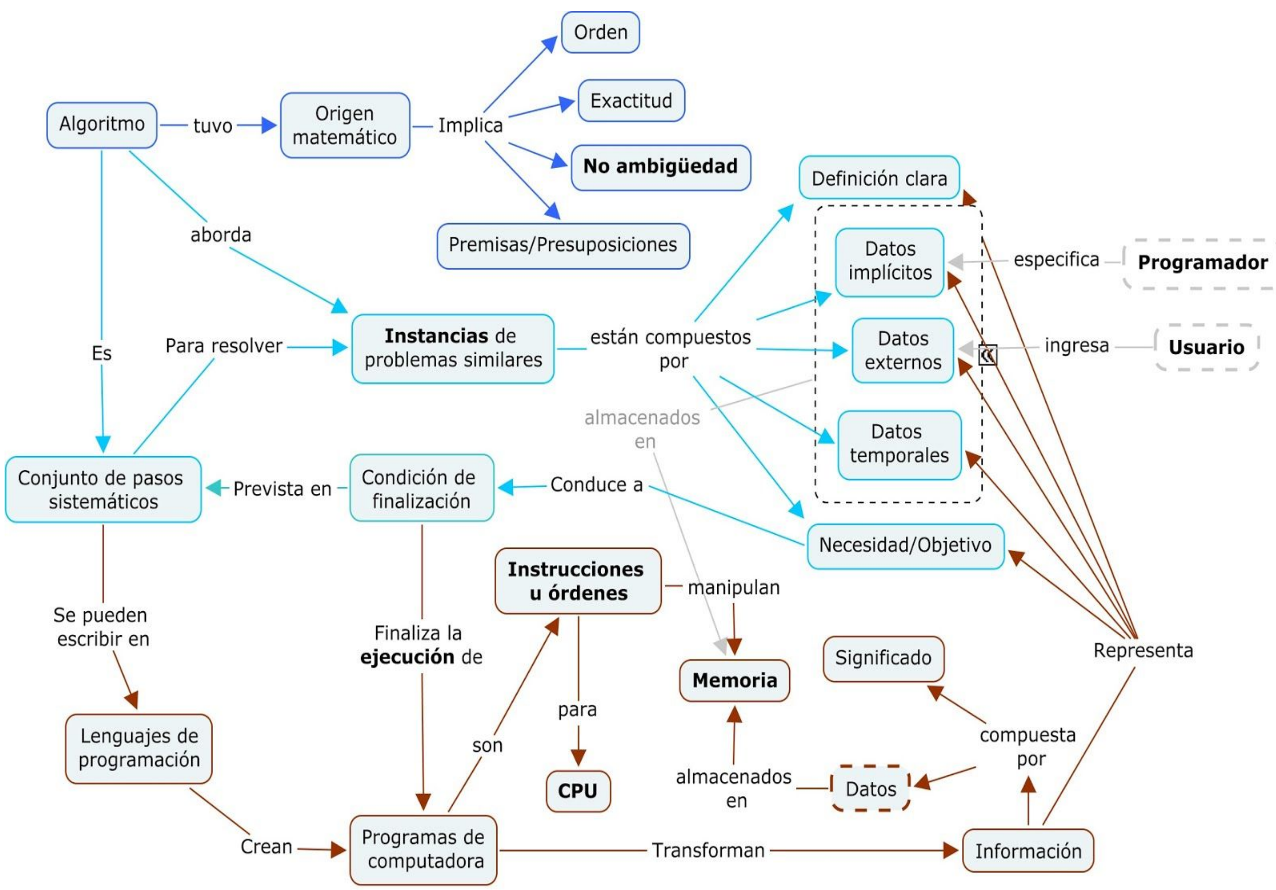
Practicas de Algorítmica



Alumno: Pablo Martínez Ruano
Profesor: Jose Antonio Garcia Soria
Grupo: C
Comentarios:Debido a que la ultima clase de practicas no pude asistir ninguna practica esta validada.

Practica 1.

Analizaremos la eficiencia de dos algoritmos, para ello analizaremos el peor caso, el caso promedio y el mejor caso.

