Problema del viajante

Solución Greedy:

Criterio greedy: Se agrega la ciudad adyacente más cercana aún no visitada a la solución.
Próximo candidato: La ciudad más cercana a la ciudad actual que no haya sido visitada.
En este caso, puede que la solución que se va construyendo no llegue a ser solución, ya que dependiendo del grafo, puede que el algoritmo se vaya "encerrando" y queden ciudades sin visitar, lo que hace que no se acepte como solución.

Conclusión: Como se puede observar en el gráfico realizado para el inciso 3, los tiempos de este algoritmo son extremadamente acotados comparados con los del backtracking, entonces para el caso que se tenga un grafo con muchas ciudades este tipo de algoritmo resulta mucho más viable.

Solución Backtracking:

Estado: El estado está formado por la ciudad actual, la ciudad origen (desde donde se empezó) y la solución que se va construyendo. Esta solución está conformada por la colección de ciudades visitadas hasta entonces y la distancia recorrida.

Estado final: Es estado final cuando la cantidad de ciudades recorridas es igual al total de ciudades y desde allí se puede volver al origen.

Estado solución: Es solución todo estado final. En este caso, todo estado final que tenga menor distancia se va guardando para retornar la mejor solución.

Hijos: Los hijos de cada estado representan un avance hacia otra ciudad en el grafo y su correspondiente inserción en la solución parcial. Son hijos todas las combinaciones de ciudades aún no visitadas a las que se pueden acceder, es decir, los estados intermedios y finales del árbol.

Podas: Cuando la solución parcial que se va generando tiene más distancia que una solución ya encontrada, entonces el algoritmo no sigue.

A continuación adjunto resultado de la poda:

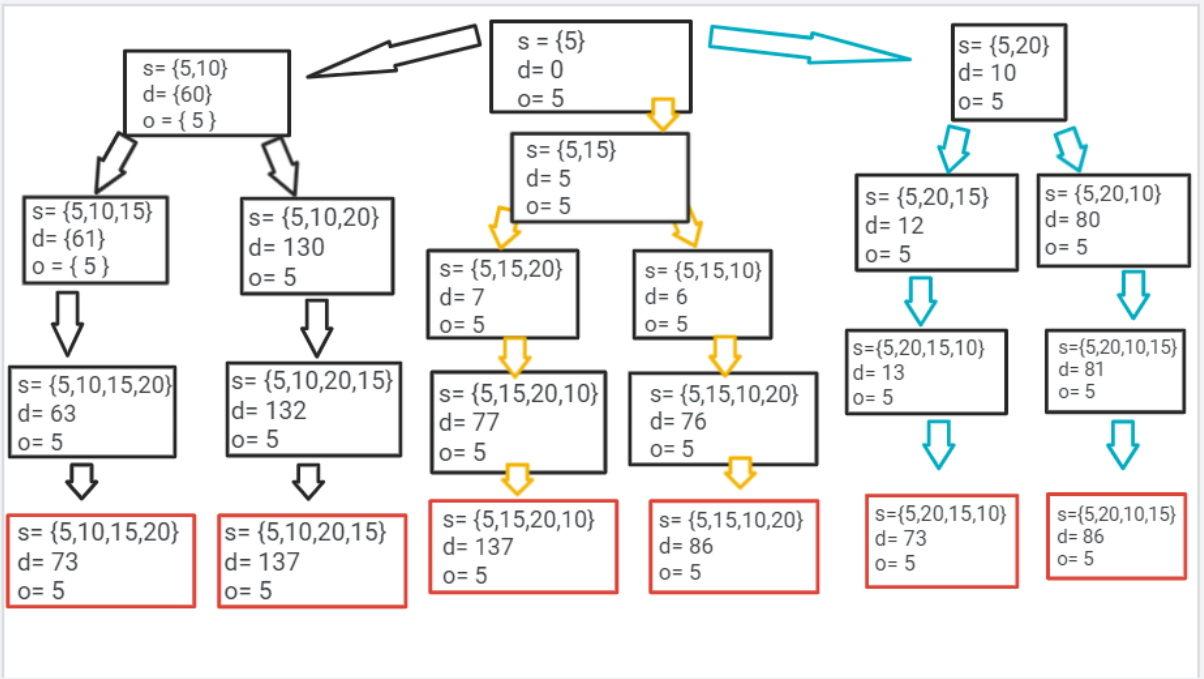
Sin poda:

```
Console  Problems  Debug Shell
<terminated> VisorCiudad [Java Application] C:\Program Files\Java\jre1.8.0_301\bi
 BackTracking
 km total: 82 Camino: {5=Nombre, 10=Pehuajo, 30=Azul, 35=Juare
Iteraciones: 368
```

Con poda:

```
Console  Problems  Debug Shell
<terminated> VisorCiudad [Java Application] C:\Program Files\Java\jre1.8.0_301\bi
 BackTracking
 km total: 82 Camino: {5=Nombre, 10=Pehuajo, 30=Azul, 35=Juare
Iteraciones: 136
```

Árbol de soluciones:



Link árbol de soluciones:
https://jamboard.google.com/d/1GH23tTnOgkW8U3CPRdrfZapK8SSITnbffk3ypRG4GQ0/edit?usp=sharing
Este árbol corresponde al grafo TEST 1 del template TEST.docx.

Cada rama que se genera en los estados corresponde a los adyacentes de la ciudad actual, se agrega la ciudad al conjunto solución y se calcula la distancia hasta el momento.
Lo que está pintado de rojo son estados finales, a los que ya se le sumó la distancia al origen. Aquel estado final que posea la menor distancia será la solución.

Conclusión: Si bien este algoritmo devuelve la mejor solución, en caso de que el grafo tenga muchas ciudades sería casi inviable la ejecución, ya que como se observa en el gráfico, a medida que la cantidad de ciudades en el grafo aumenta, el tiempo de resolución crece exorbitantemente porque tiene una complejidad factorial.
Se debería usar esta clase de algoritmos en casos donde se necesite precisión a sabiendas de pagar el costo del tiempo.