**Universidad de Santiago de Chile**

Facultad de Ingeniería Informática

# Ciclo Hamiltoniano en grafo no dirigido

Taller de Programación

Profesor: Pablo Román Asenjo
Autor: Nícolas André Farfán Cheneaux

Julio 2022

# Contents

# Ciclo Hamiltoniano

## 1 Descripción formal del problema

### 1.1 Ciclos Hamiltonianos

Según Wikipedia un ciclo Hamiltoniano visita todos los nodos de un grafo y regresa al origen sin pasar dos veces por un mismo nodo, en contexto al problema, esto es exáctamente lo que buscamos visitar todos los nodos, pero encontrando el camino menos costoso.

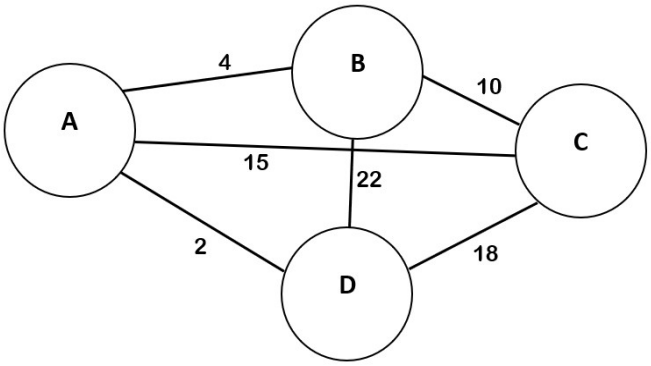


*Figura 1: Representación de un grafo no dirigido, cada nodo representa una ciudad [A,B,C,D] y cada arista tiene el valor del costo por recorrer tal camino[2,15,9,18,10,8](Este grafo debe ser conexo y ponderado).*

## 2 Aplicaciones prácticas del problema

Este problema de combinatoria tiene múltiples aplicaciones:

- Cómo su nombre lo indica, una compañía puede utilizar este método para encontrar el camino con menor costo en que un vendedor o repartidor puede ir a "n" ciudades y regresar en el menor costo posible, ejemplos cómo repartición de correos, movilidad escolar.

- Para colocar alambrado de cable de fibra óptica por ejemplo, entre ciudades, se prefiere que sea el menor costo y distancia posible.

- A gran escala, con muchas ciudades(nodos) en un aeropuerto es clave para calcular las distancias y recorrido que debe serguir un vuelo al menor costo.

- En GPS o Google Maps puede ser utilizado para encontrar la ruta más rápida a un número de "n" destinos ingresados por el usuario.

- En términos de ciencias de la computación encontrar la menor distancia o diferencia entre un dato u otro, optimiza en tiempo y espacio de ejecución.

- En algunas areas de la informática o manejo de memoria, un grafo dirigido puede representar un conjunto de instrucciones en las que deben ser ejecutadas en un cierto orden y si no se ejecuta en ese orden, no funciona, por lo tanto para encontrar tal proceso es necesario encontrar el camino ideal o correcto.

# 3 Estrategia de Solución

## 3.1 Estrategia 1: BackTracking - Fuerza Bruta

Para esta entrega se ejemplifica un grafo donde sus aristan tienen un costo para orientar a la solución propuesta. Cómo su nombre lo indica esta estrategia obtiene todas las posibles soluciones a recorrer, por lo tanto tiene un costo de n! al existir n! posibles caminos.
**Desde el punto de vista heurísitco esta solución no es eficiente y es posible encontrar una mejor.**

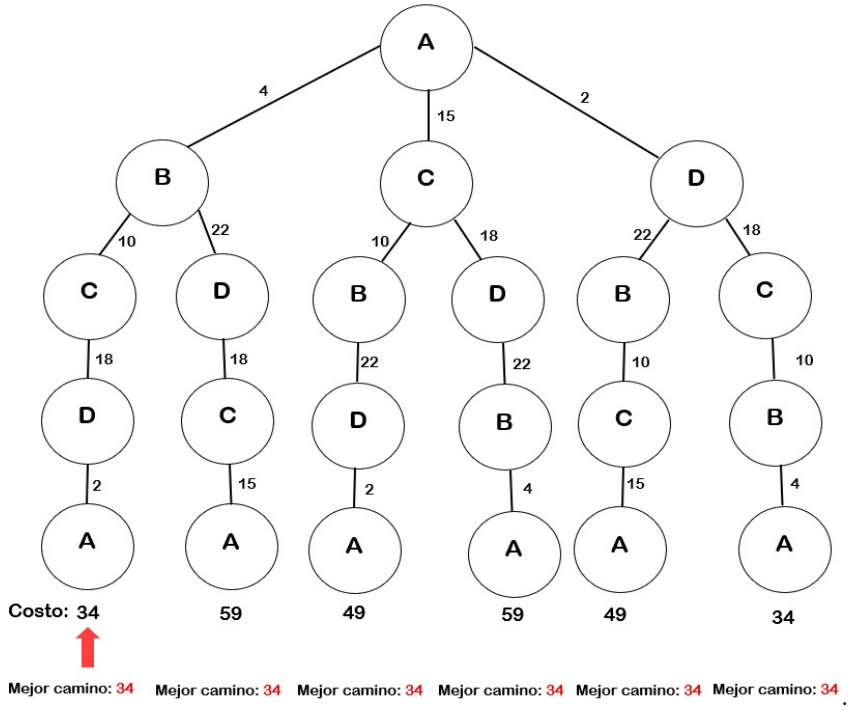


*Figura 2: Representación de un arbol, cada nodo representa una ciudad [A,B,C,D] y cada rama tiene el valor del costo por recorrer tal camino [2,15,9,18,10,8] (Punto de partida A)*

Caminos Posibles:
Punto de inicio: A

A B C D A
A B D C A
A C B D A
A C D B A
A D C B A
A D B C A

Cuyos costos a recorrer son:
A B C D A = 34
A B D C A = 59
A C B D A = 49 (Adicionando los caminos inversos)

Actualizando cada vez que llega al final, la mejor solución es 34.

