Algoritmos golosos

Algoritmos y Estructura de Datos III

Agustín Pecorari

Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

1 de abril de 2022

Algoritmos golosos:

- En inglés se llaman greedys (avaros). También les decimos miopes, ávidos, devoradores.
- Los vamos a estudiar aplicados a problemas de optimización. Particularmente a aquellos para los cuales funciona.
- Es una técnica constructiva. Es decir que, construimos la solución paso a paso.
- En cada paso tomamos la mejor decisión localmente.
- Al tomar una decisión no vuelvo atrás a revisar otras soluciones.

Algoritmos golosos:

Diseñar el algoritmo va a consistir en:

1. Dar la estructura de nuestras soluciones
   - Podemos elegirla de forma tal, de que al hacer la elección golosa, sólo haya un subproblema por resolver.
2. Decidir que elección golosa tomo y que subproblema resuelvo.
3. Probar que nuestra solución golosa sirve.

La técnica es muy sencilla. Lo más difícil es convencerse y demostrar que la decisión que tomamos nos lleva a soluciones (óptimas) del problema.

Meximización

Enunciado

Dada una secuencia de números no negativos $Y = y_1, \ldots, y_n$ se define $mex(Y, i)$ como el mínimo $y \geq 0$ que no pertenece a $Y = \{y_1, \ldots, y_n\}$. Dado un conjunto de números no negativos X, encontrar una permutación π de X que maximice $f(\pi(X)) = \sum_{i=1}^{n} mex(\pi(X), i)$.

Ejemplo 1

$Y = \{0, 1, 1, 1, 3\}$
$\pi(Y) = 0, 1, 3, 1, 1$
$mex(\pi(Y)) = 1, 2, 2, 2, 2$

Caracterizar una solución

Ejemplo 2

$Y = \{0, 1, 2, 3, 3, 4, 6, 6\}$

$\pi_1(Y) = 0, 1, 2, 3, 4, 6, 3, 6$

$mex(\pi_1(Y)) = 1, 2, 3, 4, 5, 5, 5, 5$

$\pi_2(Y) = 0, 1, 2, 3, 3, 4, 6, 6$

$mex(\pi_2(Y)) = 1, 2, 3, 4, 4, 5, 5, 5$

Idea del algoritmo goloso

El objetivo es maximizar la sumatoria del mex, por lo tanto podemos pensar en maximizar cada mex.

¿Cómo sería el algoritmo?

Formalizando ideas

Lema

Un permutación A de X es golosa si y sólo si es óptima.

Demostración

$\Rightarrow$)
Sea $B = b_1, \ldots, b_n$ una permutación cualquiera.
$P(k) \equiv$ Si $k \leq n$ entonces $\sum_{i=1}^{k} mex(A, i) \geq \sum_{i=1}^{k} mex(B, i)$ para $k \geq 0$.
Caso base: $P(0)$. Trivial, ya que ambas sumatorias valen 0.
Paso inductivo: $P(k) \Rightarrow P(k+1)$. Suponemos $k < n$.
Veamos primero que $y = mex(A, k+1) \geq mex(B, k+1)$. Trivial si $y = k+1$.
Si $y \leq k$, entonces $0, \ldots, y-1$ pertenecen a A_{k+1} e $y \notin A_{k+1}$.

Demostración

Por definición de permutación golosa, esto implica que $y \notin A$ y, como B es una permutación de A, entonces $y \notin B$
En consecuencia, $mex(B, i) \leq y = mex(A, k+1)$ para todo $1 \leq i \leq n$. Resumiendo, $y = mex(A, k+1) \geq mex(B, k+1)$ independientemente de cuál sea el valor de y.
Luego, obtemos que:

$$\sum_{i=1}^{k+1} mex(A, i) = \sum_{i=1}^{k} mex(A, i) + y \geq \sum_{i=1}^{k} mex(A, i) + mex(B, k+1)$$

$$\overset{\text{HI}}{\geq} \sum_{i=1}^{k} mex(B, i) + mex(B, k+1) = \sum_{i=1}^{k+1} mex(B, i)$$

Demostración

$\Leftarrow$)
Sea $y = mex(B, i)$ para una permutación óptima B de X y consideremos una permutación golosa A. Claramente, $mex(B, i) \leq \min\{i, y\}$ para todo $1 \leq i \leq n$. Por lo tanto,

$$z^A = \sum_{i=1}^{n} mex(A, i) \leq z^* = \sum_{i=1}^{n} mex(B, i) \leq \sum_{i=1}^{y} i + y(n - y) = z^{\mathrm{ub}}.$$

Es fácil ver que $z^A = z^{\mathrm{ub}}$ y, en consecuencia $z^* = z^{\mathrm{ub}}$. Entonces, como la única forma en que $mex(B, i) = i$ para todo $i \leq y$ es que $B[i] = i$, obtenemos que B es golosa.

