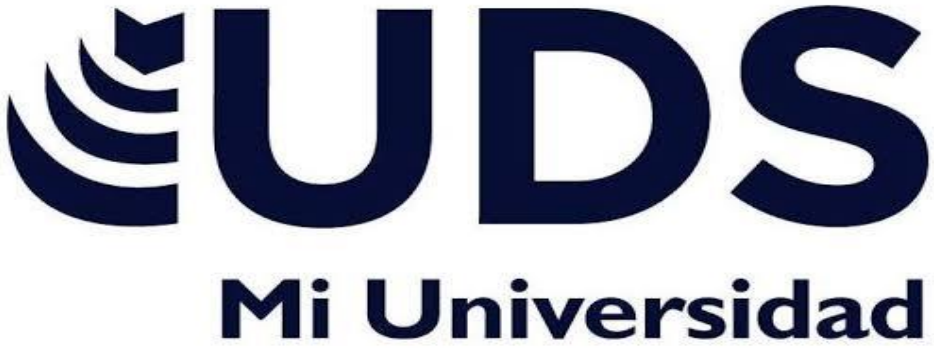


Supernota

Nombre del alumno: Yahir Aguilar Sicalhua.
Nombre del tema: Diseño y Análisis de Algoritmos & Tipos de Datos Abstractos Fundamentales
Parcial: 1.
Nombre de la materia: Algoritmos y Estructuras de Datos.
Nombre del profesor: Emmanuel Eduardo Sánchez Pérez.
Nombre de la licenciatura: Ingeniería en Sistemas Computacionales.
Cuatrimestre: 5.

Unidad I

Diseño y Análisis de Algoritmos

1.1 Conceptos básicos de algoritmos.
Formalmente definimos un algoritmo como un conjunto de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema.

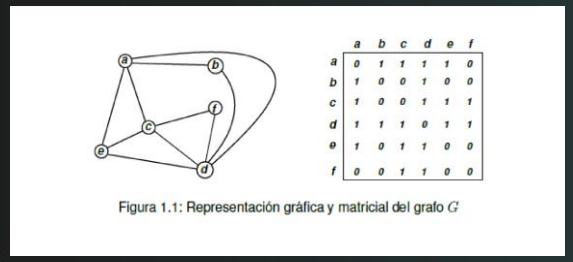
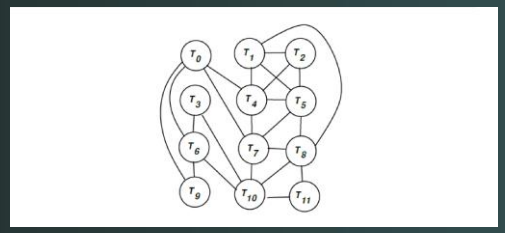


Figura 1.1: Representación gráfica y matricial del grafo G

1.1.1 Introducción básica a grafos.
Desde el punto de vista de la teoría de conjuntos un grafo es un subconjunto del conjunto G de pares de vértices. Un par de vértices está en el grafo si existe una arista que los conecta.

1.1.2. Planteo del problema mediante grafos.
Podemos plantear el problema dibujando un grafo donde los vértices corresponden a las tareas y dibujaremos una arista entre dos tareas si son incompatibles entre sí (modifican el mismo objeto).



1.1.3. Algoritmo de búsqueda exhaustiva.
Consideremos primero un algoritmo de "búsqueda exhaustiva" es decir, probar si el grafo se puede colorear con 1 solo color (esto solo es posible si no hay ninguna arista en el grafo).

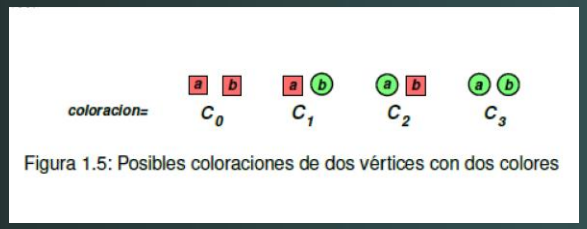


Figura 1.5: Posibles coloraciones de dos vértices con dos colores

1.1.4. Generación de las coloraciones.
Notemos primero que el procedimiento para generar las coloraciones es independiente de la estructura del grafo (es decir de las aristas), sólo depende de cuantos vértices hay en el grafo y del número de colores que pueden tener las coloraciones.

1.2.Tipos abstractos de datos.
Un "Tipo Abstracto de Datos" (TAD) es la descripción matemática de un objeto abstracto, definido por las operaciones que actúan sobre el mismo.

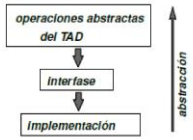


Figura 1.12: Descripción de lo diferentes niveles de abstracción en la definición de un TAD

1.2.2. Interfaz del TAD conjunto.
La "interfaz" es el conjunto de operaciones (con una sintaxis definida) que producen las operaciones del TAD.

1.2.3. Implementación del TAD conjunto.
Finalmente, la "implementación" de estas funciones, es decir el código específico que implementa cada una de las funciones declaradas en la interfaz.

1.4. Conteo de operaciones para el cálculo del tiempo de ejecución.
Comenzaremos por calcular el tiempo de ejecución de ´estas. La regla básica para calcular el tiempo de ejecución de un programa es ir desde los lazos o construcciones más internas hacia las más externas.