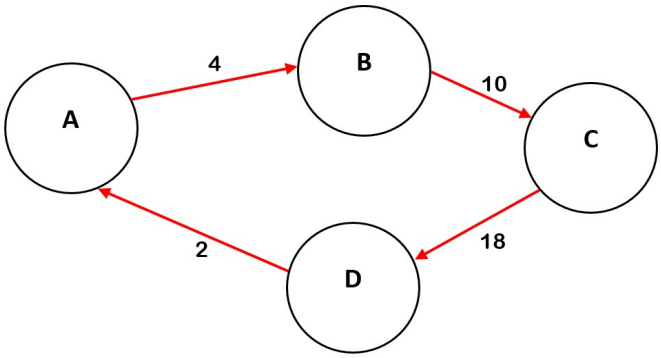


*Figura 3: Representación de la solución obtenida por el algortimo de Backtracking(A B C D A), mediante un grafo dirigido*

## 3.2 Solución y Algoritmos propuestos

```
// Busca el camino hamiltoniano
// I: nodo actual, visitados
// O: 1 se encontró ciclo, 0 no se encontró ciclo
int Hamiltonian :: buscar_ciclo(int nodo, int visitados){
  if(visitados == this->n && es_ciclo()){
    return 1;
  }
  for(int i = 0; i < n; i++){
    if(this->g->get_adj()[nodo][i] && !this->visitado[i]){
      this->visitado[i] = 1;
      this->recorrido[visitados] = i;
      if(buscar_ciclo(i,visitados+1)){
        return 1;
      }
      else{
        this->visitado[i] = 0;
      }
    }
  }
  return 0;
}
```

Para esta función se utiliza recursividad, invocando a todos los nodos que se deben visitar y al número de recorridos que indica si se logró o no recorrer todos los nodos.

### 3.2.1 Condiciones de salida

Para que esta función tenga éxito, se deben recorrer todos los nodos y además debe ser un ciclo, esta condición se logra satisfacer si la función creada más abajo esCiclo retorna 1.

### 3.2.2 Función que busca un ciclo Hamiltoniano

```
// Obtiene el ciclo hamiltoniano
// I: void
// O: void
int Hamiltonian :: obtener_ciclo(){
  int inicial = 0;
  for(int i = 0; i < n;i++){
    this->visitado[i] = 0;
    this->recorrido[i] = -1;
  }
  recorrido[0] = inicial;
  visitado[0] = 1;
  if(!buscar_ciclo(inicial,1)){
    cout << "No hay ciclo hamiltoniano" << endl;
    return 0;
  }else{
    cout << "Encontre Ciclo Hamiltoniano " << endl;
    this->recorrer_ciclo();  // Se imprime el ciclo hamiltoniano
    return 1;
  }
}
```

Para esta función se utilizan 2 arrays:

- 1 para almacenar los visitados.
- 1 para almacenar el recorrido del camino.

Primero se llena el primer arreglo de 0's y el segundo de -1's se setea el nodo inicial en recorrido y visitado como 1, despues se invoca a la función recursiva "buscarCiclo(nodoInicial, nro de visitados)"

### 3.2.3 Función que determina si el camino es un ciclo

```
// Determina si el camino hamiltoniano es un ciclo
// I: void
// O: entero que indica si es ciclo o no 1 | 0
int Hamiltonian :: es_ciclo(){
  return this->g->get_adj()[this->recorrido[0]][this->recorrido[this->n-1]];
}
```

# 4 Aprendizaje y Conclusiones

En algún momento de nuestra vida, nos enfrentaremos a este problema y curiosamente la primera solución que a una persona común se le ocurriría sería recorrer el camino más cercano a su punto de destino o el de menor costo primero, antes de evaluar un posible recorrido, y curiosamente, este método pertenece a un recorrido por el algoritmo **Greedy**, el cual no encuentra siempre el camino óptimo, por lo tanto es importante recalcar, que es conveniente conocer los métodos que se podrían utilizar antes de realizar un recorrido y el diseño de algortimos con otros métodos más eficientes cómo el **BackTracking con o sin poda**, podemos encontrar soluciones óptimas. Finalmente concluyo que este problema tiene mejores alternativas de solución ya que para un mayor números de nodos el costo o complejidad **Big O notation** es mayor y puede ser ineficiente, pero para los conocimientos abarcados en este curso, logramos mostrar alternativas de solución válidas al presente problema.

# 5 Fuentes

Definición Formal del problema:

https://en.wikipedia.org/wiki/Travelling_salesman_problem
https://www.gestiondeoperaciones.net/programacion-entera/solucion-del-problema-del-vendedor-viaj
https://es.wikipedia.org/wiki/Problema_del_camino_m%C3%A1s_corto

Solución con BackTracking y Greedy:

https://www.youtube.com/watch?v=W16E5qzzCxU
https://www.youtube.com/watch?v=EutHYzkSo5Y
https://www.youtube.com/watch?v=XaXsJJh-Q5Y
https://bit.ly/3xqEI3f/

Conceptos importantes:

https://es.wikipedia.org/wiki/Camino_hamiltoniano
https://es.wikipedia.org/wiki/Optimizaci%C3%B3n_combinatoria
https://es.wikipedia.org/wiki/Grafo_ponderado

Código:
Encontrar el mínimo por descompocisión de una ayudantía:
https://bit.ly/3rz5nZe