Competitive Programming Workshop: Algoritmos voraces

Giovanny Alfonso Chávez Ceniceros

Universidad Autónoma de Chihuahua
Facultad de Ingeniería

gchavezceniceros@acm.org

25 de marzo de 2019

Contenido

¿Qué es algoritmo *greedy*?

- Se dice que un algoritmo es codicioso si toma la decisión óptima a nivel local en cada paso con la esperanza de llegar a la solución global óptima.
- En algunos casos, la codicia funciona: la solución es corta y se ejecuta de manera eficiente. Para muchos otros, sin embargo, no lo hace.
- Cuando una estrategia voraz falla al producir resultados óptimos en todas las entradas, en lugar de algoritmo suele denominarse heurística.

Devolviendo el cambio

Descripción

El objetivo de este problema es encontrar el número mínimo de monedas necesarias para devolver el cambio del valor de entrada (un número entero) en monedas con denominaciones 1, 5 y 10.

Entrada

Un solo entero m.

Salida

El número mínimo de monedas con denominaciones 1, 5, 10 que se devuelven m.

Devolviendo el cambio

Restricciones

$0 \leq m \leq 10^3$.

Casos de prueba

Entrada	**Salida**
2	2 $(2 = 1 + 1\ )$
28	6 $(28 = 10 + 10 + 5 + 1 + 1 + 1)$

Devolviendo el cambio

```
1  int greedyChange( int money )
2  {
3      int i = 0, change = 0, coins[3] = {10, 5, 1};
4
5      while(money > 0)
6          while(i < 3)
7              if(coins[i] > money)
8                  ++i;
9              else
10             {
11                 money -= coins[i];
12                 change += 1;
13             }
14     return change;
15 }
```

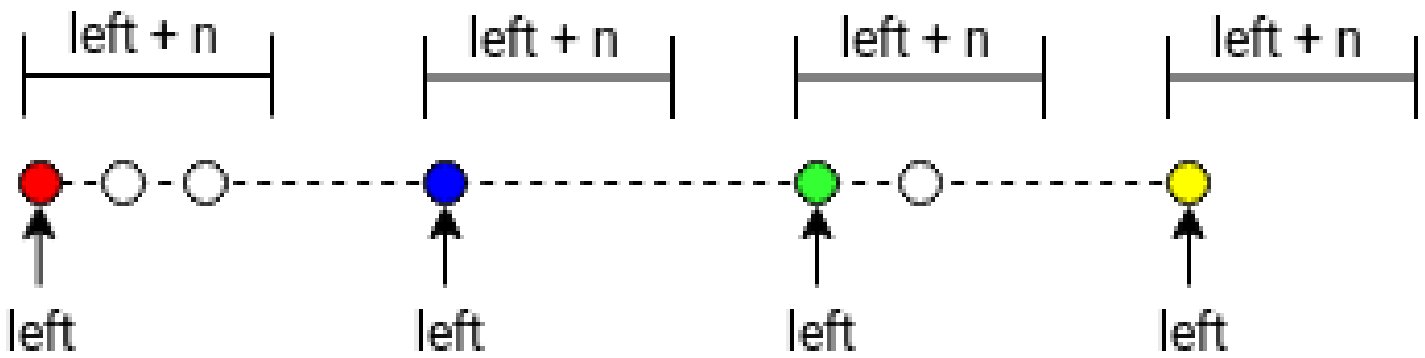
Fiesta de celebración

- Muchos niños vinieron a una fiesta de celebración. Necesitas organizarlos en grupos pero quieres que jueguen bien unos con otros, por lo que los grupos deben ser bastante homogéneos y para eso se desea que dos niños cualquiera en el mismo grupo debe diferir por lo más dos años de edad.
- Por lo tanto, se desea organizarlos en el número mínimo posible de grupos porque para cada grupo necesitas un adulto como supervisor, y quieres minimizar el número de adultos.

Fiesta de celebración

```
1  using vi  = std::vector<int>;
2  using vii = std::vector<std::pair<int, int>>;
3
4  vii pointsCoverSorted( const vi &points, int n = 2 )
5  {
6      int left = 0, l = 0, r = 0;
7      vii segments; segments.reserve(points.size( ));
8
9      while(left < points.size( ))
10     {
11         l = points[left];
12         r = points[left] + n;
13         segments.push_back(std::make_pair(l,r));
14         left += 1;
15
16         while(left < points.size( ) && points[left] <= r)
17             left += 1;
18     }
19     return segments;
20 }
```

Fiesta de celebración



Recolectando firmas

- Eres el responsable de recopilar firmas de todos los inquilinos de un edificio. Para cada inquilino, conoces el período de tiempo cuando él o ella está en casa. Te gustaría reunir todas las firmas visitando el edificio tan pocas veces como sea posible.
- El modelo matemático para este problema es el siguiente. Te dan un conjunto de segmentos en una línea y tu objetivo es marcar el menor número de puntos posible en una línea. de modo que cada segmento contenga al menos un punto marcado.