1/pmtn,r_j/L_{max}

Enunciado

Debemos schedulear tareas en un único procesador. Las tareas se pueden fraccionar, tienen duración p_j y tiempo de entrega d_j y van apareciendo en determinados momentos r_j. Queremos obtener el schedule que minimice el retraso máximo de todas las tareas.

Ejemplo

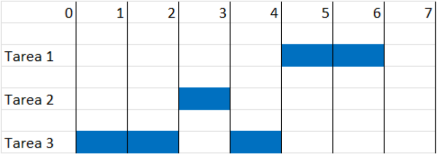
$p =< 2, 1, 3 >$
$d =< 5, 3, 3 >$
$r =< 3, 2, 0 >$

Caracterizar una solución

Ejemplo

$p =< 2, 1, 3 >$
$d =< 5, 3, 3 >$
$r =< 3, 2, 0 >$

Probemos poniendo primero las disponibles.



Veamos los retrasos: $L_1 = 1$, $L_2 = 0$, $L_3 = 1$, entonces $L_{max} = 1$.

Idea del algoritmo goloso

- El objetivo es minimizar el retraso máximo, entonces ese podría ser nuestra medida para evaluar que tarea es la siguiente.
- Pero solo podemos ir procesando las que están disponibles.
- Entonces la idea sería ir procesando, de las disponibles, la de menor d.
- Llamaremos a esta regla **preemptive EDD** (early due date).
- Ahora veamos si podemos demostrar que eso es óptimo.

Formalizando ideas

Definiciones

- $N = 1, \ldots, n$ índices de todas las tareas. $S \subseteq N$.
- $r(S) = min_{j \in S}(r_j)$
- $p(S) = \sum_{j \in S} p_j$
- $d(S) = max_{j \in S}(d_j)$
- C_j: es el momento en que termina la tarea J_j.

Lema

Para cualquier instancia de $1/\text{pmtn}, r_j/L_{max}$ se cumple $L^*_{max} \geq r(S) + p(S) - d(S)\ \forall S \subseteq N$.

Demostración.

Consideremos el schedule **óptimo**, sea J_j la última tarea en S en terminar.

- Dado que ningúna tarea empezó antes que $r(S)$ podemos decir que: $C_j \geq r(S) + p(S)$.
- Además; $d_j \leq d(S)$.
- Siempre podemos decir que: $L^*_{max} \geq L_j$.
- Pero $L_j = C_j - d_j$.
- Usando el primer punto y restando d_j;
  $C_j - d_j \geq r(S) + p(S) - d(S)$

$\square$

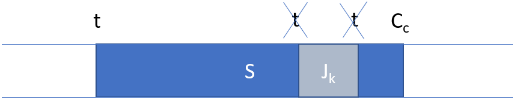
Teorema

La regla **preemptive EDD** resuelve 1/pmtn,r_j/L_{max} con $L^*_{max} = max_{S \subseteq N}(r(S) + p(S) - d(S))$.

Demostraremos este teorema mostrando que la regla **preemptive EDD** produce un schedule que alcanza el límite inferior del lema anterior.

Demostración

- Sea J_c una tarea crítica, es decir, $L_c = L_{max}$.
- Sea t el último tiempo tal que para cada tarea J_j procesada en el intervalo $[t, C_c]$ tiene $r_j \geq t$
- Sea S el subconjunto de tareas procesadas en ese intervalo.



Entonces el intervalo no contiene tiempos muertos; porque si fuese así, su final satisface el criterio utilizado para elegir t y sería posterior a t. Por lo tanto, $p(S) \geq C_c - t$.

Demostración.

- Por def. de t: $r_j \geq t$ para cada $j \in S$, si no hay tiempos muertos, tiene que haber una tarea que comience en t entonces $r(S) = t$.
- Supongamos que $d(S) = d_c$ no fuese cierto, y t' fuese el último momento en $[t, C_c]$ en el cual J_j con $d_j > d_c$ es procesada. Por eso, cada J_k procesada en $[t', C_c]$ tiene $d_k \leq d_c < d_j$. Dado que la regla **preemptive EDD** no corta la tarea J_j para empezar J_k, tenemos que $r_k \geq t'$. Pero esto implica que t' tendría que haber sido elegida en lugar de t, contradicción.

Por lo tanto, $L_{max} = L_c = C_c - d_c \leq r(S) + p(S) - d(S)$

□