**Minimal coverage**
**Johanna Beltran y Diego Triviño**
2012

# Índice

# 1. Introducción

'**Minimal coverage**' es un problema de programación el cual encontramos en el juez virtual UVA con el número **10020**.
Este documento busca mostrar una de las tantas soluciones desde el enfoque matemático teniendo en cuenta que el objetivo es realizar el código con la solución del problema en cualquier lenguaje de programación con la ayuda de este documento.
Este problema puede ser resuelto utilizando la metodología de algoritmos voraces.
Los algoritmos voraces tienden a ser bastante eficientes y pueden implementarse de forma relativamente sencilla. Su eficiencia se deriva de la forma en que trata los datos, llegando a alcanzar muchas veces una complejidad de orden lineal.

Se deben definir los siguientes elementos según el problema que abordan:

1. **El conjunto $C$:** de candidatos es decir las entradas del problema.
2. **Función solución $S$:** Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución (no importa si es óptima o no lo es).
3. **Función de selección:** Informa de cuál es el elemento más prometedor para completar la solución. Éste no puede haber sido escogido con anterioridad. Cada elemento es considerado una sola vez.
4. **Función de factibilidad:** determina si un conjunto es completable, es decir, si añadiendo a este conjunto nuevos candidatos es posible alcanzar una solución al problema, suponiendo que esta exista.
5. **Función objetivo:** Es aquella que queremos maximizar o minimizar, el núcleo del problema.

# 2. Definición del problema

Teniendo en cuenta varios segmentos de línea. Se debe elegir la cantidad mínima de ellos, tal que se pueda cubrir completamente el segmento dado.
Se deben tener en cuenta las siguientes restricciones para la solución del problema:

1. La longitud $M$ del eje $X$ debe ser menor a 5000.
2. Las coordenadas de cada segmento de línea $[L_i, R_i]$ deben ser menores a 5000
3. Para cada caso de prueba se pueden ingresar hasta 100000 coordenadas.

## 2.1. Entrada

La primera línea es el número de casos de prueba, seguido por una línea en blanco.
Cada caso de prueba en la entrada contiene un número entero $M$ ($1 \leq M \leq 5000$), seguido de las coordenadas en pares $[L_i, R_i]$ ($|L_i|$,$|R_i| \leq 5000$, i $\leq 100000$), cada uno en una línea separada. Cada caso de prueba de entrada es terminada por "0 0".
Cada caso de prueba estarán separados por una sola línea.

*EJEMPLO*

2

1
−1 0
−5 -3
2 5
0 0

1
−1 0
0 1
0 0

## 2.2. Salida

Para cada caso de prueba, en la primera línea de salida de su programa debe imprimir el número mínimo de segmentos de línea que pueden cubrir el segmento [0,$M$]. En las siguientes líneas, las coordenadas de segmentos, ordenados por su extremo izquierdo ($L_i$), debe ser impreso en el mismo formato que en la entrada. La entrada '0 0' no se debe imprimir. Si [0,$M$] no puede ser cubierto por los segmentos de recta dados, su programa debe imprimir '0'.
Imprimir una línea en blanco entre las salidas para dos casos de prueba consecutivos.

*EJEMPLO ANTERIOR*

0

1
0 1

3

## 3. Modelamiento matemático

'*intervalos*' y '*solucion*' son conjuntos de intervalos que se encuentran sobre el eje $X$; donde *solucion* $\subseteq$ *intervalos* y contiene todos aquellos intervalos que cubren completamente el segmento $M$.
Para este problema necesitamos un número entero '*conexión*' el cual indica el limite superior del último intervalo agregado al conjunto '*solucion*'.

1. **Conjunto de candidatos '*solucion*':** Aquellos segmentos que cubren al segmento M.
2. **Función de selección (SeleccioneIntervalo):** El siguiente intervalo $X \in$ '*intervalos*' tal que $X \notin$ '*solucion*', tiene la longitud mas larga y contiene el número conexión entre sus límites.
3. **Función solución (objetivo):** Los intervalos de '*solucion*' cubren todo el segmento $M$.

## 4. Planteamiento de la solución

Para cada caso de prueba se definen los elementos de los algoritmos voraces mencionados anteriormente.
Para empezar, con la función de selección se elige un intervalo $x$ y se adiciona al conjunto '*solucion*'. Por último, cuando no existan mas intervalos por seleccionar se verifica la función solución devolviendo así la respuesta correcta para cada caso de prueba.

### 4.1. Pseudocódigo propuesto

Funcion Voraz (conjunto *intervalos*,entero $M$)
Define conjunto *solucion*
Define entero *conexion* = 0
Define booleano *haySeleccion* = Verdad // Esta variable indica si existen elementos en el conjunto '*intervalos*' que sean aptos.
**mientras** ¬**solucion**(*intervalos*,$M$) $\wedge$ *haySeleccion*
    $x$ := **SeleccioneIntervalo** (*intervalos*,*conexion*)
    **ElimineIntervalo** (*intervalos*,$x$)
    **si** (x! = *null*)
      **entonces** *solucion* := *solucion* + $x$
                *conexion* := $x$.limiteSuperior
    **sino**
      **entonces** *haySeleccion* := falso
**si** (**Objetivo** (*solucion*,$M$))
    **entonces devolver** *solucion*
**sino**
    **entonces devolver** *no hay solucion*

## 5. Conclusiones

1. Para este problema se recomienda utilizar algoritmos voraces ya que cumple con las características de dichos algoritmos, teniendo en cuenta que en cada paso se toma una decisión de la que estamos seguros, las decisiones tomadas nunca se reconsideran y el algoritmo termina cuando no quedan decisiones por tomar.

2. En este caso específico la función de factibilidad se decidió eliminar, puesto que cualquier elemento de la función selección es factible.