Trabajo Práctico Especial
Segunda parte

Programación III

Acosta Jose

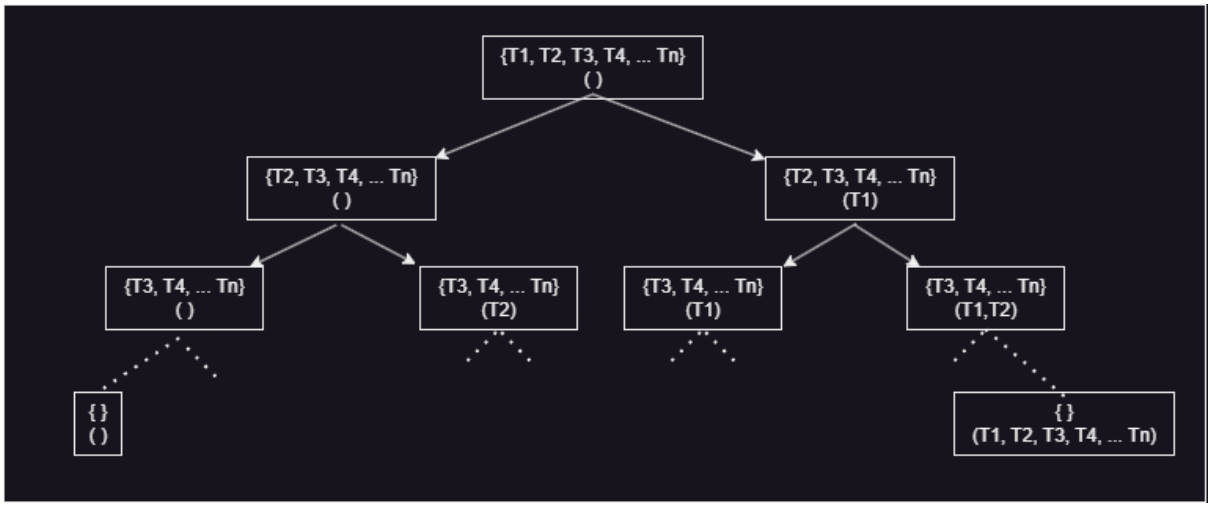
Facultad de ciencias exactas

UNICEN

Backtracking

Estrategia

La estrategia Backtracking que se llevó a cabo en este trabajo fue partir de un conjunto de todos los túneles posibles (los del dataset) e ir tomando o no cada túnel.



En esta estrategia se definen:

- **Estado**: Túneles disponibles, túneles tomados en la solución actual y kilómetros totales de la solución actual.
- **Decisión tomada en cada paso**: La decisión que se toma en cada paso es **tomar** o **no tomar** el túnel que se quita de los disponibles.
- **Estado final**: Es estado final si no hay más túneles disponibles.

Costo computacional

En esta sección **t** se refiere a la cantidad total de túneles del dataset y **e** a la cantidad de estaciones implicadas en esos túneles.

La primera operación que se realiza en esta solución es transformar los túneles de un Iterator a una LinkedList, paso necesario para la estrategia Backtracking utilizada. Esta operación implica utilizar el método obtenerArcos() del grafo dirigido implementado (complejidad O(e * t)), y luego un recorrido de los túneles (O(t)), donde en cada iteración se agrega el mismo a una LinkedList (complejidad constante).

En resumen este paso tiene una complejidad de **O(e * t + t)**.

Luego viene el algoritmo Backtracking. En este la condición de corte y la poda son constantes y hay principalmente 3 complejidades:

1. Estrategia: El costo computacional asociado a la estrategia adoptada en esta solución es 2^t, ya que por cada túnel se pueden tomar 2 decisiones: tomarlo o no tomarlo.
2. Verificar si es solución: una vez que se llega a un estado final, se realizan 3 operaciones para verificar si es una solución válida:
    a. Se transforma la solución a un grafo no dirigido: Para esto se recorren todos los túneles de la solución. En el peor de los casos (si la solución toma todos los túneles disponibles) este paso tiene un costo O(t).
    b. Se realiza un recorrido DFS: Esto se hace para luego verificar si el grafo no dirigido generado en el punto anterior es conexo. En este paso, en el peor de los casos habría que recorrer todas las estaciones implicadas en todos los túneles del dataset, por lo que tiene una complejidad O(e).
    c. Se verifica que el grafo solución sea conexo: En este punto se recorren todas las estaciones implicadas en todos los túneles del dataset, y se verifica que todas estén en el recorrido DFS realizado en el paso anterior. Otra vez, en el peor de los casos complejidad O(e).
3. Sobreescribir mejor solución: Si se llega a una mejor solución, al guardarla se utiliza el método clone() de LinkedList, añadiendo (en el peor de los casos) una complejidad O(t).

