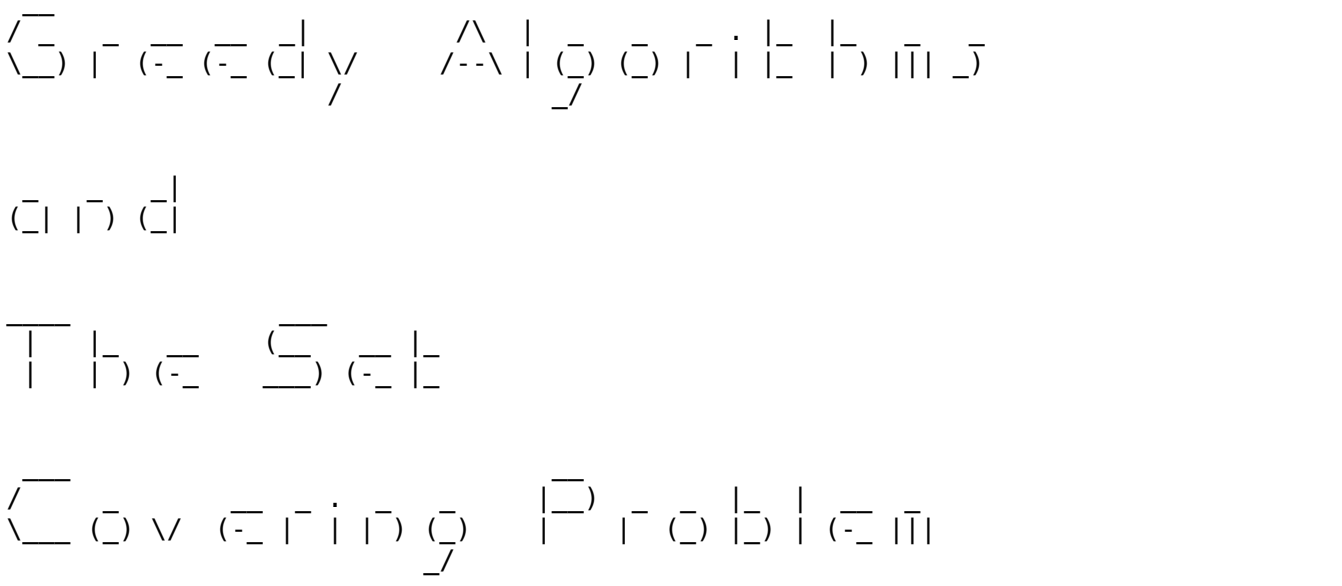
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬ 1. Introducción.

En este sección seguimos trabajando con problemas que son NP y que por lo tanto, no existe un algoritmo rápido para resolverlos.

Vamos a identificar estos problemas, pues si conocemos que un problema pertenece a esta familia, no se debe gastar tiempo tratando de construir un algoritmo rápido. En su lugar se construirá un tipo especial de algoritmo de aproximación.

Esta estrategia se conoce como “algoritmos ambiciosos”, en inglés “greedy algorithms”. Estos algoritmos no dan una solución exacta, en su lugar dan una aproximación hacia la solución exacta.

▬▬▬ 2. Set Covering Problem.

Supongamos que queremos iniciar una campaña de publicidad radial en el oeste de los Estados Unidos. Se desea poder cubrir los estados de:

Ide.	Estado
wa	Washington
mt	Montana
id	Idaho
og	Oregon
nv	Nevada
ut	Utah
ca	California
az	Arizona

En los Estados Unidos, existe una abreviación de dos letras para cada estado. La abreviación correcta de Oregon es “or”, pero en muchos lenguajes de programación es una palabra reservada, por lo que usaremos “og”.

Cada estación cobra la misma cantidad de dinero dinero por difundir el anuncio y por lo tanto se desea contratar el menor número posible de estaciones radiales. A continuación se presenta una lista con el nombre de la estación y los estados que cubren.

Estación Radial	Disponible en
kone	id, nv, ut
ktwo	wa, id, mt
kthree	og, nv, ca
kfour	nv, ut
kfive	ca, az

Esta estructura la representaremos como:

objetos:

```
[
[kone,   {id, nv, ut} ],
[ktwo,   {wa, id, mt} ],
[kthree, {og, nv, ca} ],
[kfour,  {nv, ut}     ],
[kfive,  {ca, az}     ]
];
```

Como pueden ver cada estación cubre una cierta región y algunos estados son cubiertos por más de una estación. Así que la cobertura de cada radio tiene un cierto nivel de traslape.

