Estructuras de Datos

Greedy

Motivación

Son algoritmos usualmente usados para resolver problemas de *optimización* como:

- encontrar **el mejor** orden en el que ejecutar ciertas tareas;
- encontrar **la ruta más** corta para ir de un punto a otro en un mapa;
- dar el vuelto con **la menor** cantidad de monedas posible.

Es decir problemas dedicados a encontrar el valor que maximice o minimice cierta función.

Además son problemas que cuentan con la propiedad de **subestructura óptima**. Es decir, que es posible obtener una solución al problema general mediante soluciones parciales de subproblemas. Esta propiedad nos permite construir soluciones mediante la resolución de subproblemas.

Descomposición Problema Greedy

En general podemos descomponer el algoritmo greedy de la siguiente forma

- Un conjunto de candidatos: Las tareas a ejecutar, las posibles rutas a tomar, las diferentes formas de dar cambio. Son los elementos con los que se construye una solución.
- Una solución parcial de candidatos que iremos construyendo
- Una función es_solucion determina si un conjunto de candidatos es una solución (no necesariamente la óptima)
- Una función es_solucion_parcial que determina si un conjunto de candidatos forman una solución parcial.
- Una función de seleccion que indica en todo momento cual es el mejor candidato
- Una función objetivo que indica el *valor* de una solución: El tiempo necesario para ejecutar todas las tareas, la distancia total a recorrer, la cantidad de monedas, etc.

Algoritmo Greedy - Ejemplo

Problema del Cambio: En un cajero requiere dar cambio a un cliente con la **menor** cantidad de monedas. Se dispone una cantidad arbitraria de monedas de 1,5, 10 y 20 centavos.

Descomposición del problema:

- Candidatos: un conjunto finito de monedas, representando las denominaciones: 1, 5, 10, y 20 centavos.
- función es_solucion: el valor total del conjunto de candidatos hasta el momento es **exactamente** el valor que debe devolverse.
- función es_solucion_parcial: el valor total del conjunto de candidatos hasta el momento es **menor** que el total a devolverse.
- función seleccion: retorna la moneda de mayor denominación disponible en el conjunto de candidatos
- función objetivo: el número de monedas usadas en la solución.

Algoritmo Greedy

Para resolver el problema de optimización lo que haremos es utilizar que el problema presenta subestructura óptima y **construir** un conjunto de candidatos que optimicen (minimicen o maximicen) el valor obtenido por la función objetivo.

En cada momento el algoritmo llevará entonces un subconjunto de candidatos que llamaremos solucion_parcial.

Y utilizaremos la función de seleccion para elegir el mejor candidato disponible, el uso de la función de seleccion es la idea fundamental de los algoritmos Greedy: la elección del mejor candidato en ese momento sin importar los candidatos ya seleccionados ni los que se puedan llegar a seleccionar.

Resaltamos que la elección de un candidato puede desplazar la elección de otros futuros candidatos, y si es fundamental tener en cuenta este aspecto el algoritmo Greedy **no** presentará una solución óptima al problema.

Algoritmo Greedy

Sea C el conjunto de candidatos, y partiendo con el conjunto vacío como la solucion_parcial inicial, vamos a iterar el siguiente procedimiento hasta llegar a una solución. Es decir, hasta que es_solcuión(solucion_parcial) o nos quedamos sin candidatos.

1. Sea c = seleccion(C) el mejor candidato en éste momento
2. Nos preguntamos si es factible incorporar c a la solución: es_solucion_parcial(solución_parcial ∪ {c}) :
3. De **ser** factible actualizamos la solución parcial agregando el candidato: solución_parcial = solución_parcial ∪ {c} y volvemos al paso 1.
4. De **no ser** factible simplemente descartamos el candidato c y volvemos al paso 1.