Algoritmo 1

```
i:= 1
mientras i <= n hacer
        si a[i] >= a[n] entonces
                a[n]:=a[i]
        finsi
        i:= i *2
finmientras
```

Analisis del algoritmo para el peor de los casos

$$t(n) = 1 + \sum_{i=1}^{logn} 3 + 3 + 2 = 1 + 8logn$$

Su orden de eficiencia sera O (logn).

Analisis del algoritmo para el mejor de los casos

$$t(n) = 1 + \sum_{i=1}^{logn} 3 + 2 = 1 + 5logn$$

Su orden de eficiencia sera O (logn).

Analisis del algoritmo para el caso promedio

Al ser la eficiencia del peor caso y del mejor caso iguales, la eficiencia del caso promedio es del mismo orden O (logn).

Algoritmo 2

```
cont:=0
para i:= 1,...,n hacer
        pata j:= 1,...,i-1 hacer
                si a[i] < a[j] entonces
                        cont:= cont + 1
                finsi
        finpara
finpara
```

Analisis del algoritmo para el peor de los casos

$$t(n) = 1 + \sum_{i=1}^{n}\sum_{j=1}^{i-1} 3 + 2 = 1 + \sum_{i=1}^{n}(5i - 5) = 1 + 5(\tfrac{n(n+1)}{2}) - 5$$

Su orden de eficiencia sera O ($n^2/2$)

Analisis del algoritmo para el mejor de los casos

$$t(n) = 1 + \sum_{i=1}^{n}\sum_{j=1}^{i-1} 3 = 1 + \sum_{i=1}^{n}(3i - 3) = 1 + 3(\tfrac{n(n+1)}{2}) - 3$$

Su orden de eficiencia sera O ($n^2/2$)

Analisis del algoritmo para el caso promedio

Al ser la eficiencia del peor caso y del mejor caso iguales, la eficiencia del caso promedio es del mismo orden O ($n^2/2$)

Practica 2 Método de Selección

Sea T[1..n] un array (no ordenado) de enteros, y sea s un entero entre 1 y n. El problema de selección consiste en encontrar el elemento que se encontraría en la posición s si el array estuviera ordenado.

Analisis de eficiencia

Codigo del pivote

```
#include<iostream>
#include<stdlib.h>

using namespace std;

void intercambiar (int &x, int &y){
   int aux=x;
   x=y;
   y=aux;
}

int pivote(int* a, int i, int j) {
   int s=i;
   i++;
   while(i<j) {
         while(a[i]<=a[s] && i<j){
                 i++;
         }
         while(a[j]>a[s]){
                 j--;
         }
         if(i<j){
                 intercambiar(a[i],a[j]);
         }
   }
   intercambiar(a[s],a[j]);
   return j;
}

int mediana(int *a,int tam){
   int m=tam/2,r=0,i=0,j=tam-1;
```

```
    while(r!=m){
        r=pivote(a,i,j);
        if(r>m){
            j=r-1;
        }else{
            i=r+1;
        }
    }
    return a[r];
}

int main(){
    int tam,med;
    cout<<    "Introduce el tamaño del vector: "<<endl;
    cin>>tam;

    int a[tam];

    cout<<"Introduce los elementos del vector: "<<endl;
    for(int i=0;i<tam;i++){
        cin>>a[i];
    }

    med=mediana(a,tam);
    cout<<endl;
    for(int i=0;i<tam;i++){
        cout<<a[i]<<", ";
    }

    cout<<"Mediana del vector: "<<med<<endl;

    return 0;
}
```

Analisis del algoritmo para el peor de los casos

t(mediana)= $7 + \sum_{i=1}^{logn} (3 + t(pivote))$

t(pivote) = O(n)

t(mediana) = $7 + \sum_{i=1}^{logn} (3 + n) = 7 + 3logn + nlog = O(nlogn)$

t(n) = $5 + \sum_{i=1}^{n} + 1 + 1 + nlogn = 6 + n + nlogn = O(nlogn)$

Analisis del algoritmo para el mejor de los casos

t(mediana)= $7 + \sum_{i=1}^{logn} (3 + t(pivote))$

t(pivote) = O(n)