Se desea entonces el menor número de estaciones que cubran todos los estados que se tienen. Los cuales son:

{wa, mt, id, og, nv, ut, ca, az}

▬▬▬ 3. Solución por Búsqueda Exhaustiva.

El problema anterior parece relativamente sencillo, sin embargo es bastante difícil, además de que su solución exacta es sumamente lenta. Veamos cuál es la solución exacta.

• Haga una lista con cada subconjunto de estaciones que cubre la región completamente. El número de posibilidades está dado por 2^n.

• De esta lista, escoja el conjunto con el menor número de estaciones radiales.

Observe que para nuestra problema con 5 estaciones, tenemos el siguiente número de subconjuntos:

conjunto vacío: ⌀

conjuntos de tamaño 1: C(5,1) = 5

conjuntos de tamaño 2: C(5,2) = 10

conjuntos de tamaño 3: C(5,3) = 10

conjuntos de tamaño 4: C(5,4) = 5

conjuntos de tamaño 5: C(5,5) = 1

Al sumar estos valores se obtiene:

1 + 5 + 10 + 10 + 5 + 1 = 32

$2^5 = 32$

En general, si se tienen “n” estaciones, el número de subconjuntos estará dado por 2^n posibles subconjuntos.

$T(n) = 2^n$

$O(2^n)$

●● Generando los subconjutos.

Si se tiene el siguiente conjunto de estaciones:

radio:{kone, ktwo, kthree, kfour, kfive}

Se deberán analizar los siguiente subconjuntos:

```
(%i) powerset({kone, ktwo, kthree, kfour, kfive});

{

{},

{kone},
{ktwo}
{kthree},
{kfour},
{kfive},


{kone,  ktwo  },
{kone,  kthree},
{kone,  kfour },
{kone,  kfive },
{ktwo,  kthree},
{ktwo,  kfour },
{ktwo,  kfive },
{kthree,kfour },
{kthree,kfive },
{kfour, kfive },


{kone,  ktwo,  kthree},
{kone,  ktwo,  kfour },
{kone,  ktwo,  kfive },
{kone,  kthree,kfour },
{kone,  kthree,kfive },
{kone,  kfour, kfive },
{ktwo,  kthree,kfour },
{ktwo,  kthree,kfive },
{ktwo,  kfour, kfive },
{kthree,kfour, kfive },


{kone, ktwo,  kthree,kfour },
{kone, ktwo,  kthree,kfive },
{kone, ktwo,  kfour, kfive },
{kone, kthree,kfour, kfive },
{ktwo, kthree,kfour, kfive },


{kone, ktwo, kthree, kfour, kfive},


}
```

Cada uno de estos subconjuntos se debe analizar para ver cuáles cubren todas las estaciones. A continuación se muestran las rutas que cubren todos los estados. Recuerde que la unión de los estados debe ser igual a:
{az,ca,id,mt,nv,og,ut,wa}

```
[

{
  [kone,  {id,nv,ut}],
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfive, {az,ca}   ]
},


{
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfour, {nv,ut}   ],
  [kfive, {az,ca}   ]
},


{
  [kone,  {id,nv,ut}],
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfour, {nv,ut}   ],
  [kfive, {az,ca}   ]
}

]
```

De las rutas que cubren todas las estaciones, se selecciona la de menor tamaño. Estas serían cualquier de estas rutas:

```
{
  [kone,  {id,nv,ut}],
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfive, {az,ca}   ]
},



{
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfour, {nv,ut}   ],
  [kfive, {az,ca}   ]
},
```

▬▬▬ 4. Tiempo de Ejecución.

Tal como se mencionó anteriormente, para 5 elementos se deben generar:

2^5 = 32 subconjuntos.

En general, si se tienen “n” estaciones, el número de subconjuntos estará dado por 2^n posibles subconjuntos.

$T(n) = 2^n$

$O(2^n)$

●● Pongamos el tiempo de forma más real.

Por lo tanto, este problema se puede resolver para un pequeño número de problemas, pero conforme aumente el número de estaciones el tiempo de procesamiento se vuelve muy grande. Por ejemplo, supongamos que podemos calcular 10 subconjuntos por segundo. Veamos los tiempos en la siguiente tabla.

Número Estaciones	Tiempo requerido
5	3.2 segs
10	102.4 segs
32	13.62 años
100	4*10^23 años

▬▬▬ 5. Solución por Algoritmos Aproximados.

Por eso debemos utilizar algoritmos aproximados, en este caso una estrategia “greedy”. La solución puede ser igual que en el algoritmo exhaustivo, o puede ser una aproximación a la solución correcta. Esto depende de las condiciones del problema.

