Algoritmos y estructuras de Datos
P2PU

Listas, Pilas, Colas y Punteros

Semana 3

Dictado por Marco González Núñez
14 de Febrero de 2011

Algoritmos y estructuras de Datos
P2PU

Estructuras de datos simples

Hoy hablaremos de lo que sin duda me motivó a crear este curso que son los punteros, sin embargo debemos iniciar nuestro estudio describiendo estructuras que pueden implementarse fácilmente en pilas (*stacks)*, colas y la estructura lineal básica vinculada mediante punteros simples y listas doblemente enlazadas.

Algoritmos y estructuras de Datos
P2PU

Pilas y Colas

Introducción: Las **pilas** y las **colas** son conjuntos dinámicos de datos relacionados entre sí, la idea es insertar y eliminar cada una de las estructuras que lo componen:

Pila: *Es el elemento insertado más recientemente (política LIFO, Last In First Out), o sea el último que entra es el primero que sale. Se puede entender como una pila de platos, en el cual la única opción que tenemos es sacar el que está más arriba, que es el último que ingresamos.*

Cola: *Es el elemento que ha permanecido en el conjunto por más tiempo política FIFO (First In First Out), o sea el primero que entra es el primero que sale. Se puede entender como una fila en un banco, donde el primero que llega es el primero en ser atendido.*

Algoritmos y estructuras de Datos
P2PU

Las únicas operaciones permitidas por estas estructuras son:

Insert y Delete

Hay varias formas eficientes de implementar *pilas* y *colas*, p.ej., aquí usaremos un arreglo simple.

Pilas, en este caso:

Insert se llama Push

Delete se llama Pop y no toma elementos como argumentos.

El orden en el cual los elementos salen de la pila es el inverso del orden en el cual son puestos allí, sólo el elemento de más arriba está accesible.

Algoritmos y estructuras de Datos
P2PU

Implementamos una pila, con a lo más *n* elementos, en un arreglo *S*[0] … *S*[*n*–1] : El arreglo tiene un atributo *top*[*S*] — el índice del elemento insertado más recientemente.

La pila consiste en los elementos *S*[0] … *S*[*top*[*S*]] :

S[0] es el elemento en el fondo, y *S*[*top*[*S*]] es el elemento en el tope. Si *top*[*S*] = –1, entonces la pila está vacía — la operación Empty? (*S*) verifica esta situación.

```
Pop(S) :
        if (Empty?(S))
                error "underflow"
        else    top[S] = top[S]–1
                return S[top[S]+1]
```

```
Push(S, x) :
        if (top[S] == n–1)
                error "overflow"
        else    top[S] = top[S]+1
                        S[top[S]] = x
```

```
Empty?(S) :
return top[S] == –1
```

Cada una de estas operaciones toma tiempo ***O***(1).

Algoritmos y estructuras de Datos
P2PU

Colas, en este caso:

Insert se llama EnQ, y

Delete se llama DeQ y no toma elementos como argumentos.

Una cola tiene un *primer* y un *último* elemento:

- El elemento que sale de una cola es el primero.
- Un elemento ingresa a una cola después del último.

Implementamos una cola, con a lo más n–1 elementos, en un arreglo $Q[0] \ldots Q[n-1]$:

La cola tiene dos atributos:

head[*Q*] — posición del primer elemento

tail[*Q*] — posición que sigue al último elemento

Algoritmos y estructuras de Datos
P2PU

Los elementos en la cola están en las posiciones *head*[*Q*], *head*[*Q*] +1, **...**, *tail*[*Q*]–1, circularmente — la posición 0 sigue a la posición *n*–1.

Los valores de los atributos reflejan el estado de la cola:

la cola está vacía cuando *head*[*Q*] = *tail*[*Q*]

la cola está llena cuando *head*[*Q*] = *tail*[*Q*]+1

```
EnQ(Q, x) :
        Q[tail[Q]] = x
        if (tail[Q] == n–1)
                tail[Q] = 0
        else tail[Q] = tail[Q]+1
```

```
DeQ(Q) :
        x = Q[head[Q]]
        if (head[Q] == n–1)
                head[Q] = 0
        else head[Q] = head[Q]+1
        return x
```

Cada una de estas operaciones toma tiempo ***O***(1).

Algoritmos y estructuras de Datos
P2PU

Introducción a los punteros :)

Los punteros como el nombre lo indica apuntan a *direcciones de memorias* dentro de algun dispositivo de almacenaje volatil o permanete, como una memoria RAM o un disco duro.

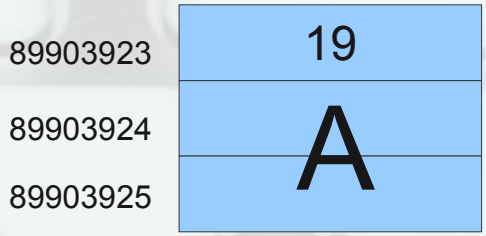
Los punteros nos sirven para relacionar información almacenadas en distintas direcciones de memoria no necesariamente continuas.

Primero que todo definiremos como son las ranuras de memoria de dispositivos que usamos diariamente.

Cuando vamos a un supermercado podemos guardar nuestras pertenencias en ***lokers*** o ***casilleros*** enumerados de forma única y todos del mismo tamaño, lo mismo pasa con las ranuras de memoria, todas tienen una dirección única con algun dato o fragmento de éste.

