Algoritmos Voraces (Greedy)

Algoritmos Voraces (Greedy)
Objetivos

- Comprender la finalidad de los algoritmos greedy.
- Comprender los elementos de los algoritmos greedy.
- Comprender el esquema de los algoritmos greedy.
- Saber demostrar que un algoritmo greedy alcanza óptimo.

Algoritmos Voraces (Greedy)
Comprender la finalidad de los algoritmos greedy

Los **algoritmos greedy** se aplican a **problemas de optimización** (minimización/maximización): Pathfinding, mochila, etc.

Un algoritmo greedy toma, **en un instante determinado** y **de forma irrevocable**, decisiones en función de la información de que dispone.

Son **rápidos** y **fáciles de implementar**.

En principio, **no garantizan** alcanzar **la solución óptima** (algunos sí).

Algoritmos Voraces (Greedy)
Comprender los elementos de los algoritmos greedy

En cada instante se consideran:

- Conjunto de **candidatos a seleccionar**.
- Conjunto de **candidatos seleccionados**.
- **Función solución**: Determina si los candidatos seleccionados han alcanzado una solución completa.
- **Función de factibilidad**: Determina si es posible completar el conjunto de candidatos seleccionados para alcanzar una solución al problema.
- **Función selección**: Determina el mejor candidato del conjunto a seleccionar.
- **Función objetivo**: Da el valor de la solución alcanzada.

Algoritmos Voraces (Greedy)
Comprender el esquema de los algoritmos greedy

```
greedy(set<candidatos> C) {
  //Inicialmente C contiene todos los candidatos
  set<candidatos> S; //Solución inicial vacía.
  while (!C.empty() && !solucion(S)) {
    X = seleccionar(C);
    C.erase(x);
    if (factible(S,x))
      S.insert(x);
  }
  if (solucion(S))
    return S;
  else
    return “no hay solución”;
}
```

Algoritmos Voraces (Greedy)
Saber demostrar que un algoritmo greedy alcanza óptimo

Si el **óptimo local** forma parte de la **solución global óptima**, el algoritmo greedy nos devuelve la solución óptima.

Para ello, debemos demostrar que **la primera decisión es correcta**, y que existen **subestructuras optimales**: cuando se ha tomado la decisión número i, el problema se reduce a encontrar una solución optimal para el subproblema que tiene como candidatos aquellas actividades en S compatibles con 1...i.

Algoritmos Voraces (Greedy)
Saber demostrar que un algoritmo greedy alcanza óptimo

Demostramos por reducción al absurdo que la primera decisión tomada es la mejor posible y forma por tanto parte de la solución óptima:

Suponemos que hemos escogido un candidato que forma parte de la solución óptima.

Para que no lo hiciera, debería haber otro mejor.

Si hubiera otro mejor, con nuestra función de selección habríamos escogido ese otro. (**CONTRADICCIÓN**)

Algoritmos Voraces (Greedy)
Saber demostrar que un algoritmo greedy alcanza óptimo

Demostramos por inducción que si hemos tomado n-1 decisiones que forman parte de la solución óptima, la siguiente también lo hará:

Suponemos que los n-1 candidatos tomados hasta ahora forman parte de la solución óptima.

Para que no lo hicieran, debería haber otro mejor.

Si hubiera otra mejor, con nuestra función de selección habríamos escogido ese otro, porque la primera que tomamos siempre es la mejor y el resto quedan demostrados por esta propia inducción. (**CONTRADICCIÓN**)

En el examen, podemos demostrar que no se alcanza la solución óptima buscando un contraejemplo, es decir, un ejemplo en el que la función de selección no alcance el óptimo.

Algoritmos Voraces (Greedy)
¿Qué hemos aprendido?

- **Finalidad de los algoritmos greedy.**
- **Elementos de los algoritmos greedy.**
- **Esquema de los algoritmos greedy.**
- **Demostrar que un algoritmo greedy alcanza el óptimo.**

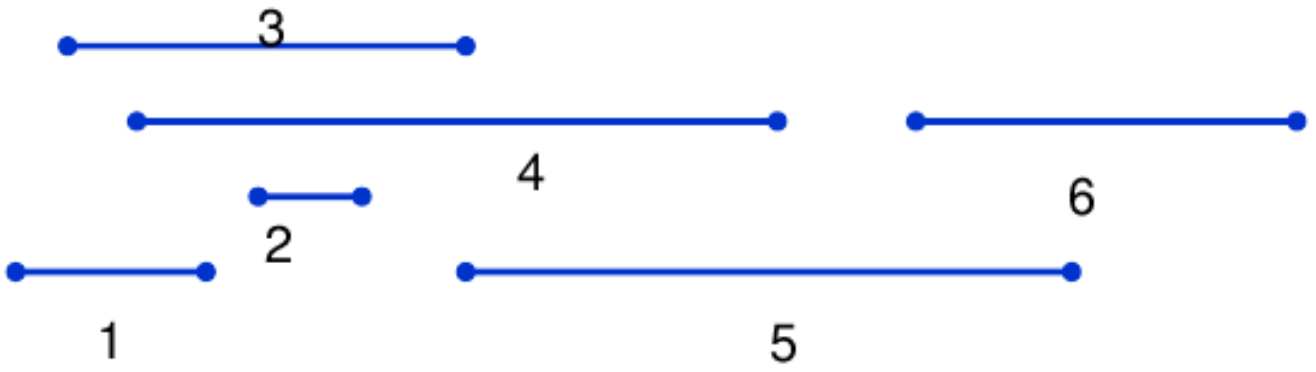
Algoritmos Voraces (Greedy)
Ejemplos

Ejemplo: Selección de actividades.

- Dado un conjunto S de n actividades

  s_i = tiempo de comienzo de la actividad i

  f_i = tiempo de finalización de la actividad i
- Encontrar el subconjunto de actividades compatibles A de tamaño máximo



Algoritmos Voraces (Greedy)
Ejemplos

Ejemplo: Árbol de recubrimiento mínimo.

Dado G(V,A) grafo conexo no dirigido, donde cada arista tiene una longitud no negativa. Encontrar un subconjunto de aristas, T, de forma que todos los vértices de G queden conectados de forma que la suma de las aristas de T tenga peso mínimo.