Arreglos y listas

Alberto Benavides
11 de marzo de 2020

Los arreglos son estructuras de datos capaces de almacenar n elementos. Si llamamos t al tamaño en bits requerido por cada uno de esos elementos, entonces podemos decir que los arreglos reservan nt bits de memoria *secuencial* donde podrá alojarse dicha cantidad de elementos.

Un ejemplo de arreglos para los lenguajes programación de la familia `C` es el arreglo de caracteres, que da como resultado una cadena de texto de longitud limitada. Para fines ilustrativos, supongamos que se desea almacenar el primer nombre de contacto de una persona en un arreglo. En dicho caso, una longitud de veinte caracteres parecería suficiente para este fin, sin embargo, es poco probable que la longitud de algún nombre supere los diez caracteres [4].

```
1 int main()
2 {
3     char nombre[20];
4 }
```

Código 1: Arreglo de veinte caracteres en `C`.

Así como en el caso anterior, es recomendable utilizar arreglos de tamaño suficientemente superior al esperado para evitar encontrar excepciones por exceder el tamaño definido del arreglo. Por esto, el uso de arreglos suele estar asociado al problema del desperdicio de memoria [3].

Por su parte, las listas ligadas son estructuras de datos que dependen de punteros para hacer referencias a nuevos espacios de memoria que se designan y reservan conforme haya necesidad de hacerlo. Este fin se logra gracias a los punteros, que son tipos de datos capaces de almacenar direcciones de memoria. Así, un elemento puede almacenar las direcciones de memoria del elemento siguiente o anterior. En los lenguajes de programación de la familia `C` esto se logra mediante estructuras (`struct`).

```
1 struct nodo{
2     int valor;
3     struct nodo *sig;
4 };
5
6 nodo *inicio, *fin;
7
8 void CrearNodo(int t){
9     nodo *temp = new nodo;
10    temp->valor = t;
11    temp->sig = NULL;
12    if(inicio == NULL){
13        inicio = temp;
14        fin = temp;
15    } else {
16        fin->sig = temp;
17        fin = temp;
18    }
19 }
20
21 void RecorrerNodos(){
22    nodo *temp = new nodo;
23    temp = inicio;
24    while(temp != NULL)
25    {
26      temp = temp->sig;
27    }
28 }
```

Código 2: Funciones para creación y recorrido de listas.

Con algunas pruebas computacionales se puede explorar el comportamiento de estas estructuras de datos. Por ejemplo, la asignación de valores aleatorios asignados de manera secuencial a un arreglo de mil elementos en un millón de repeticiones da un promedio de 11.8698ms, mientras que esta asignación con las mismas repeticiones toma en promedio 67.0838ms en una lista ligada, o sea 5.6516 veces más lento. El recorrido secuencial los elementos de este arreglo y lista ligada toma, respectivamente y en promedio, 2.4357ms y 3.24435ms. Estas diferencias en tiempos de ejecución se deben, en el primer caso, a que existen condiciones que determinan que la lista haya sido inicializada. En el segundo caso, el hecho de que el arreglo se recorriera ligeramente más rápido que la lista ligada, puede deberse a que las direcciones de memoria de los arreglos son secuenciales.

Si para cada ejecución $p \in \{\text{arreglo}, \text{lista}\}$ se ponderara el tiempo de ejecución τ y tamaño en memoria utilizada μ, se podría establecer una función $f(p)$ para comparar estos procedimientos tal que

$$f(p) = \frac{\tau_p}{\text{máx}(\tau_{\text{arreglo}}, \tau_{\text{lista}})} + \frac{\mu_p}{\text{máx}(\mu_{\text{arreglo}}, \mu_{\text{lista}})}.$$

Ahora se podría determinar que la estructura de datos más conveniente viene dada por la mín$(f(p))$.

De todas formas, estas operaciones son del tipo $\mathcal{O}(n)$ puesto que se han de agregar o recorrer los n elementos determinados. La inserción de elementos no secuencial (en un

lugar distinto al de la última posición utilizada), por otro lado, representa un problema para los arreglos debido a que insertar un elemento en la posición $0 \leq i < n - 1$ del arreglo implica desplazar $n - i$ elementos hacia la posición siguiente, perdiendo, además, el último elemento. En el peor de los casos (si se desea insertar en el primer elemento) se trata de un procedimiento de complejidad $\mathcal{O}(n - i)$.

```
1 int a[n];
2 void Insertar(int i, int entero){
3     for(int j = n - 1; j > i; j--){
4         a[j] = a[j - 1];
5     }
6     a[i] = entero;
7 }
```

Código 3: Función para insertar un entero en la posición i de un arreglo de n elementos.

Las listas ligadas requieren i operaciones para lograr esta misma inserción. Pero podrían reducirse estas operaciones si se utilizan listas doblemente ligadas y se almacena la longitud de la lista en una variable entera, esto con el fin de saber si la posición donde se desea insertar está antes o después de la mitad de la longitud y así iterar desde el extremo más cercano, lo cual, en el peor de los casos es de una complejidad $\mathcal{O}(n/2)$.

La mejor opción de entre estas dos estructuras de datos para crear un conjunto sería una lista doblemente ligada porque así como la inserción requiere menos operaciones que en los arreglos, las operaciones para la búsqueda de elementos existentes (un conjunto no tiene elementos repetidos) tienen las mismas complejidades. Encima, la cantidad de elementos de un arreglo está definida, lo que limita el uso de esta estructura a conjuntos de cardinalidad conocida, lo cual no sucede con las listas.

Así como estas preferencias, existen otras registradas que suelen favorecer a las listas dada su versatilidad. Sin embargo, los arreglos suelen ser mucho más eficientes en tiempos de ejecución que las listas. Entre los casos que los arreglos son preferidos frente a las listas se encuentra la manipulación de imágenes de tamaño definido ya que la información de sus pixeles puede vectorizarse en un arreglo de tamaño conocido.

En conclusión, se puede decir que es preferible el uso de arreglos cuando se tiene un tamaño fijo o poco variable de elementos y se realizarán operaciones secuenciales, mientras se opta por listas ligadas en caso distinto.

Referencias

[1] Kamal Choudhary. Búsqueda en google.com de *static array vs linked list*. `https://www.google.com/search?q=static+array+vs+linked+list&oq=static+array+vs+linked+list&aqs=chrome..69i57.4785j0j1&sourceid=chrome&ie=UTF-8`, .

[2] Kamal Choudhary. A comprehensive guide to singly linked list using C++. `https://www.codementor.io/@codementorteam/a-comprehensive-guide-to-implementation-of-singly-linked-list-using-c_plus_plus-ondlm5azr`, .

[3] Hermann Krohn. Linked lists vs. arrays. https://towardsdatascience.com/linked-lists-vs-arrays-78746f983267.

[4] Quora. What is the average length of first names in the United States? https://www.quora.com/What-is-the-average-length-of-first-names-in-the-United-States.