Problema C

Valerie Parra Cortés

December 11, 2018

1 Identificación del problema

Para la solución de este problema se utilizo un recorrido de exploración en un grafo de estados como el que se muestra en la figura 1 En este caso, en el primer estado se un estado que contiene los 4 conjuntos que representan las estimaciones de los sensores, donde por orden de prioridad (x menores luego y menores) asumimos que $1 \leq 2 \leq 3 \leq 4$. En el siguiente nivel del árbol se tiene los conjuntos de 3 elementos que tienen el menor intervalo, es decir 1, luego los que contienen al 2 y como no se pueden formar subconjuntos de 3 que contengan el 3 sin que se repitan de los anteriores, se pasa a los subconjuntos de 2 que contienen al primer elemento, luego los de a 2 que contiene al 2do elemento,luego los que contienen al 3er elemento y finalmente los subconjuntos de 1 elemento.

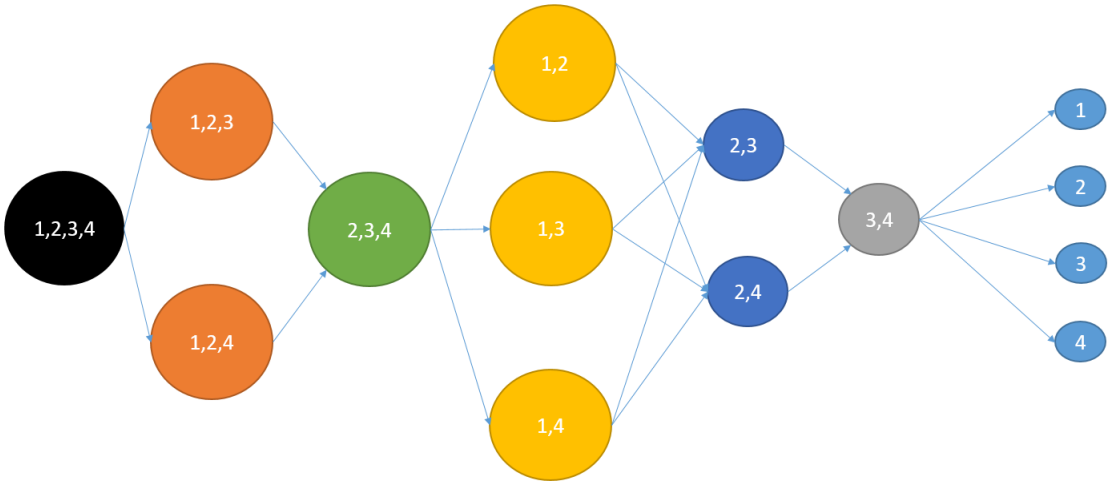

Figure 1: Grafo de exploración

Para implementar esto, se definió una clase intervalo para representar los intervalos, dicha clase implementa Comprable y el compareTo() daba prioridad

a los intervalos más cercanos al 0. Luego se definió una función de sucesores que dado un estado, sus intervalos y su intervalo más pequeño elemento encontrará ya sea o las combinaciones de intervalos con la misma cantidad de elementos pero que contengan todos el intervalo siguiente en prioridad al más pequeño de su padre o las combinaciones con un intervalo menos empezando en el intervalo más pequeño de las observaciones dadas, sí no se podían formar más estados con la misma cantidad de intervalos empezando en el intervalo siguiente a más pequeño del padre, tal como se muestra en el grafo. Luego se exploraba el grafo con el algoritmo visto en clase con ayuda de una agenda. Como optimización se implemento un HashSet que guardaba los estados recorridos para no tener que volver a procesalos, esto implica un mayor tiempo en memoria pero una ganancia significativa en tiempo.

Entrada	Descripción
n: nat	Número de observaciones
a: int[0,n)	Arreglo con los intervalos de las observaciones

Table 1: Entradas del problema

Salida	Descripción
voe:nat	Voe de los intervalos dados

Table 2: Salidas del problema

1.1 Complejidad espacial

Método main

1. Priority Queue con los intervalos: n
2. ArrayList con los intervalos: n

Total $\theta(2n + n!)$ dónde n! corresponde al llamado de feasibleSolution() que se encuentra a continuación

Método feasibleSolution()

1. Queue agenda que a lo sumo tendrá $\sum_{r=0}^{r=n} \frac{n!}{n-r}$ estados
2. HashSet que a los sumo tendrá $\left(\sum_{r=0}^{r=n} \frac{n!}{n-r}\right) * fc$, donde fc es el factor de carga, estados

Aproximando nos da $\theta(n!)$

Método sucesores()

1. ArrayList de los sucesores, un estado tendrá como máximo $n!$ sucesores
2. ArrayList de los intervalos de un estado, cómo máximo un estado tiene n intervalos

Total $\theta(n! + n)$
Método isSolution()
Este método sólo guarda un intervalo que dice la intersección de los intervalos del estado $\theta(1)$

1.2 Complejidad temporal

Método main

1. Cargar la información n
2. Llamada al método feasibleSoltuion()

Total $\theta(n + n! \cdot (n! + n))$ dónde n! corresponde al llamado de feasibleSolution() que se encuentra a continuación

Método feasibleSolution()
En el peor de los casos se recorren todas las permutaciones; $\theta(n! \cdot (n! + n))$ donde n! nos habla de recorrer cada estado, que a los sumo son n!, y el (n!+n) es lo que cuesta encontrar su sucesores y verificar si es solución
Método sucesores()
En el peor de los casos un estado tiene n! sucesores: $\theta(n!)$
Método isSolution()
En el peor de los casos se recorren todos los intervalos para ver si se intersectan $\theta(n)$

2 Comentarios finales

Este algoritmos se basa en la técnica de exploración de grafos vista en el curso.