitud del reporte no deberá exceder las 3 páginas, sin contar la carátula.
- El reporte deberá seguir todos los lineamientos para documentos escritos establecidos al inicio del curso.
- Todas las referencias deberán estar debidamente citadas.

IMPORTANTE: No se aceptan archivos en otros formatos ni con nombres distintos a los especificados.