Algoritmos Golosos

Leopoldo Taravilse[1]

[1] Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Training Camp 2014

Contenidos

Contenidos

Definición de algoritmo goloso

Definición

Un algoritmo goloso es un algoritmo que va encontrando óptimos locales para subproblemas para en base a los óptimos locales encontrar un óptimo global que contiene a los óptimos locales.

Definición de algoritmo goloso

Definición

Un algoritmo goloso es un algoritmo que va encontrando óptimos locales para subproblemas para en base a los óptimos locales encontrar un óptimo global que contiene a los óptimos locales.

OJO!!!

No confundir con programación dinámica! Un algoritmo goloso da una solución que tiene como parte de la misma a las soluciones de los subproblemas.

Cúando podemos usar algoritmos golosos

- Es muy común que en un problema parezca que podemos utilizar un goloso cuando el goloso en realidad no anda.

Cúando podemos usar algoritmos golosos

- Es muy común que en un problema parezca que podemos utilizar un goloso cuando el goloso en realidad no anda.
- Para poder darnos cuenta cuando el goloso anda suele ser conveniente demostrar que el algoritmo es correcto. Muchas veces esto es muy difícil.

Cúando podemos usar algoritmos golosos

- Es muy común que en un problema parezca que podemos utilizar un goloso cuando el goloso en realidad no anda.
- Para poder darnos cuenta cuando el goloso anda suele ser conveniente demostrar que el algoritmo es correcto. Muchas veces esto es muy difícil.
- Es por esto que conviene aprender con ejemplos a utilizar los algoritmos golosos.

Contenidos

Enunciado

Problema

Juan tiene *n* actividades que realizar y sabe cuándo empieza y cuándo termina cada una. Lamentablemente algunas se superponen y por lo tanto no puede realizarlas todas. El problema pide la máxima cantidad de actividades que Juan puede realizar sin que se le superpongan dos de ellas.

Solución

- Para poder aplicar un algoritmo goloso, primero tenemos que ordenar las actividades para que los subproblemas tengan sentido como subproblemas cuya solución es parte de la solución del problema.

Solución

- Para poder aplicar un algoritmo goloso, primero tenemos que ordenar las actividades para que los subproblemas tengan sentido como subproblemas cuya solución es parte de la solución del problema.
- Lo primero que tenemos que hacer es ver cómo ordenarlas.

Solución

- Para poder aplicar un algoritmo goloso, primero tenemos que ordenar las actividades para que los subproblemas tengan sentido como subproblemas cuya solución es parte de la solución del problema.
- Lo primero que tenemos que hacer es ver cómo ordenarlas.
- La forma correcta de ordenarlas es por horario de finalización.

Solución

- Una vez que las actividades están ordenadas por horario de finalización vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer si hacemos sólo algunas de las primeras *i* actividades?

Solución

- Una vez que las actividades están ordenadas por horario de finalización vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer si hacemos sólo algunas de las primeras *i* actividades?
- Si la solución óptima implica hacer la actividad *i*, entonces la solución óptima para las primeras *i* actividades también implica hacer la actividad *i*.

Solución

- Una vez que las actividades están ordenadas por horario de finalización vamos resolviendo los siguientes subproblemas: ¿Cuántas actividades podemos hacer si hacemos sólo algunas de las primeras *i* actividades?
- Si la solución óptima implica hacer la actividad *i*, entonces la solución óptima para las primeras *i* actividades también implica hacer la actividad *i*.
- La forma de resolver el problema es la siguiente: Primero hacemos la actividad que empieza primero, luego cuando encontramos una actividad que podemos hacer sin superposiciones con las que ya hicimos la hacemos también.

Contenidos

Enunciado

Problema

Dado un string T, queremos reemplazar cada caracter del string por un string binario (código) de modo tal que dados dos códigos ninguno es prefijo del otro (esto se llama codificación libre de prefijos). De todas las codificaciones posibles queremos encontrar aquella que codifique T con la menor cantidad de caracteres binarios posibles.

Solución

- Supongamos que las frecuencias de los caracteres (cantidad de veces que aparecen en T) las tenemos ordenadas de la siguiente manera: $f_1 \leq f_2 \leq f_3 \leq ... \leq f_n$. Entonces sabemos que los caracteres c_n y c_{n-1} van a ser los que menos aparezcan, y por lo tanto van a tener los dos códigos más largos.

Solución

- Supongamos que las frecuencias de los caracteres (cantidad de veces que aparecen en T) las tenemos ordenadas de la siguiente manera: $f_1 \leq f_2 \leq f_3 \leq ... \leq f_n$. Entonces sabemos que los caracteres c_n y c_{n-1} van a ser los que menos aparezcan, y por lo tanto van a tener los dos códigos más largos.
- Entonces los códigos de esos dos caracteres pueden ser el mismo código salvo por el último caracter, que en uno es 0 y en otro es 1, ya que los dos códigos más largos van a tener la misma longitud y van a diferir sólo en el último bit. Si esto no pasa, al código más largo le borramos el último bit y vemos que no era óptimo el código que habíamos encontrado.

Solución

- Supongamos que las frecuencias de los caracteres (cantidad de veces que aparecen en T) las tenemos ordenadas de la siguiente manera: $f_1 \leq f_2 \leq f_3 \leq ... \leq f_n$. Entonces sabemos que los caracteres c_n y c_{n-1} van a ser los que menos aparezcan, y por lo tanto van a tener los dos códigos más largos.
- Entonces los códigos de esos dos caracteres pueden ser el mismo código salvo por el último caracter, que en uno es 0 y en otro es 1, ya que los dos códigos más largos van a tener la misma longitud y van a diferir sólo en el último bit. Si esto no pasa, al código más largo le borramos el último bit y vemos que no era óptimo el código que habíamos encontrado.
- Luego podemos tratarlos a ambos como un caracter sólo con frecuencia $f_n + f_{n-1}$, resolvemos golosamente el problema y luego le agregamos el 0 a un código y el 1 al otro.

Contenidos

Enunciado

Problema

Se tienen n números y se quiere calcular la suma de todos ellos.
Sumar $a + b$ cuesta $a + b$. Por ejemplo, si queremos sumar $1 + 2 + 3$ cuesta 9 $((1 + 2) + 3)$, 10 $((1 + 3) + 2)$ o 11 $((2 + 3) + 1)$.
El problema consiste en encontrar el costo mínimo de sumar los n números..

Solución

- La solución a este problema consiste en sumar siempre los dos más chicos.

Solución

- La solución a este problema consiste en sumar siempre los dos más chicos.
- Este es uno de los casos donde probar que la solución óptima es realmente óptima puede requerir un poco de matemática y no es trivial demostrar su correctitud.

Problemas de Tarea

- http://goo.gl/qkhVLJ
- http://goo.gl/SFDWBL