Veamos el algoritmo:

• Ordene las estaciones de radio, iniciando con la que más estaciones cubre hasta la que menos estaciones cubre.

• Escoja la estación de radio que cubra el mayor número de estados. No hay problema si esa estación también cubre algunos de los estados que ya han sido cubiertos antes.

• Repita el proceso hasta que todos los estados estén cubiertos.

Esto es lo que se llama un algoritmo de aproximación. Si calcular la solución exacta toma mucho tiempo, se puede utilizar un algoritmo que calcule una solución aproximada. Para valorar este tipo de algoritmos se deben tomar en cuenta los siguientes elementos.

• Qué tan rápidamente resuelven el problema.

• Qué tan cerca queda su solución de la solución óptima.

▬▬▬ 6. Una Corrida del Algoritmo.

Veamos una corrida del este algoritmo.
Al inicio tenemos las siguientes variables:

S: [] lista para guardar las radios escogidas.

C: {} conjunto de estados cubiertos

T: {az,ca,id,mt,nv,og,ut,wa} conjunto total que se debe cubrir

>>> Se tienen las siguientes estaciones ordenadas:

```
estaciones:
[
[kone,   {id, nv, ut} ],
[ktwo,   {wa, id, mt} ],
[kthree, {og, nv, ca} ],
[kfour,  {nv, ut}     ],
[kfive,  {ca, az}     ]
];
```

>>> Inicio.

```
[
[kone,   {id, nv, ut} ],
[ktwo,   {wa, id, mt} ],
[kthree, {og, nv, ca} ],
[kfour,  {nv, ut}     ],
[kfive,  {ca, az}     ]
];

/////

[]
{__,__,__,__,__,__,__,__}
{az,ca,id,mt,nv,og,ut,wa}
```

>>> Revisamos kone:

```
[
[kone,   {id, nv, ut} ], ◄
[ktwo,   {wa, id, mt} ],
[kthree, {og, nv, ca} ],
[kfour,  {nv, ut}     ],
[kfive,  {ca, az}     ]
];

/////

[ [kone, {id,nv,ut}]
]
```

```
{__,__,id,__,nv,__,ut,__}
{az,ca,id,mt,nv,og,ut,wa}



>>> Revisamos ktwo:

[
[kone,   {id, nv, ut} ], ◄
[ktwo,   {wa, id, mt} ], ◄
[kthree, {og, nv, ca} ],
[kfour,  {nv, ut}     ],
[kfive,  {ca, az}     ]
];

/////

[ [kone,{id,nv,ut}],
  [ktwo,{id,mt,wa}]
]
{__,__,id,mt,nv,__,ut,wa}
{az,ca,id,mt,nv,og,ut,wa}



>>> Revisamos kthree:

[kone,   {id, nv, ut} ], ◄
[ktwo,   {wa, id, mt} ], ◄
[kthree, {og, nv, ca} ], ◄
[kfour,  {nv, ut}     ],
[kfive,  {ca, az}     ]
];

/////

[ [kone,  {id,nv,ut}],
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}]
]
{__,ca,id,mt,nv,og,ut,wa}
{az,ca,id,mt,nv,og,ut,wa}



>>> Revisamos kfour:

[
[kone,   {id, nv, ut} ], ◄
[ktwo,   {wa, id, mt} ], ◄
[kthree, {og, nv, ca} ], ◄
[kfour,  {nv, ut}     ], ◄
[kfive,  {ca, az}     ]
];

/////

[ [kone,  {id,nv,ut}],
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}]
]
{__,ca,id,mt,nv,og,ut,wa}
{az,ca,id,mt,nv,og,ut,wa}


kfour no se toma en cuenta, pues no agrega elementos nuevos!
```