Algoritmos y estructuras de Datos
P2PU

Por ejemplo los números (int) ocupan 1 casilla y los caracteres (char) 2 casillas, entonces podemos indicar donde están con el siguiente ejemplo:



Por lo tanto decimos que el número 19 está en la dirección *89903923,* y el caracter A en la dirección *89903924,* y como se sabe que un caracter ocupa 2 casillas, entonces tomará ésta y la siguiente, obteniendo asi la información buscada.

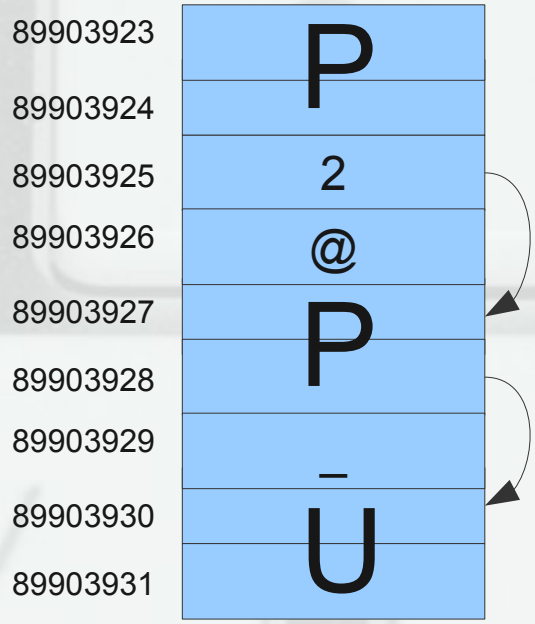
Antes de seguir: ¿Han desfragmentado un disco?, si lo han echo han notado que el espacio libre no esta continuo, es un muy buen ejemplo gráfico de lo que quiero explicar.

Algoritmos y estructuras de Datos
P2PU

¿Qué pasa ahora cuando queremos guardar una palabra?

El SO tratará de guardarla en alguna posición donde se almacene continuamente, en el caso de que no exista memoria continua (en la práctica casi imposible para una palabra), buscará guardarla en pequeños trozos de memoria, con sus punteros correspondientes.

Ejemplo del segundo caso (guardaremos la palabra P2PU):



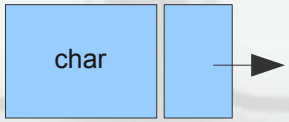
Por lo tanto estamos diciendo que la palabra P2PU está en la casilla de memoria MEMORIA[89903923] e internamente éste tiene los punteros que se ven gráficamente.

Algoritmos y estructuras de Datos
P2PU

Ahora que ya conocemos que son y como funcionan los punteros veremos estructuras de datos un poco más complejas sin preocuparnos mucho de cuanta memoria ocupa, si no que del puntero que lleva a la siguiente estructura.

Una estructura la visualizaremos como una caja, ésta contiene información de cualquier tipo y un puntero que “apunte” a otra estructura de iguales caracteristicas:

Por ejemplo nuestra estructura será la siguiente:



Que esta compuesta por un char y un puntero.

Algoritmos y estructuras de Datos
P2PU

Recordemos que estas estructuras pueden apuntar solo a otras similares, en un caso de ejemplo haremos una lista:

Que está compuesta por un char y un puntero, apuntando a otra estructura de iguales características (un char y un puntero), notese que estas estructuras las creamos nosotros, por ende puede tener tanta información y punteros como se desee (obviamente posible). Y la última la apuntamos a null (tail en el código), para saber que es la última por ende la próxima en salir, recordemos que es una lista (FIFO).

Algoritmos y estructuras de Datos
P2PU

Tarea: Cree un diagrama de flujos en el cual se puedan hacer las siguientes funciones para una lista FIFO (considerando la estructura de ejemplo):

- Añadir una nueva estructura (revisar si la lista esta vacía).
  Pasos: Crear la estructura, añadir un dato, crear un puntero, y dirigirlo, recibe un dato *char*.

- Buscar si un dato existe dentro de una lista, recibe el dato *char,* y devuelve un *true* o un *false*.

- Eliminar una estructura, recibe un dato *char.*

Algoritmos y estructuras de Datos
P2PU

La tarea debe ser enviada a mi correo electrónico (markogonzalez84@gmail.com), a más tardar a las 23:59 del día Viernes 18 de Febrero de 2011, hora de Chile, en formato pdf.

Debe tener una portada con el nombre y nick de cada uno(P2PU), letra Arial 16, para títulos y 12 para texto, interlineado 1.5 y justificado.

Tarea Opcional: Averigue como podemos crear una estructura, o nodo en su lenguaje favorito, y como podemos declarar un puntero.

Algoritmos y estructuras de Datos
P2PU

Prguntas para debatir en los foros:

1.- ¿Como se pueden relacionar las pilas y colas con los algoritmos de búsqueda?

2.- ¿En qué tipos de problemas están presentes pilas y colas?

3.- ¿Cuál es la diferencia entre formatear y formatear rápido un disco duro?

Frase de la semana: ”La ventaja de una lista contra un arreglo es que no está limitado por su tamaño”