Trabajo Práctico Especial

Programación 3

Grupo 21

Integrantes:

* Tomas Quinteros
* Lautaro Barraza

Profesores:

* Federico Casanova
* Federico Améndola
* Sebastián Vallejos
* Soledad Merino

# Índice:

[Índice: 2](#_i4m9fshzmy5x)

[Introducción 3](#_u7ojuqswytqk)

[Enunciado 4](#_67q6hkedwyp2)

[Problemática: 4](#_2tje2po6q30f)

[Objetivo: 4](#_xbq4nlmir9pp)

[Implementación: 4](#_55sylkd6vs6)

[Algoritmo Backtracking: 5](#_jm7ps4md4ty)

[Determinación de la estrategia de backtracking a seguir: 5](#_llix0kem49i1)

[Diseño de la estrategia: 5](#_4knaajspxbbi)

[Implementación del algoritmo: 7](#_k4lnr1uwxp90)

[Coste Computacional del algoritmo: 7](#_8vst5azb6cdh)

[Métrica para medir el coste de obtener dicha solución: 8](#_hcfekmppi6ju)

[Algoritmo Greedy: 9](#_bognk35dj2qj)

[Determinación de la estrategia de Greedy a seguir: 9](#_7xlub297mc0n)

[Diseño de la estrategia: 9](#_gav7leczuf3h)

[Implementación del algoritmo: 11](#_n7blewgto7hd)

[Coste Computacional del algoritmo: 11](#_kr54fjx3e7u3)

[Métrica para medir el coste de obtener dicha solución: 12](#_c8rqzsdo43yl)

[Comparativa de algoritmos 13](#_glca2ldukg7t)

[DataSet 1: 13](#_vchcld8ik29g)

[DataSet 2: 13](#_uu3r94ilfs5v)

[DataSet 3: 13](#_2ipap1nhivfm)

[DataSet 4: 14](#_ch5062z87ql)

[DataSet 5: 14](#_zgn0ljz71iic)

[DataSet 6: 14](#_y2tpla1m52n3)

[DataSet 7: 15](#_hul2dtmecb5k)

[Conclusión 16](#_gwayllbw1faq)

## **Introducción**

Existen diversas estrategias algorítmicas para hallar soluciones eficientes y satisfactorias a un problema. Dos enfoques populares son el algoritmo de backtracking y el algoritmo de greedy. En el informe se mostrará una problemática a resolver y luego se explorará la aplicación de ambos enfoques en la resolución de la misma, para luego comparar los resultados y eficiencias de los mismos y llegar a una conclusión.

A continuación se detalla la problemática a desarrollar, luego las implementaciones de los enfoques, las comparativas entre ambos y una conclusión final.

## **Enunciado**

### **Problemática:**

Las autoridades de una ciudad deciden construir una red de subterráneos para resolver los constantes problemas de tráfico. La ciudad ya cuenta con N estaciones construidas, pero todavía no tienen ningún túnel que conecte ningún par de estaciones entre sí. La red de subterráneos que se construya debe incluir a todas las estaciones (es decir, que de cualquier estación H pueda llegar a cualquier otra estación J, ya sea de manera directa o atravesando otras estaciones). Sin embargo, debido al acotado presupuesto, las autoridades desean construir la menor cantidad de metros de túnel posibles. Para esto han calculado cuantos metros de túnel serían necesarios para conectar de manera directa cada par de estaciones existentes.

### **Objetivo:**

El objetivo de esta segunda parte del trabajo será resolver el problema planteado mediante dos técnicas algorítmicas distintas: Backtracking y Greedy. Luego se deberán comparar los resultados teniendo en cuenta distintas métricas que permitan visualizar, mínimamente, la calidad de la solución y el costo de obtener dicha solución, con ambas técnicas.

### **Implementación:**

La aplicación comenzará obteniendo la información de las estaciones y las distancias entre ellas de un archivo de texto.

Una vez llevada a memoria la información de las estaciones y sus distancias, la aplicación deberá resolver el problema planteado mediante ambas técnicas. La solución deberá ser mostrada por consola.