t(mediana) = $7 + \sum_{i=1}^{logn} (3 + n) = 7 + 3logn + nlog = O(nlogn)$

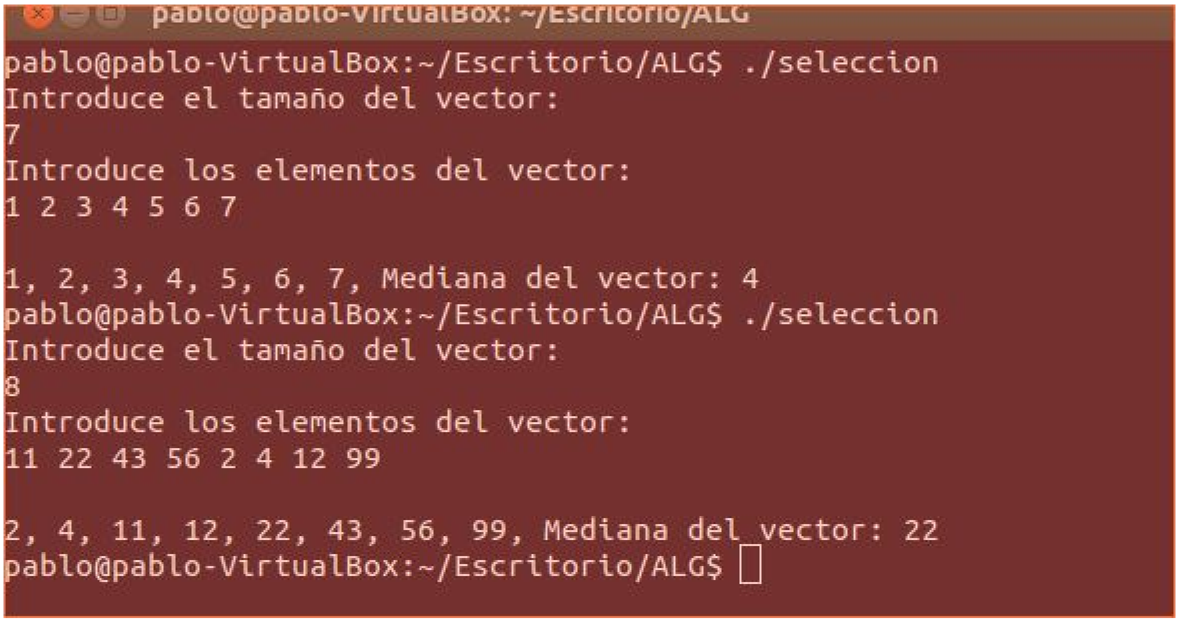
t(n) = $5 + \sum_{i=1}^{n} + 1 + 1 + nlogn = 6 + n + nlogn = O(nlogn)$

Analisis del algoritmo para el caso promedio

Al ser la eficiencia del peor caso y del mejor caso iguales, la eficiencia del caso promedio es del mismo orden $O(nlogn)$

Ejercicios de aplicacion de la practica

Para esta practica en vez de 2 capturas he realizado en una sola captura 2 veces su ejecucion



Practica 3 Coloracion de Grafos

El objetivo de esta practica trata de implementar un método de coloración de grafos, que es la asignación de un color a cada nodo, de forma que dos nodos unidos con un arco tengan siempre distinto color, tambien se realizara una coloración utilizando el número mínimo de colores. Para ello se usara un algoritmo voraz, dicho de otra manera, un algoritmo de avance rápido. Para ello se usara una heuristica voraz para obtener una solución rápida. Inicialmente ningún nodo tiene un color asignado.

Representación de la solución: una solución tiene la forma (c1, c2, ..., cn), donde ci es el color asignado al nodo i. La solución es válida si para toda arista (v, w) ∈ A, se cumple que cv ≠ cw.