Recolectando firmas

Descripción

Dado un conjunto de n segmentos $[a_0, b_0], [a_1, b_1], ..., [a_{n1}, b_{n1}]$ con coordenadas de enteros en una línea, encuentre el número mínimo de puntos de tal manera que cada segmento contenga al menos un punto. Es decir, encuentre un conjunto de enteros X del tamaño mínimo tal que para cualquier segmento $[a_i, b_i]$ haya un punto $x \in X$ tal que $a_i \leq x \leq b_i$.

Recolectando firmas

Entrada

La primera línea contiene el número n de segmentos. Cada una de las siguientes n líneas contiene dos enteros a_i y b_i (separados por un espacio) que definen las coordenadas de los puntos finales del i-esimo segmento.

Salida

El número mínimo m de puntos y las coordenadas enteras de m puntos (separados por espacios).

Recolectando firmas

Restricciones

$1 \leq n \leq 100$ $0 \leq a_i \leq b_i \leq 10^9$ para toda $0 \leq i \leq n-1$.

Casos de prueba

Entrada	Salida
3 1 3 2 5 3 6	1 3

Recolectando firmas

Casos de prueba

Entrada	Salida
4	2 3 6
4 7	
1 3	
2 5	
5 6	

Recolectando firmas

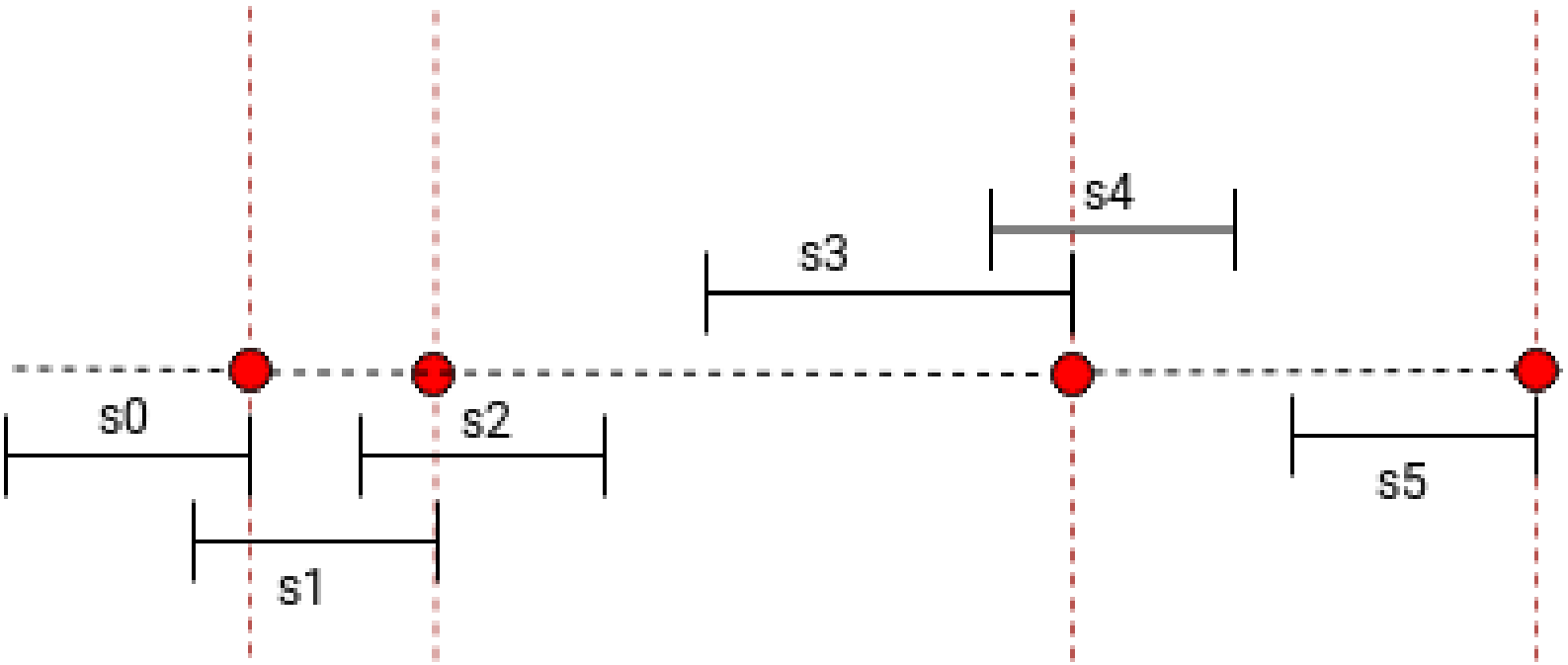
```
1  vi optimalPoints( vii &segments )
2  {
3      std::sort(segments.begin(), segments.end()
4                ,[](auto &x, auto &y) -> bool
5                 {
6                   return x.second < y.second;
7                 });
8
9      vi points; points.reserve(segments.size());
10     int point = segments[0].second;
11     points.push_back(point);
```

Recolectando firmas

continuación...

```
1       for(int i = 1; i < segments.size( ); ++i)
2           if(point<segments[i].first || point>segments[i].second)
3           {
4               point = segments[i].second;
5               points.push_back(point);
6           }
7
8       return points;
9   }
```

Recolectando firmas



Referencias


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009)
Introduction to algorithms


Skiena, S. S. (2003)
Programing Challenges. The Programming Contest Training Manual