1.4.1. Bloques if.

Para evaluar el tiempo de un bloque `if`

```
if(<cond>) {
  <body>
}
```

podemos o bien considerar el peor caso, asumiendo que <*body*> se ejecuta siempre

$$T_{\text{peor}} = T_{\text{cond}} + T_{\text{body}} \tag{1.47}$$

o, en el caso promedio, calcular la probabilidad P de que <*cond*> de verdadero. En ese caso

$$T_{\text{prom}} = T_{\text{cond}} + PT_{\text{body}} \tag{1.48}$$

Notar que T_{cond} no está afectado por P ya que la condición se evalúa siempre. En el caso de que tenga un bloque `else`, entonces

```
if(<cond>) {
  <body-true>
} else {
  <body-false>
}
```

1.4.2. Lazos.
El caso más simple es cuando el lazo se ejecuta un número fijo de veces, y el cuerpo del lazo tiene un tiempo de ejecución constante.

```
for (i=0; i<N; i++) {
  <body>
}
```

donde $T_{\text{body}} =$ constante. Entonces

$$T = T_{\text{ini}} + N(T_{\text{body}} + T_{\text{inc}} + T_{\text{stop}}) \tag{1.50}$$

Unidad II

Tipos de Datos Abstractos Fundamentales.

2.1. El tad lista.

Las listas constituyen una de las estructuras lineales más flexibles, porque pueden crecer y acortarse según se requiera, insertando o suprimiendo elementos tanto en los extremos como en cualquier otra posición de la lista.

2.1.1. Descripción matemática de las listas.

Desde el punto de vista abstracto, una lista es una secuencia de cero o más elementos de un tipo determinado, que en general llamaremos elem_t, por ejemplo, int o double.

2.1.2. Operaciones abstractas sobre listas.

El algoritmo más simple consiste en un doble lazo, en el cual el lazo externo sobre i va desde el comienzo hasta el último elemento de la lista.

2.2. El TAD pila.

Básicamente es una lista en la cual todas las operaciones de inserción y borrado se producen en uno de los extremos de la lista.

2.2.1. Una calculadora RPN con una pila.

- Si el usuario ingresó un operando, entonces simplemente se almacena en la pila.
- Si ingresó un operador θ se extraen dos operandos del tope de la pila, digamos t el tope de la pila y u el elemento siguiente, se aplica el operador a los dos operandos (en forma invertida) es decir u θ t y se almacena el resultado en el tope de la pila.

2.3. El TAD cola.

Por contraposición con la pila, la cola es un contenedor de tipo “FIFO” (por “First In First Out”, el primero en entrar es el primero en salir).

2.3.1.2. Tiempo de ejecución.

Consideremos ahora el tiempo de ejecución de este algoritmo. El lazo sobre j se ejecuta n − 1 veces, y el lazo interno se ejecuta, en el peor caso j − 1 veces, con lo cual el costo del algoritmo es, en el peor caso:

$$T_{\text{peor}}(n) = \sum_{j=1}^{n-1}(j-1) = O(n^2) \qquad (2.13)$$

2.3.1.3. Particularidades al estar las secuencias pares e impares ordenadas.

Al estar las posiciones pares e impares ordenadas entre sí puede ocurrir que en promedio el desplazamiento sea menor, de hecho, generando vectores en forma aleatoria.

2.3.1.4. Algoritmo de intercalación con una cola auxiliar

```
void merge(vector<int> &a) {
 int n = a.size();
 // C = cola vacia ...
 int p=0, q=0, minr, maxr;
 while (q<n) {
  // minr = min(a_q,a_{q+1}, maxr = max(a_q,a_{q+1}
  if (a[q]<=a[q+1]) {
   minr = a[q];
   maxr = a[q+1];
  } else {
   maxr = a[q];
   minr = a[q+1];
  }
  // Apendizar todos los elementos del frente de la cola menores que
  // min(a_q,a_{q+1}) al rango [0,p), actualizando eventualmente
  while ( /* C no esta vacia ... */) {
   x = /* primer elemento de C ... */;
   if (x>minr) break;
   a[p++] = x;
   // Saca primer elemento de C ...
  }
  a[p++] = minr;
  a[p++] = minr;
  // Apendizar 'maxr' al rango [0,p) ...
  q += 2;
 }
 // Apendizar todos los elementos en C menores que
 // min(a_q,a_{q+1}) al rango [0,p)
 // ...
}
```

2.4. El TAD correspondencia.

La “correspondencia” o “memoria asociativa” es un contenedor que almacena la relación entre elementos de un cierto conjunto universal D llamado el “dominio” con elementos de otro conjunto universal llamado el “contradominio” o “rango”.

2.4.1. Interfaz simple para correspondencias.

```
class iterator_t {/* ... */};

class map {
 private:
  // ...
 public:
  iterator_t find(domain_t key);
  iterator_t insert(domain_t key,range_t val);
  range_t& retrieve(domain_t key);
  void erase(iterator_t p);
  int erase(domain_t key);
  domain_t key(iterator_t p);
  range_t& value(iterator_t p);
  iterator_t begin();
  iterator_t next(iterator_t p);
  iterator_t end();
  void clear();
  void print();
};
```

Código 2.24: *Interfaz básica para correspondencias. [Archivo: mapbas.h]*

Fuentes Bibliográficas:

MWConsultores. (s/f) Antología UDS. Com.mx. Recuperado el 13 de marzo de 2023, de https://plataformaeducativauds.com.mx/libro.php?idLibro=167873550016