Algoritmo Greedy - Pseudo Código C

```
set greedy(set C) {// Greedy toma el conjunto |C| de todos los posibles candidatos
  set solucion_Parcial = set_vacio();
  set solucion_Temporal;
  while(!es_solucion(solucion_Parcial) && !set_es_vacio(C)) {
    void *cand = seleccion(C);
    C = set_remover(C, cand);
    solucion_Temporal = set_agregar(solucion_Parcial, cand);
    if (es_solucion_parcial(solucion_Temporal))
      solucion_Parcial = solucion_Temporal;
  }
  if (!es_solucion(solucion_Parcial)) solucion_Parcial = set_vacio();// No se encontró una solución
  return solucion_Parcial;
}
```

Algoritmo Greedy

Es fácil ver porqué este tipo de algoritmos se llaman Greedy (Goloso), ya que siempre que tienen que realizar una elección de un candidato para *construir* la solución final lo hace eligiendo **el mejor candidato en ese momento sin importar los futuros candidatos**. El proceso de elección se encuentra encapsulado en la función de seleccion.

La función objetivo identifica el problema, y la tarea del algoritmo es encontrar el conjunto de candidatos que que minimicen o maximicen la función objetivo.

Veamos como resolver el ejemplo del problema del cambio siguiendo el esquema anterior.

Algoritmo Greedy - Ejemplo Problema del Cambio

```
#define Monedas 4

// Asumimos que en monedero están las denominaciones de las diferentes monedas en orden creciente

unsigned greedyMonedas(unsigned monedero[], unsigned vuelto){
  unsigned res = 0;
  for(unsigned i = Monedas - 1; (vuelto != 0) && (i >= 0); i--)
    if (vuelto >= monedero[i]){
      res ++;
      vuelto -= monedero[i];
    }
  // Corroboramos que se haya llegado a una solución
  if (vuelto != 0) res = 0;
  return res;
}
```

Algoritmo Greedy

Los algoritmos Greedy **no** resuelven todos los problemas. En la motivación se hace mención que se utilizan para problemas de optimización y para aquellos que poseen la propiedad de subestructura óptima, pero necesitamos una propiedad.

Para que el algoritmo Greedy tal cual fue planteado realmente encuentre la solución **óptima** al problema se debe asegurar que la solución óptima se puede construir utilizando sucesivamente la elección Greedy, encapsulada en la función seleccion.

Intuitivamente podemos pensar que existen ciertos problemas que requieren que en el proceso de elección de candidatos elijan un candidatos que **no sea el mejor** en ese momento. Es decir, permitir la elección de un **mal** candidato con la idea de que eventualmente **el valor total** de la solución final sea el óptimo.

Veremos más adelante cómo resolver algunos estos problemas con otra estrategia.

Algoritmo Greedy - Problema del Cambio

El problema para devolver el cambio justo planteado de forma general **no** se resuelve utilizando la estrategia Greedy.

El problema general consiste en dada un conjunto de monedas devolver el cambio justo con la menor cantidad de monedas, pero dependiendo de con qué monedas se cuenten se podrá resolver utilizando Greedy o no. En el caso del ejemplo contábamos con las denominaciones 1, 5, 10 y 20 centavos.

Pensemos ahora cuál sería la solución si contasemos con monedas de valor 1, 5 y 8 centavos, y tenemos que dar cambio de 10 centavos.

Siguiendo la estrategia Greedy elegiremos primero 8, ya que es el mayor candidato aceptable, luego 1 y 1. Dando como solución {8,1,1}, 3 monedas.

Pero el problema acepta una mejor solución, que es dar dos monedas de 5 centavos.

Algoritmo Greedy

Como moraleja del ejemplo anterior tenemos que **no siempre** es una buena estrategia elegir el mejor candidato en todo momento. Tenemos un ejemplo donde elegir un candidato que **no** era el mejor llevaba a una solución óptima.

En el caso que el problema **no** puedan ser resuelto de forma óptima por la estrategia Greedy, de todas maneras puede ser utilizada para encontrar una solución (no necesariamente óptima) para problemas de optimización con subestructura óptima.

Es decir, el algoritmo greedy si retorna una propuesta de solución, la propuesta es una solución correcta (por el uso de la función es_solucion) aunque esta puede ser subóptima (**no** ser óptima).