

LOOK UP

Semillero de Programación

Santiago Vanegas Gil

Universidad EAFIT

30 de agosto del 2014

Contenido

- 1 Introducción al Semillero de Programación
 - Descripción
- 2 Estructuras de datos lineales
 - Arreglos estáticos
 - Arreglos dinámicos
 - Pilas
 - Colas
- 3 Estructuras de datos no lineales
 - Arbol binario de búsqueda
 - Mapa
 - Set
 - Cola de prioridad

Contenido

- 1 Introducción al Semillero de Programación
 - Descripción

¿Qué hacemos?

Aprendemos varios temas importantes y entrenamos para las maratones de programación de ACIS/REDIS y ACM-ICPC, donde se busca resolver problemas de Ciencias de la Computación conocidos lo más rápido posible.

¿Cómo así?, explícame mejor

Podemos ir de lo más simple a lo más complicado:

- Encontrar la cantidad de ocurrencias de una palabra en un texto.
- Encontrar la máxima cantidad de basura que puede ser almacenada en un sub-cubo tridimensional, teniendo en cuenta que forma un grafo isomorfo y que el triple de la suma de sus componentes no exceda el producto entre las mismas.

¿Cómo así?, explícame mejor

Podemos ir de lo más simple a lo más complicado:

- Encontrar la cantidad de ocurrencias de una palabra en un texto.
- Encontrar la máxima cantidad de basura que puede ser almacenada en un sub-cubo tridimensional, teniendo en cuenta que forma un grafo isomorfo y que el triple de la suma de sus componentes no exceda el producto entre las mismas.

Mentiras, sólo tengan en cuenta que **casi cualquier problema** puede ser resuelto utilizando programación.

¿Dónde es eso?

Nos reunimos todos los viernes, de 4 a 6 pm, en el bloque 18, salón 311.

Programas en C++ en consola

Para la ejecución de programas por consola

- Escribir el programa.

Programas en C++ en consola

Para la ejecución de programas por consola

- Escribir el programa.
- Compilar el programa.

Programas en C++ en consola

Para la ejecución de programas por consola

- Escribir el programa.
- Compilar el programa.
- Abrir la consola de Windows (Windows + R, escribir cmd).

Programas en C++ en consola

Para la ejecución de programas por consola

- Escribir el programa.
- Compilar el programa.
- Abrir la consola de Windows (Windows + R, escribir cmd).
- Establecer el directorio en la carpeta del ejecutable del programa.

Programas en C++ en consola

Para la ejecución de programas por consola

- Escribir el programa.
- Compilar el programa.
- Abrir la consola de Windows (Windows + R, escribir cmd).
- Establecer el directorio en la carpeta del ejecutable del programa.
- Ejecutar el programa escribiendo el nombre del archivo.

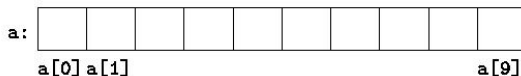
Contenido

2 Estructuras de datos lineales

- Arreglos estáticos
- Arreglos dinámicos
- Pilas
- Colas

Arreglos estáticos

- Es una colección de datos secuenciales que son guardados para luego ser consultados basandose en sus índices.
- Es claramente la estructura más común en maratones de programación.
- Típicamente se usan arreglos de 1, 2 y 3 dimensiones como máximo.
- Por ejemplo, un arreglo de 10 posiciones llamado *a* puede ser representado así:



Arreglos estáticos

A continuación, aprenderemos cómo declarar arreglos estáticos en C++.

Declaración de arreglos estáticos en C++

```
tipo_de_dato nombre [número_de_elementos];
```

Ejemplos:

```
int arr [10];  
string words [50];
```

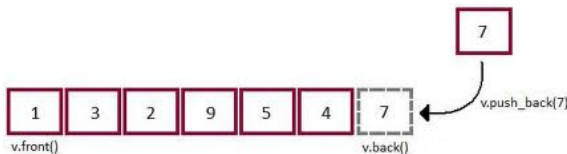
Arreglos estáticos

Un ejemplo de uso de arreglos estáticos es leerlo y luego recorrerlo hasta encontrar un 0.

