

Programación III

TRABAJO PRÁCTICO OBLIGATORIO

GRUPO de TRABAJO  
“La barba de Godio”

TEMAS  
Algoritmo de Floyd  
Problema del Viajante

PROFESOR: Lic. Esteban Calabria

|  |  |
| --- | --- |
| Integrante | Legajo |
| CARIATI, Yamila Soledad | 1064010 |
| MALDONADO, Martín | 1056424 |
| RICCOMBENI, Maximiliano | 1017172 |
| VIEIRO, Javier Martín | 1062141 |

Segundo Cuatrimestre 2015

Viernes Noche

# Índice

[1. Índice 1](#_Toc434163084)

[2. Enunciado 2](#_Toc434163085)

[2.1. Algoritmo de Floyd 2](#_Toc434163086)

[2.2. Problema del Viajante 2](#_Toc434163087)

[3. Resolución 3](#_Toc434163088)

[3.1. Algoritmo de Floyd 3](#_Toc434163089)

[3.2. Problema del Viajante 4](#_Toc434163090)

# Enunciado

## Algoritmo de Floyd

Se desea que el alumno investigue y documente el algoritmo de Floyd. Se desea que desarrolle los siguientes puntos. Cada uno de los siguientes puntos debe figurar como un apartado distinto de este documento.

* Explicación con sus propias palabras del algoritmo citando fuentes de donde obtuvo la información. Para poder aprobar este punto el grupo debe demostrar poder realizar una investigación exhaustiva y a conciencia y de calidad profesional.
* Backtracking
  + Pseudocódigo de backtracking que aplique la lógica del algoritmo de floyd
  + Realizar un TDA grafo adecuado para la implementación
  + Análisis de complejidad del punto anterior
  + Implementación en java
* Programación dinámica
  + Pseudocódigo
  + Análisis de complejidad del mismo
  + Implementación en java

## Problema del Viajante

Se desea investigar y entender el famoso problema del viajante y su relación con la complejidad algorítmica. Se desea completar los siguientes puntos:

* Definición del problema
* Implicancias del problema a nivel complejidad temporal
* Pseudocódigo de una resolución del problema
* Análisis de complejidad del pseudocódigo anterior
* Implementación en java del pseudocódigo propuesto

# Resolución

## Algoritmo de Floyd

# Creado por Bernard Roy en 1959 es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica, teniendo en cuenta que este tipo de programación tiene como fin encontrar una solución óptima a dicho problema recursivamente. Compara todos los posibles caminos a través del [grafo](https://es.wikipedia.org/wiki/Grafo) entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V^3comparaciones Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

### Pseudocódigo

**Algoritmo**

Floyd

**Entrada**

G grafo de entrada

**Salida**

R grafo con una arista con la distancia mínima entre cada par de vértices

**Pseudocódigo**

inicializarGrafo(R)

copiarGrafo(G, R)

VerticesK = Vertices(G)

para cada k de VerticesK

VerticesI = Vertices(G)

para cada i de VerticesI

VerticesJ = Vertices(G)

para cada j de VerticesJ

si i!=j Y existeArista(R, i, k) Y existeArista(R, k, j)

si existeArista(R, i, j)

si pesoArista(R, i, k) + pesoArista(R, k, j) < pesoArista(R, i, j)

agregarArista(R, i, j, pesoArista(R, i, k) + pesoArista(R, k, j))

fin si

sino

agregarArista(R, i, j, pesoArista(R, i, k)+pesoArista(R, k, j))

fin si

fin si

fin para

fin para

fin para

devolver R

### Implementacion Java

public static GrafoDirTDA <Integer> floyd(GrafoDirTDA <Integer> g) {

ConjuntoTDA <Integer> conjuntoI, conjuntoJ, conjuntoK;

int i, j, k;

GrafoDirTDA <Integer> r = new GrafoDir < Integer > ();

r.InicializarGrafo();

// Copio el grafo original

conjuntoK = g.Vertices();

while (!conjuntoK.conjuntoVacio()) {

k = conjuntoK.elegir();

conjuntoK.sacar(k);

r.AgregarVertice(k);

}

conjuntoK = g.vertices();

while (!conjuntoK.conjuntoVacio()) {

k = conjuntoK.elegir();

conjuntoK.sacar(k);

ConjuntoI = g.Adyacentes(k);

While(!conjuntoI.conjuntoVacio()) {

i = conjuntoI.elegir();

conjuntoI.sacar(i);

r.AgregarArista(k, i, g.PesoArista(k, i));

}

}

conjuntoK = g.Vertices();

while (!conjuntoK.conjuntoVacio()) {

k = conjuntoK.elegir();

conjuntoK.sacar(k);

conjuntoI = g.Vertices();

conjuntoI.sacar(k);

while (!conjuntoI.conjuntoVacio()) {

i = conjuntoI.elegir();

conjuntoI.sacar(i);

if (r.ExisteArista(i, k)) {

conjuntoJ = r.Adyacentes(k);

conjuntoJ.sacar(i);

while (!conjuntoJ.conjuntoVacio()) {

j = conjuntoJ.elegir();

conjuntoJ.sacar(j);

if (r.ExisteArista(i, j)) {

if (r.PesoArista(i, k) + r.PesoArista(k, j) < r.Pesoarista(i, j)) {

r.AgregarArista(i, j, r.PesoArista(i, k) + r.PesoArista(k, j);

}

} else {

r.AgregarArista(i, j, r.PesoArista(i, k) + r.pesoarista(k, j));

}

}

}

}

}

return r;

}

## Problema del Viajante

|  |
| --- |
| Travelling Salesman Problem |
| *Fuente: xkcd (http://www.xkcd.com/399/)* |