Analisis de eficiencia

Analisis Factible(nodo x, int a)

```
bool factible(nodo x, int a) {
   for ( int i=0;i<x.nadyacentes;i++){
       if (a==x.adyacentes[1] ->color){
           return false;
        }
    }
   return true;
}
```

$$t(n) = \sum_{i=0}^{n-1} 2 = 2n = O(n)$$

Analisis del algoritmo para el peor de los casos

```
int colorear(nodo* grafo,int tamaño) {
    int colores=1;
    for (int i=0;1<tamaño;i++) {
        if(factible (grafo[i], colores)) {
            grafo[i] .color=colores;
        }else{
                colores++;
                grafo[i].color=colores;
        }
    }
    return colores;
}
```

$$t(n) = 1 + \sum_{i=0}^{n-1} j(factible) + 2 + 2 = \sum_{i=0}^{n-1} 2n + 4 = 2n^2 + 4n = O(n^2)$$

Analisis del algoritmo para el mejor de los casos

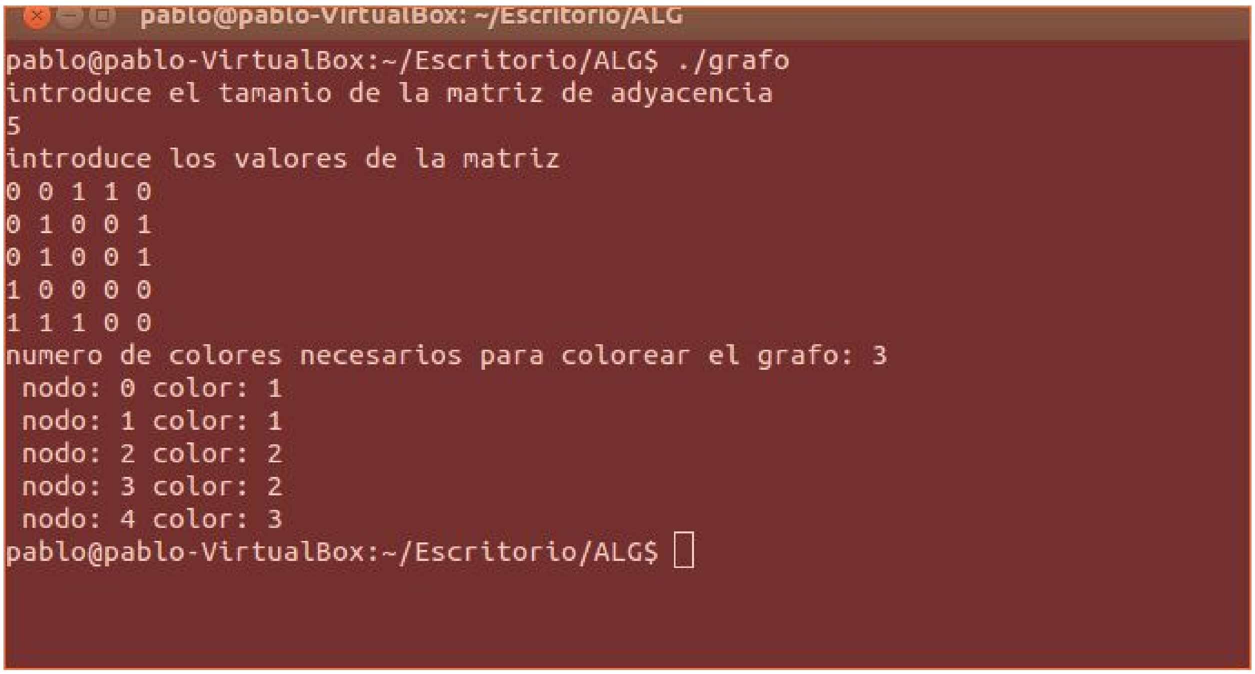
```
int colorear(nodo* graf,int tam) {
    int colores=1;
    for (int i=0;1<tam;i++) {
        if(factible (graf[i], colores)) {
            graf[i] .color=colores;
        }else{
                colores++;
                graf[i].color=colores;
        }
    }
    return colores;
}
```

$$t(n) = 1 + \sum_{i=0}^{n-1} j(factible) + 1 = \sum_{i=0}^{n-1} 2n + 1 = 2n^2 + n = O(n^2)$$

Analisis del algoritmo para el caso promedio

Al ser la eficiencia del peor caso y del mejor caso iguales, la eficiencia del caso promedio es del mismo orden O(n^2)

Ejercicio de aplicacion de la practica



En este caso no nos sale el resultado mas optimo para realizar la aplicacion, ya que nos sale 3 colores cuando lo mas optimo seria 2 colores. Sin embargo la finalidad de voraces es trata de realizar el ejercicio con rapidez con un buen resultado incluso si a veces no da el mas optimo.