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int MAXN = 10;
6  int nums[MAXN];
7
8  int
9  main() {
10     for (int i = 0; i < sizeof(nums) / sizeof(nums[0]); i++) {
11         cin >> nums[i];
12     }
13     for (int i = 0; i < sizeof(nums) / sizeof(nums[0]); i++) {
14         if (nums[i] == 0) {
15             cout << "I found a 0 in position: " << i << endl;
16             break;
17         }
18     }
19     return 0;
20 }
```


Arreglos dinámicos

- Los arreglos dinámicos o vectores son estructuras similares a los arreglos estáticos, excepto que son diseñados para permitir cambio de tamaño en tiempo de ejecución.
- Al igual que los arreglos, los elementos pueden ser accedidos por medio del índice de su posición, empezando por en índice 0.
- Las operaciones comunes en vectores en C++ son: *push_back()*, operador `[]` o *at()*, *clear()*, *erase()*, *size()*.
- En C++, para hacer uso de vectores es necesario importar la biblioteca *vector*, y en Java *ArrayList*.



Arreglos dinámicos

Veamos un ejemplo del uso de vectores en C++:

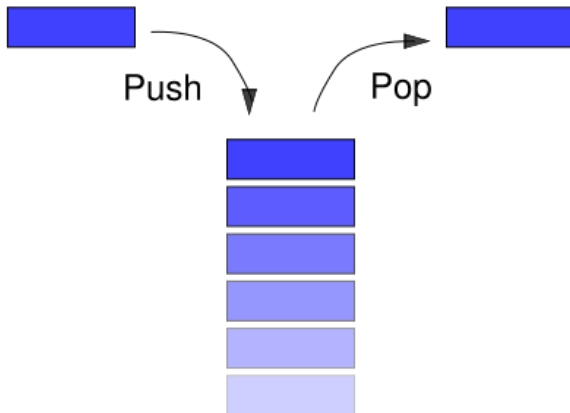
```
1  #include <iostream>
2  #include <vector> //Remember to include the library
3
4  using namespace std;
5
6  int
7  main() {
8      vector <int> nums(5, 5); //Initial size 5 with values {5, 5, 5, 5, 5}
9      for (int i = 0; i < nums.size(); i++) {
10         nums[i] = i * 2; //Store new value in position i
11     }
12     for (int i = 0; i < nums.size(); i++) {
13         cout << nums[i] << " ";
14     }
15     //Output will be: 0 2 4 6 8
16     return 0;
17 }
```

Programa ejemplo 1

Funcionamiento básico de arreglos y vectores
`arreglosYVectores.cpp`

Pilas

- Sólo permite una operación aplicada al tope de la pila, es decir, meter o sacar el último elemento del contenedor.
- **El último elemento que ingresó es el primer elemento en salir (LIFO)**



Pilas

Las operaciones que se pueden usar en una pila son:

Operaciones soportadas por una pila

push(x) - Inserta el elemento x al final de la pila.

pop() - Remueve el último elemento de la pila.

top() - Retorna el último elemento de la pila, sin removerlo.

empty() - Retorna verdadero si la pila está vacía.

size() - Retorna el tamaño de la pila.

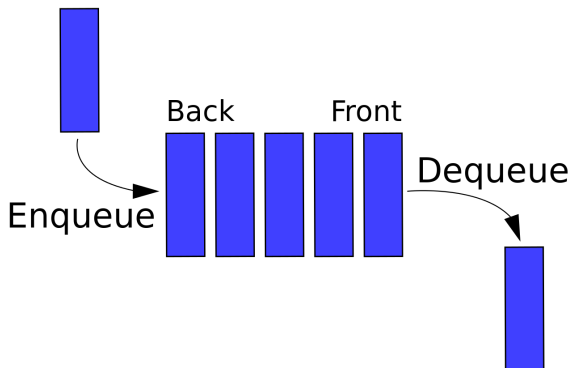
Pilas

Veamos un ejemplo de implementación de pila usando la librería de C++ *stack*.

```
1  #include <iostream>
2  #include <stack>           // Remember to include stack
3  using namespace std;
4
5  int main(){
6      stack <int> s;          // Create an integer stack
7      s.push(10);             // Insert 10
8      s.push(-1);            // Insert -1
9      cout << s.top() << endl; // Print -1
10     s.pop();                // Remove -1
11     cout << s.top() << endl; // Print 10
12     cout << s.size() << endl; // The stack size is 1
13     return 0;
14 }
```

Colas

- Las operaciones básicas son insertar al final de la cola, y eliminar del frente de la cola.
- El primer elemento insertado es el primero en salir de la cola (FIFO)**



Colas

Las operaciones que se pueden usar en una cola son:

Operaciones soportadas por una pila

push(x) - Inserta el elemento x al final de la cola.

pop() - Remueve el primer elemento (frontal) de la cola.

front() - Retorna el primer elemento de la cola, sin removerlo.

empty() - Retorna verdadero si la cola está vacía.

size() - Retorna el tamaño de la cola.

Colas

Veamos un ejemplo de implementación de cola usando la librería de C++ *queue*.