En resumen, el costo computacional del algoritmo Backtracking es **O(** 2^t *** (2t + 2e)).**

Por último escribir el retorno del método implica un recorrido de los túneles implicados en la solución. **O(t)** en el peor de los casos.

Dicho esto, el costo computacional total de la solución Backtracking implementada es:

O(e * t + t + 2^t * (2t + 2e) + t), simplificando: O(2^t * (t+e) + e*t + t)

Greedy

Estrategia

La estrategia Greedy llevada a cabo en este trabajo para resolver el problema fue ordenar los túneles de menor cantidad de kilómetros a mayor cantidad de kilómetros. Una vez hecho esto se iteran y se agregan a la solución sólo si es factible.

Factibilidad: Para verificar si es factible tomar un túnel o no, se deben cumplir 2 condiciones:

1. Que **alguna** de las estaciones involucradas en el túnel que se está verificando **ya esté** en la solución actual. Esto significa que este túnel es alcanzable de alguna manera.
2. Que **no** estén en la solución actual **las 2** estaciones involucradas en el túnel que se está verificando. Esto implicaría construir 2 túneles distintos que conectan las 2 mismas estaciones.

Costo computacional

En esta sección **t** se refiere a la cantidad total de túneles del dataset y **e** a la cantidad de estaciones implicadas en esos túneles.

La primera operación que se realiza en esta solución es transformar los túneles de un iterator a una LinkedList, para luego poder ordenarlos. Esta operación implica utilizar el método obtenerArcos() del grafo dirigido implementado (complejidad O(e * t)), y luego un recorrido de los túneles (O(t)), donde en cada iteración se agrega el mismo a una LinkedList (complejidad constante).
En resumen este paso tiene una complejidad de O(e*t + t).

Luego viene el algoritmo Greedy, el cual tiene principalmente 3 complejidades:

1. El ordenamiento de los túneles: se hace mediante el método sort de List. Este método utiliza un algoritmo mergesort, por lo que implica un costo computacional O(t log t).
2. La iteración de los túneles: Una vez ordenados se recorren, dando en el peor de los casos una complejidad O(t). Además dentro de esta iteración, en cada paso se verifica que no se haya llegado ya a una solución:
    - Verificación: Esta validación implica obtener las estaciones del grafo (constante en la implementación del grafo dirigido), recorrerlos (O(e)) y chequear si todos están en la solución (constante, ya que se trata de un HashSet).
3. Una última validación para saber si se llegó a una solución: implica un recorrido de las estaciones, O(e).

De esta manera, el algoritmo Greedy tiene una complejidad de O(t log t + t * e + e).

Por último escribir el retorno del método implica un recorrido de los túneles implicados en la solución. O(t) en el peor de los casos.

Dicho esto, el costo computacional total de esta solución es de O(2 (e*t) + t log t + e + 2t), simplificando $O(t \log t + e*t + e + t)$.

Comparativa

Dataset 1

E1;E2;15
E1;E3;20
E1;E4;30
E2;E3;15
E2;E4;25
E3;E4;50

Cantidad de túneles del dataset: 6
Cantidad de estaciones del dataset: 4

Soluciones mostradas por consola:

```
Greedy
E1-E2,E2-E3,E2-E4,
Km totales de la solución: 55km
Costo temporal de la busqueda de solución: 1.3289 segundo/s
Cantidad de candidatos evaluados por el Greedy: 4 candidatos
```

Backtracking sin poda:

```
Backtracking
E1-E2,E2-E3,E2-E4,
Km totales de la solución: 55km
Costo temporal de la búsqueda de solución: 2.0706 segundo/s
Cantidad de estados generados por el Backtracking: 127 estados, de los cuales 64 son estados finales.
```

Backtracking con poda:

```
Backtracking
E1-E2,E2-E3,E2-E4,
Km totales de la solución: 55km
Costo temporal de la búsqueda de solución: 1.8948 segundo/s
Cantidad de estados generados por el Backtracking: 101 estados, de los cuales 42 son estados finales.
```

Tabla comparativa:

	Greedy	Backtracking sin poda	Backtracking con poda
Cantidad de túneles de la solución	3	3	3
Km totales de la solución	55 km	55 km	55 km
Costo temporal	1.3 segundos	2.1 segundos	1.9 segundos
Costo métrica	4 candidatos evaluados	127 estados generados	101 estados generados