Practica 4 Cambio de monedas

Dado un conjunto de n tipos de monedas, cada una con valor ci, y dada una cantidad P, encontrar el número mínimo de monedas que tenemos que usar para obtener esa cantidad.

El algoritmo voraz es muy eficiente, pero sólo funciona en un número limitado de casos, para ello usaremos programacion dinamica, de esta manera definiremos el problema en problemas mas pequeños y conseguiremos un buen resultado y buen tiempo de ejecución.

Analisis de eficiencia

Analisis del algoritmo para el peor de los casos

```
...
int n = 3, p=8, D[3] [9], c[]={1,4,6},x[3] = {0,0,0};

for (int i=0; i<n;i++) {
            D[i] [0]=0;
}
for (int i=0; i<n; i++) {
            for (int j=1; j <=p;j++) {
                  int a,b =1000;
                  if (i>=1) a = D[i-1] [j];
                  if (j>=c[i]) b = 1+D[i] [j-c[i]];
                 if (a<=b) {
                        D[i] [j] = a;
                   }else{
                        D[i] [j] = b;
                   }
            }
}
int i = n-1;
int j = p;

while (i!=0 && j!= 0) {
            if(D[i] [j] == D[i-1] [j] )
                        i=i-1;
            else{     x[i]=x[i]+1;
                        j=j-c[i];
            }
}
...
```

$$t(n) = 14 + \sum_{i=0}^{n-1} 2 + \sum_{i=0}^{n-1}\sum_{j=1}^{n} 2+1+2+2+1+2+2+1+2\sum_{i=0}^{n-2} 1+4+3 = 14 + \sum_{i=0}^{n-1} 2 + \sum_{i=0}^{n-1}\sum_{j=1}^{n} 15 - 8n - 8 =$$

$$t(n) = 14 + \sum_{i=0}^{n-1} 2 + \sum_{i=0}^{n-1} 7n + 8n^2 = 14 + \sum_{i=0}^{n-1} 2 + 7n^2 + 8n^3 = 14 + 2n + 7n^3 + 8n^4 = O(n^{4)}$$

Analisis del algoritmo para el mejor de los casos

```
...
int n = 3, p=8, D[3] [9], c[]={1,4,6},x[3] = {0,0,0};

for (int i=0; i<n;i++) {
        D[i] [0]=0;
}
for (int i=0; i<n; i++) {
        for (int j=1; j <=p;j++) {
            int a,b =1000;
            if (i>=1) a = D[i-1] [j];
            if (j>=c[i]) b = 1+D[i] [j-c[i]];
           if (a<=b) {
                D[i] [j] = a;
             }else{
                D[i] [j] = b;
             }
        }
}
int i = n-1;
int j = p;

while (i!=0 && j!= 0) {
        if(D[i] [j] == D[i-1] [j] )
                i=i-1;
        else{     x[i]=x[i]+1;
                j=j-c[i];
        }
}
...
```

$$t(n) = 14 + \sum_{i=0}^{n-1} 2 + \sum_{i=0}^{n-1}\sum_{j=1}^{n} 2 + 1 + 2 + 2 + 1 + \sum_{i=0}^{n-2} 1 = 14 + \sum_{i=0}^{n-1} 2 + \sum_{i=0}^{n-1}\sum_{j=1}^{n} 9 + n - 1 =$$

$$t(n) = 14 + \sum_{i=0}^{n-1} 2 + \sum_{i=0}^{n-1} 8n + n^2 = 14 + \sum_{i=0}^{n-1} 2 + 8n^2 + n^3 = 14 + 2n + 8n^3 + n^4 = O(n^4)$$

Analisis de eficiencia del caso promedio

Al ser la eficiencia del peor caso y del mejor caso iguales, la eficiencia del caso promedio es del mismo orden O(n^4)

Ejercicio de aplicacion de la practica

Como se puede ver obtenemos el resultado mas optimo y tiempo bastante bajo.