```
1 #include <iostream>
2 #include <queue>           // Remember to include queue
3 using namespace std;
4
5 int main(){
6     queue <int> q;          // Create an integer queue
7     q.push(10);             // Insert 10
8     q.push(-1);             // Insert -1
9     cout << q.front() << endl; // Print 10
10    q.pop();                 // Delete 10
11    cout << q.front() << endl; // Print -1
12    cout << q.size() << endl;  // The queue size is 1
13    return 0;
14 }
```

Programa ejemplo 2

Funcionamiento básico de pilas y colas `pilasYColas.cpp`

Contenido

3 Estructuras de datos no lineales

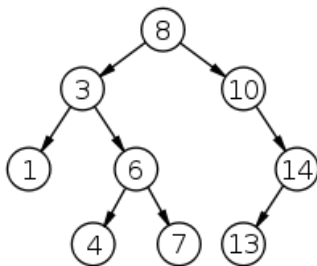
- Arbol binario de búsqueda
 - Mapa
 - Set
 - Cola de prioridad

Estructuras de datos no lineales

- Para algunos problemas, el almacenamiento lineal no es la mejor opción por su eficiencia en ciertas operaciones.
- Colecciones dinámicas no lineales como *Maps* o *HashTables* son las más adecuadas debido a su rendimiento en operaciones de inserción, búsqueda y eliminación.

Arbol binario de búsqueda

- Es una forma de organizar los datos en una estructura de árbol.
- Los items en una rama izquierda de un elemento x son menores que x , y los items en una rama derecha de x son mayores o iguales a x .
- En C++, implementaciones de árboles binarios de búsqueda están las bibliotecas *map* y *set*



Mapa

Mapa

Un mapa es un contenedor que guarda parejas de elementos. El primer elemento de la pareja (key) sirve para identificarla y el segundo elemento (mapped value) es el valor asociado a la llave.

Declaración

```
#include <map>
map <tipo_dato_key, tipo_dato_value> nombre;
```

Ejemplos:

```
map <string, int> m;
map <char, int> char2int;
```

Acceso a elementos de un mapa

Los elementos de un mapa se llaman por su llave así:

```
map <string, int> m;  
m["Hola"] = 3;  
int a = m["Cangrejo"];
```

Para ingresar un elemento se puede hacer así:

```
if (m.count["Nuevo"] == 0) m["Nuevo"] = 123;
```

Con la función `count` se busca cuántas veces está el elemento con la llave "Nuevo". Si el elemento no está, cuando accedemos a él con `[]` éste se crea automáticamente.

Cuando accedo con `[]` a un elemento del mapa que no existe este se crea con valores por defecto. Para strings es el string vacío `""` y para enteros es el número 0.

Complejidad del mapa

Complejidad

- Insertar / acceder un elemento al mapa es $O(\log n \times k)$ donde n es el número de elementos en el mapa y k es el tiempo que toma comparar dos llaves del mapa.
- Comparar dos enteros es $O(1)$, comparar dos strings es $O(m)$ donde m es la longitud de los strings.
- Insertar / acceder un elemento a un mapa con llaves strings es $O(\log n \times m)$ donde n son los elementos del mapa y m es la longitud del string.

Nota

Los elementos de un mapa se almacenan en orden de acuerdo a una función de comparación. Por defecto la función de comparación es la de menor, es decir que los elementos se almacenan de menor a mayor.

Recorrer un mapa

Para recorrer un mapa es necesario usar iteradores.