## **Algoritmo Backtracking:**

En primer lugar se abordará el enfoque de backtracking, que es una técnica basada en la exploración exhaustiva de todas las posibles soluciones. Este enfoque busca una solución paso a paso, evaluando sistemáticamente todas las combinaciones posibles y retrocediendo cuando se alcanza un punto muerto o condición de corte

Para resolver la problemática a través del algoritmo backtracking se siguieron los siguientes pasos:

#### **Determinación de la estrategia de backtracking a seguir:**

Sabiendo que dado un problema el algoritmo backtracking busca y genera todas las posibilidades y la solución la va armando de manera progresiva se determinó la siguiente estrategia:

Dado que se cuenta con las estaciones y las distancias entre ellas, el objetivo/estrategia del backtracking es hallar todas las posibles combinaciones de túneles que se pueden armar entre estas estaciones, y así ir armando un estado hasta que no haya más túneles a construir y ahí se corrobora si ese estado es la mejor solución hasta el momento.

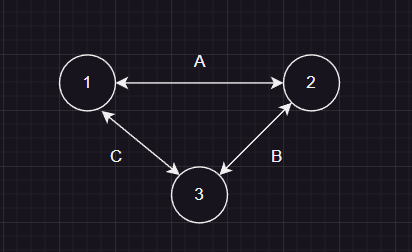
Este estado se irá armando con las posibilidades en cada conexión de construir un túnel o no, quedando un árbol binario.

#### **Diseño de la estrategia:**

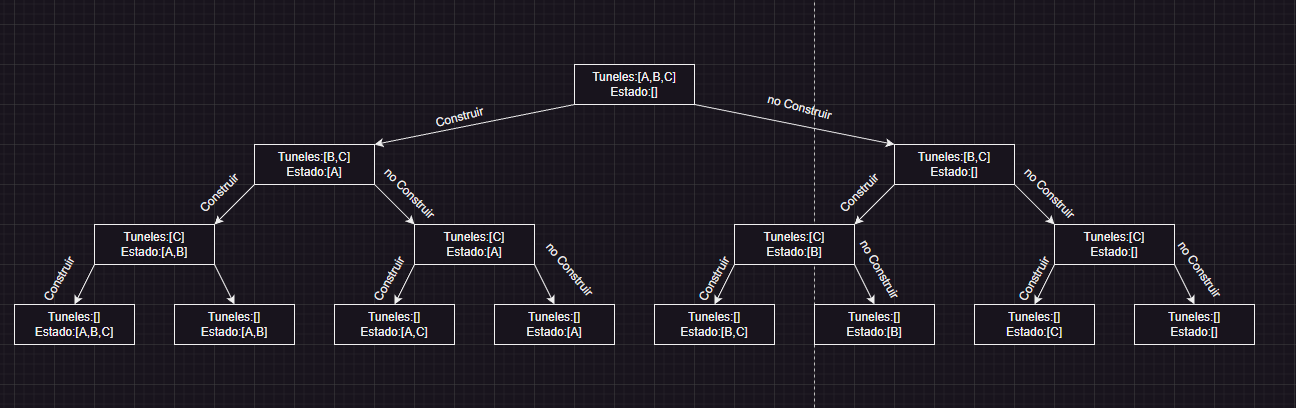
Una vez determinada la estrategia se empezó la parte de diseño, es decir armar un bosquejo del árbol de exploración de backtracking con un ejemplo. En este caso se tomó como ejemplo un caso donde haya 3 estaciones {1,2,3} con las siguientes conexiones asignadas a una letra:

* Conexión A= 1-2 50km
* Conexión B= 2-3 20km
* Conexión C=1-3-10km

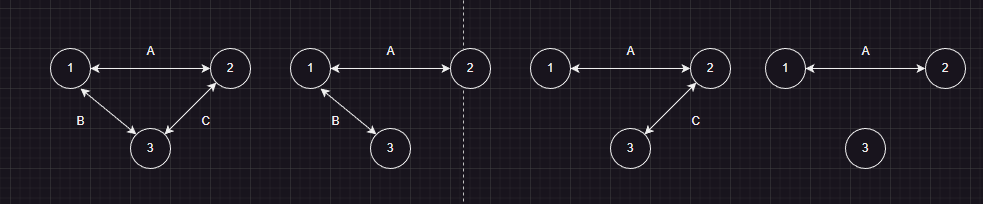
Con estos datos se diseñó un grafo para tener una idea general de los estados y así observar las soluciones:

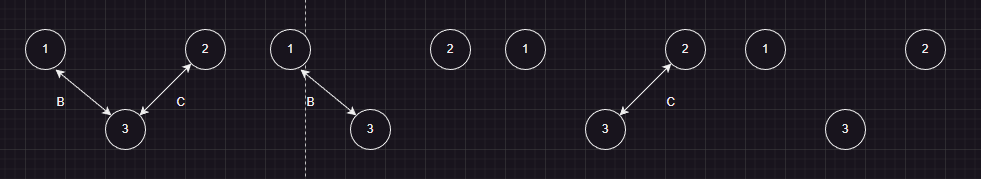


Para diseñar el árbol de exploración se tomó en cuenta las decisiones de la estrategia planteada de por cada conexión construir el túnel o no construirlo (agregar arco al estado o no agregarlo) y en cada nivel ir quitando una conexión hasta que no quede ninguna y ahí se determinan las hojas, es decir, las posibles soluciones.



Las posibles soluciones en grafos quedaron:





#### **Implementación del algoritmo:**

Basándose en el árbol de exploración obtenido se planteó el siguiente código para el backtrackin:

linkedList<Arco> tuneles;

linkedList<Arco> solucion;

int mejorSuma;

Grafo conexiones;

Grafo solucionParcial;

public void Backtraking(Grafo solucionParcial){

if(not tuneles.empty()){

if(esMejorSolucionActual(solucionParcial){

agregarSolucion(solucionParcial);

}

}else{

//opcion de no construir tunel

Arco<T> a= tuneles.removeFirst();

this.getTunelesBackTraking(solucionParcial);

tuneles.addFirst(a);

//opcion de construir

a= tuneles.removeFirst();

solucionParcial.agregarArco(a.origen, a.destino, a.getEtiqueta());

this.getTunelesBackTraking(solucionParcial);

solucionParcial.borrarArco(a.origen, a.destino);

tuneles.addFirst(a);

}

}

#### **Coste Computacional del algoritmo:**

Dado que el árbol de exploración del backtracking es binario es decir cada nodo padre siempre tiene dos hijos, una por cada posición posible, en este caso el de construir túnel o no, la complejidad computacional del algoritmo será 2 elevado a n, donde 2 son los hijos y n es el tamaño de la entrada, en este caso, la cantidad de conexiones existentes.

En el ejemplo anterior la complejidad computacional sería de 2^3=8, dado que es un árbol de exploración binario y la cantidad de conexiones entre las 3 estaciones es 3.

En conclusión el costo computacional temporal expresado en notación big O sera de O(2^n) donde n es la cantidad de conexiones.

#### **Métrica para medir el coste de obtener dicha solución:**

La métrica que se tomó para medir el costo es la de contar la cantidad de iteraciones en donde se llama recursivamente al backtracking, es decir cuando el estado actual no es posible solucion por lo tanto sigue explorando el árbol

## **Algoritmo Greedy:**

En segundo lugar se llevará a cabo el enfoque greedy, este se caracteriza por tomar la localmente óptima en el momento, con la esperanza de alcanzar una solución global óptima. A diferencia del enfoque de backtracking, el algoritmo greedy no realiza una exploración exhaustiva, sino que se guía por criterios heurísticos para tomar decisiones en cada etapa.

En este caso el criterio heurístico que se utiliza es obtener la conexión con menor distancia con la esperanza de que la solución quede con la menor cantidad de kms construidos.

#### **Determinación de la estrategia de Greedy a seguir:**

Sabiendo que dado un problema el algoritmo Greedy siempre toma la decisión que parece mejor en ese momento, con la esperanza de construir una solución globalmente óptima se determinó la siguiente estrategia:

Dado que se cuenta con las estaciones y las distancias entre ellas, el objetivo/estrategia del greedy es ir obteniendo la mejores conexiones(la de menor distancia), verificar si es factible construirla, es decir, si las estaciones de la conexión no están ya conectadas directa/indirectamente y agregarla a un conjunto solución y así de esta manera ir llevando una solución localmente óptima.

#### **Diseño de la estrategia:**

Para diseñar la estrategia anteriormente mencionada se siguió en general el pseudo-código de greedy y se planteó la idea de tener un conjunto de candidatos, en donde se almacenan todas las conexiones posibles a construir, y ir obteniendo el mejor de esos candidatos y verificar si es factible agregarlo a la solución, en caso de que lo sea se agrega y así se va armando la misma.