Dataset 2

E1;E2;25
E1;E4;90
E1;E3;60
E1;E5;50
E1;E6;50
E2;E3;60
E2;E6;35
E2;E5;30
E2;E4;70
E3;E5;60
E3;E6;80
E4;E3;10
E4;E5;70
E4;E6;70
E6;E5;10

Cantidad de túneles del dataset: 15
Cantidad de estaciones del dataset: 6

Soluciones mostradas por consola:

```
Greedy
E4-E3,E1-E3,E2-E3,E3-E5,E4-E6,
Km totales de la solución: 260km
Costo temporal de la busqueda de solución: 1.2659 segundo/s
Cantidad de candidatos evaluados por el Greedy: 13 candidatos
```

Backtracking sin poda:

```
Backtracking
E1-E2,E1-E3,E2-E5,E4-E3,E6-E5,
Km totales de la solución: 135km
Costo temporal de la búsqueda de solución: 112.7272 segundo/s
Cantidad de estados generados por el Backtracking: 65535 estados, de los cuales 32768 son estados finales.
```

Backtracking con poda:

```
Backtracking
E1-E2,E1-E3,E2-E5,E4-E3,E6-E5,
Km totales de la solución: 135km
Costo temporal de la búsqueda de solución: 8.9432 segundo/s
Cantidad de estados generados por el Backtracking: 2071 estados, de los cuales 434 son estados finales.
```

Tabla comparativa:

	Greedy	Backtracking sin poda	Backtracking con poda
Cantidad de túneles de la solución	5	5	5
Km totales de la solución	260 km	135 km	135 km
Costo temporal	1.3 segundos	113 segundos	9 segundos
Costo métrica	13 candidatos evaluados	65.535 estados generados	2071 estados generados

Dataset 3

No lo escribo porque ocupa 2 páginas.

Cantidad de túneles del dataset: 78
Cantidad de estaciones del dataset: 13

Para la ejecución y comparación de este dataset no se va a utilizar el Backtracking sin poda, ya que con poda ya tarda mucho tiempo.

Soluciones mostradas por consola:

```
Greedy
E9-E13,E2-E13,E8-E13,E1-E8,E5-E8,E8-E10,E1-E4,E1-E7,E1-E12,E3-E10,E10-E11,E3-E6,
Km totales de la solución: 670km
Costo temporal de la busqueda de solución: 1.6984 segundo/s
Cantidad de candidatos evaluados por el Greedy: 46 candidatos

Backtracking
E1-E3,E2-E3,E2-E5,E2-E13,E4-E8,E5-E6,E5-E11,E6-E12,E7-E11,E8-E10,E8-E11,E9-E13,
Km totales de la solución: 440km
Costo temporal de la búsqueda de solución: 1216508.1112 segundo/s
Cantidad de estados generados por el Backtracking: 1052545535 estados, de los cuales 1040202950 son estados finales.
```

Tabla comparativa:

	Greedy	Backtracking con poda
Cantidad de túneles de la solución	12	12
Km totales de la solución	670 km	440 km
Costo temporal	1.7 segundos	20 minutos
Costo métrica	46 candidatos evaluados	1.052.545.535 estados generados

Conclusión

En este trabajo se buscó la solución a un problema sencillo de dos maneras distintas. Se plasmó lo aprendido durante la cursada puntualmente de los algoritmos de Backtracking y Greedy y se compararon los resultados obtenidos.

Como conclusión considero que ambas herramientas son muy útiles para resolver problemas comunes.

Por otro lado, está bueno implementar a nivel funcional estos algoritmos, ya que de esta manera es una experiencia bastante más enriquecedora. Por ejemplo en cuanto a costo computacional, no es lo mismo ver la diferencia entre las notaciones BigO que ejecutar los algoritmos con una entrada bastante grande y notar que uno tarda significativamente más que el otro en resolver el problema.

También deja ver claramente el factor exponencial del algoritmo de Backtracking implementado, donde por agregar pocos túneles a la entrada, resulta mucho más costoso buscar una solución.

Además haciendo este trabajo entendí la importancia de encontrar una estrategia para realizar una poda en un algoritmo Backtracking, ya que las diferencias a nivel ejecución son muy grandes.

Por último, este trabajo también me ayudó a entender la importancia de buscar algoritmos o estrategias que resuelvan problemas específicos, pero bajando considerablemente su costo computacional (como el problema de la Mochila Fraccionaria o Dijsktra), ya que por “fuerza bruta” pueden llegar a ser muy complejos de resolver, o de encontrar la mejor solución.