```
1  #include <iostream>
2  #include <map>
3  using namespace std;
4
5  int main(){
6      map <string, int> m;
7      m["b"] = 4;
8      m["bc"] = 1;
9      m["a"] = 3;
10     map <string, int> :: iterator it;
11     for (it = m.begin(); it != m.end(); it++){
12         cout << "( " << it->first << " " << it->second << " ) ";
13         // cout << "( " << (*it).first << " " << (*it).second << " ) ";
14     }
15     return 0;
16 }
17 // La funcion imprime ( a 3 ) ( b 4 ) ( bc 1 )
```

Otras funciones en el mapa

Otras funciones que se pueden hacer con el mapa son:

- Recorrerlo al revés con `rbegin` y `rend`
- Obtener el tamaño con `size`
- Borrar el contenido con `clear`
- Insertar elementos (si no están antes) con `emplace`
- Borrar elementos con `erase`
- Buscar un elemento con `find`

Para más información mirar

<http://www.cplusplus.com/reference/map/map/>

Set

Set

Un set es un contenedor que guarda conjuntos, es decir grupos de elementos iguales donde cada elemento aparece una sola vez. Los elementos de un set se almacenan en orden de acuerdo a una función de comparación. Por defecto la función de comparación es la de menor, es decir que los elementos se almacenan de menor a mayor.

Declaración

```
#include <set>
set <tipo_dato> nombre;
Ejemplos:
set <string> s;
set <int> amigos;
```

Set

Sobre un set se pueden hacer las siguientes operaciones.

insert Inserta un elemento al set. Ejemplo:

```
amigos.insert(9)
```

count Cuenta cuántas veces aparece un elemento (0 o 1 vez). Ejemplo: `amigos.count(3)`

find Retorna un iterador al lugar donde está el elemento. Ejemplo: `amigos.find(3)`

erase Elimina un elemento del set. Ejemplo:
`amigos.erase(amigos.find(3))`

Todas las operaciones anteriores tienen una complejidad $O(\log n \times k)$ donde n es el número de elementos del set y k es el tiempo que toma comparar dos elementos.

Para más información mirar

<http://www.cplusplus.com/reference/set/set/>

Recorrer un set

Para recorrer un set es necesario usar iteradores.

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main(){
6      set <int> s;
7      s.insert(4);
8      s.insert(-1);
9      s.insert(3);
10     s.insert(4);
11     set <int> :: iterator it;
12     for (it = s.begin(); it != s.end(); it++){
13         cout << *it << " ";
14     }
15     return 0;
16 }
17 // La funcion imprime -1 3 4
```

Cola de prioridad

Un heap se puede representar en C++ como una cola de prioridades así: `#include <queue>`

```
priority_queue <tipo_dato> nombre;
```

Ejemplos:

```
priority_queue <int> heap;
```

```
priority_queue <pair <int, int> > q;
```

Para más información mirar: [http:](http://www.cplusplus.com/reference/queue/priority_queue/)

[//www.cplusplus.com/reference/queue/priority_queue/](http://www.cplusplus.com/reference/queue/priority_queue/)

Operaciones

La cola de prioridades (heap) soporta las siguientes operaciones

push Inserta un elemento

pop Extrae un elemento

top Retorna el máximo elemento de la cola (heap)

size Retorna el tamaño de la cola (heap)

La cola de prioridades se ordena de acuerdo a la función de ordenamiento $<$ (menor que) por lo que retorna el elemento con el cual todos los demás comparan menor que él, es decir, el mayor elemento.

Para recordar...

- Ejercicio de entrada y salida de datos por archivo.

Para recordar...

- Ejercicio de entrada y salida de datos por archivo.
- Ejercicio de leer hasta fin de archivo.

Para recordar...

- Ejercicio de entrada y salida de datos por archivo.
- Ejercicio de leer hasta fin de archivo.
- Explicación funcionamiento plataforma de juzgamiento BOCA.

Para recordar...

- Ejercicio de entrada y salida de datos por archivo.
- Ejercicio de leer hasta fin de archivo.
- Explicación funcionamiento plataforma de juzgamiento BOCA.
- Asignación de usuarios para comenzar la maratón.

Usuarios BOCA

Equipo	Usuario
team1	Épsilon X
team2	Apocafit
team3	decode-team
team4	Praise the Sun
team5	Los Crafters
team6	ENCRYPTED
team7	LosSuper op
team8	Los PoderOsitos
team9	Pink floyd

Gracias

Esta presentación fue basada en el libro Competitive Programming 3 de Steven Halim y Felix Halim.
Ciertos elementos contenidos en las diapositivas fueron tomados de:

[https://github.com/anaechavarria/
SemilleroProgramacion/tree/master/Diapositivas](https://github.com/anaechavarria/SemilleroProgramacion/tree/master/Diapositivas)