Esto se diagramó con un ejemplo de 4 estaciones todas conectadas y se siguió un paso a paso para poder dejar en claro la idea general y así empezar con la implementación

Primero se planteó un conjunto solucion vacío donde no hay ninguna conexión, por lo tanto la solución actual estaría de la siguiente forma:



Con el conjunto candidatos con todas las posibles conexiones:

E1;E2;15

E1;E3;20

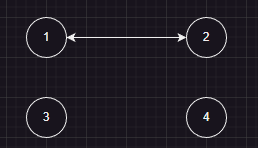
E1;E4;30

E2;E3;15

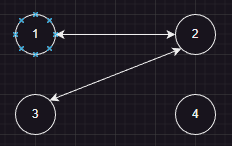
E2;E4;25

E3;E4;50

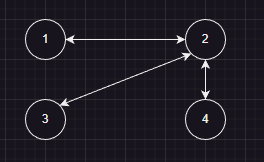
Entonces la idea del algoritmo greedy es obtener de este conjuntos candidatos, el mejor actual, es decir el de menor distancia, en este caso el E1-E2, y en caso de poder construirse(en este caso si) se agrega al conjunto solución dejandolo asi:



Luego se elije el siguiente candidato, el cual es E2-E3:



Luego el siguiente, el cual es E2-E4:



A partir de ahí, como todas las estaciones ya están conectadas los siguientes candidatos serán descartados.

Una vez planteado el funcionamiento general del algoritmo se empezó con la implementación del mismo.

#### **Implementación del algoritmo:**

Para la implementación del algoritmo se planteó un método público del servicio el cual inicializa los candidatos a construir ya ordenados según el criterio anteriormente mencionado, inicializa las estaciones y las almacena en un hashmap donde se ira realizando la tecnica union Find para poder corroborar si es posible solucion, y retorna lo que el algoritmo greedy haya encontrado.

El algoritmo greedy se planteó en una función privada, la cual recorre y va extrayendo candidato a candidato, por cada iteración va buscando si ese candidato es factible para agregar a la solución, es decir, si esa conexión no une dos estaciones ya conectadas de manera directa o indirectamente (esto lo hace mediante la técnica union find). En caso de ser factible agregarla, entonces la agrega y une los subconjuntos de estaciones, dejando así un registro de que esas estaciones fueron conectadas.

La implementación con un pseudo-java quedaría así:

public LinkedList<Arco<T>> TunelesGreedy() {

guardarArcosOrdenadoPorCriterio();

guardarVertices(); //guarda los vértices para el union find

return greedy();

}

private void greddy() {

while(!arcos.estaVacia && !haySolucion) {

Arco u = seleccionarArco();

arcos.remove(u);

if(esFactible()) {

union(estacion1,estacion2);

solucion.add(u);

}

}

}

#### **Coste Computacional del algoritmo:**

El coste computacional en este caso va a estar dado por el ordenamiento de los candidatos según el criterio greedy, ya que si se analiza el costo del metodo publico tunelesGreedy el cual será ejecutado obtenemos los siguientes costos:

* metodo guardarArcosPorCriterio
  + O(N) donde n la cantidad de arcos que recorre para guardar
  + O(n log N) con n como arcos, dado que ese es el costo del algoritmo de collections.sort
* Metodo guardarVertices
  + O(N) con n como vértices, ya que recorre todos los vértices y los guarda
* Método Greedy
  + O(N\*N) donde N son los arcos y M los vértices, ya que hay un loop donde en el pero de los casos se recorre todos los arcos y por cada iteración hay métodos con complejidad constante excepto el método find que tiene una complejidad O(M) donde como peor caso recorre todos los vértices.

Entonces en conclusión el método público tunelesGreedy tendrá una complejidad de O(NlogN) + O(N)+O(N\*M) quedando el peor caso es decir O(N logN).

En conclusión la complejidad Temporal medida en la notación Big O será de O(N logN).

#### **Métrica para medir el coste de obtener dicha solución:**

Greedy al ser un algoritmo iterativo se decidió llevar una métrica de iteraciones la cual incrementa cada vez que se selecciona una conexión de la lista de candidatos.

## **Comparativa de algoritmos**

Para realizar las comparativas se armaron dataset con diferentes tamaños de entrada para así poder analizar las métricas y soluciones de ambos algoritmos. Los resultados arrojados fueron los siguientes:

#### **DataSet 1**:

Tamaño entrada: 6 conexiones.