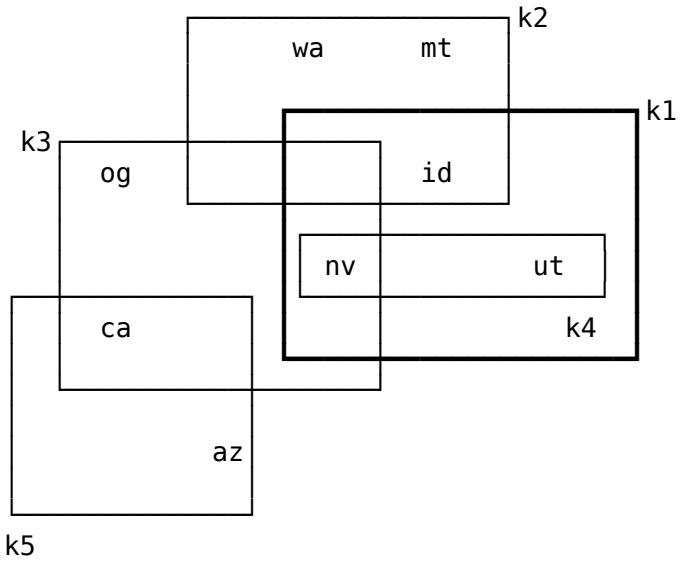
>>> Revisamos kfive:

```
[
[kone,   {id, nv, ut} ], ◄
[ktwo,   {wa, id, mt} ], ◄
[kthree, {og, nv, ca} ], ◄
[kfour,  {nv, ut}     ], ◄
[kfive,  {ca, az}     ]  ◄
];

/////

[ [kone,  {id,nv,ut}],
  [ktwo,  {id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfive, {az,ca}] ]

{az,ca,id,mt,nv,og,ut,wa}
{az,ca,id,mt,nv,og,ut,wa}
```

>>> Solución Final:

```
[ [kone,{id,nv,ut}],
  [ktwo,{id,mt,wa}],
  [kthree,{ca,nv,og}],
  [kfive,{az,ca}]
]
```

Observemos como esta solución final cumple con un recubrimiento de todos los estados:



▬▬▬ 7. Tiempo de Ejecución.

Veamos el tiempo de ejecución del algoritmo:

O(ordenar) + O(n)

$O(n*\log_2(n)) + O(n)$

$= O(n*\log_2(n))$

Comparemos los tiempos del algoritmo que da la solución exacta y nuestro algoritmo de aproximación. Al igual que antes suponemos que “n” es el número de estaciones y procesar n=10 toma un segundo.

Número Estaciones	Tiempo $O(2^n)$	Tiempo O(n*log(n))
5	3.2 segs	0.55 segs
10	102.4 segs	1.59 segs
32	13.80 años	7.68 segs
100	4*10^21 años	31.92 segs

▬▬▬ 8. La Solución Greedy es una Aproximación.

Suponga que tenemos el siguiente grupo de estaciones:

```
objetos:
[
[kone,   {og, nv, ut, ca, az} ],
[ktwo,   {id, og, nv, ut}     ],
[kthree, {mt, og ,nv, ut}     ],
[kfour,  {wa, og, nv, ut}     ],
[kfive,  {wa, mt, id}         ]
];
```

Se desean cubrir, todos los estados a los que se puede llegar:

{wa, mt, id, og, nv, ut, ca, az}

Si se ejecuta el algoritmo Greedy, se obtiene la siguiente solución:

```
[
   [kone,  {az,ca,nv,og,ut}],
   [ktwo,  {id,nv,og,ut}   ],
   [kthree,{mt,nv,og,ut}   ],
   [kfour, {nv,og,ut,wa}   ]
]
```

Sin embargo, al ejecutarse el algoritmo exhaustivo, se ve que la solución correcta está dada por:

```
[  { [kone, {az,ca,nv,og,ut}]
     [kfive,{id,mt,wa}],
   }
]
```

▬▬▬ 9. Problemas NP-Completos.

Muchos de los problema que estamos analizando son problemas NP. Se suelen llamar así porque poseen ciertas características que estudiaremos más adelante.

No hay una forma sencilla de saber si un problema es NP, pero aquí tenemos algunas guías.

• El algoritmo es rápido con un conjunto pequeño de datos, pero realmente lento cuando el número de datos aumenta.

• Buscar todas las combinaciones de “X” para resolver un problema, significa usualmente que es de esta categoría.

• Al tener que calcular cada posible versión de “X”, porque el problema no se puede dividir en subproblemas más pequeños.

• Si su problema requiere de una secuencia, secuencia de ciudades, secuencia de horarios.

• Si su problema requiere de calcular un grupo de subconjuntos.

• Si puede modelar su problema como otro problema que es NP, entonces su problema es NP.

▬▬▬ 10. Resumen.

• Se ha presentado el problema de cubrir un conjunto con base en un grupo de subconjuntos.

• Si se resuelve el problema de forma exhaustiva se tiene un problem NP, cuya tiempo de ejecución es de $O(2^n)$

• Si se resuelve por medio de un algoritmo aproximado y greedy, en el cual se ordenan los elementos, se tiene un problema cuyo tiempo de ejecución es $O(n*\log_2(n))$

▬▬▬ 11. Ejercicios.

●● Ejercicio 1.

Para cada uno de estos problema, indique si se trata de un algoritmo “greedy” o no.

- Quicksort
- Breadth First Search
- Algoritmo de Dijkstra