Estaciones: 4.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 64 | 4 |
| Resultado | [1-2, 2-3, 2-4] | [1-2, 2-3, 2-4] |
| Suma Total | 55 kms | 55 kms |
| ¿Obtuvo mejor resultado? | si | si |

#### **DataSet 2**:

Tamaño entrada: 10 conexiones.

Estaciones: 5.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 1024 | 5 |
| Resultado | [1-2, 2-5, 3-5, 3-4] | [3-4, 1-2, 2-5, 1-3] |
| Suma Total | 125 kms | 125 kms |
| ¿Obtuvo mejor resultado? | si | si |

#### **DataSet 3:**

Tamaño entrada: 15 conexiones.

Estaciones: 6.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 32768 | 8 |
| Resultado | [1-2, 2-5, 3-5, 3-4, 5-6] | [3-4, 5-6, 1-2, 2-5, 1-3] |
| Suma Total | 135 kms | 135 kms |
| ¿Obtuvo mejor resultado? | si | si |

#### **DataSet 4:**

Tamaño entrada: 21 conexiones.

Estaciones: 7.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 2097152 | 6 |
| Resultado | [1-3, 2-3, 2-5, 4-5, 5-6, 6-7] | [2-3, 2-5, 1-3, 5-6, 6-7, 3-4] |
| Suma Total | 230 kms | 230 kms |
| ¿Obtuvo mejor resultado? | si | si |

#### **DataSet 5:**

Tamaño entrada: 28 conexiones.

Estaciones: 8.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 268435456 | 7 |
| Resultado | [1-3, 2-3, 2-5, 2-8, 4-8, 5-6, 6-7] | [2-3, 2-5, 1-3, 2-8, 4-8, 5-6, 6-7] |
| Suma Total | 260 kms | 260 kms |
| ¿Obtuvo mejor resultado? | si | si |

#### **DataSet 6:**

Tamaño entrada: 36 conexiones.

Estaciones: 9.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 2^36 | 9 |
| Resultado | no se corroboro dado la alta iteraciones que debe realizar el algoritmo | [2-3, 2-5, 1-3, 2-8, 4-8, 5-6, 6-7, 1-9] |
| Suma Total | no se corroboro dado la alta iteraciones que debe realizar el algoritmo | 310 kms |
| ¿Obtuvo mejor resultado? | no se corroboro dado la alta iteraciones que debe realizar el algoritmo | no se ejecutó Backtracking, por lo tanto no se corroboró |

#### **DataSet 7**:

Tamaño entrada: 78 conexiones.

Estaciones: 13.

|  | Backtracking | Greedy |
| --- | --- | --- |
| Iteraciones | 2^78 | 25 |
| Resultado | no se corroboro dado la alta iteraciones que debe realizar el algoritmo | [9-13, 2-3, 2-5, 5-11, 7-11, 8-11, 1-3, 2-13, 4-8, 5-6, 8-10, 6-12] |
| Suma Total | no se corroboro dado la alta iteraciones que debe realizar el algoritmo | 440 kms |
| ¿Obtuvo mejor resultado? | no se corroboro dado la alta iteraciones que debe realizar el algoritmo | no se ejecutó Backtracking, por lo tanto no se pudo corroborar |

## **Conclusión**

En base a los análisis realizados sobre las implementaciones de ambos enfoques sobre el mismo problema planteado, se llegó a la conclusión de que en esta problemática el enfoque Greedy es más eficiente que el backtracking ya que:

* El enfoque Greedy fue capaz de encontrar soluciones de alta calidad de manera más rápida y con menos recursos computacionales en comparación con el enfoque Backtracking.
* Se demostró que el enfoque Greedy siempre proporciona el mejor resultado en este caso particular donde existen conexiones entre todas las estaciones. Esto significa que las soluciones generadas por el algoritmo Greedy son óptimas o muy cercanas a la óptima, garantizando así la calidad de la solución obtenida.
* El enfoque Greedy tuvo un menor costo computacional en comparación con el enfoque Backtracking. Esto implica que el algoritmo Greedy fue más rápido y consume menos recursos de procesamiento, lo cual es una ventaja significativa en términos de eficiencia y escalabilidad del algoritmo.

En términos generales, estas conclusiones sugieren que el enfoque Greedy es la opción más óptima para abordar la problemática del enunciado, sin embargo si se agregaran mas condiciones como por ejemplo conexiones dirigidas o estaciones sin conectar puede ser que greedy no brinde la mejor solución y backtracking sea un mejor enfoque.