TP2 Ej.1 Descripción del algoritmo

Matías Chapresto
Nicolás Dato
Ezequiel Fattori
Silvio Vileriño

8 de mayo de 2014

1. Descripción del problema

El problema consiste en modelar un juego de robanúmeros en el cual ambos jugadores juegan empleando la estrategia óptima. El juego consiste en una serie de cartas con valores enteros que los jugadores deben ir robando por turnos. Solo se puede robar una serie de cartas que sean contiguas y que comiencen por alguno de los extremos. El jugador que obtenga una suma mayor gana. El algoritmo debe calcular, a partir de una tanda de cartas, como van a ser las jugadas de ambos jugadores en cada turno, suponiendo que ambos jugadores juegan de "forma óptima", es decir, de manera de maximizar la diferencia final a favor suponiendo que el contrincante hará lo mismo cuando sea su turno. En la figura 1 se muestran ejemplos del problema junto con parte de su solución. Los turnos del primer jugador están en rojo y los del segundo jugador en azul.

2. Descripción del algoritmo

Este problema presenta una estructura recursiva. Cada vez que un jugador hace su jugada, el problema queda reducido a una instancia mas pequeña del mismo, en la cual el jugador que comienza es el otro. La estrategia para resolverlo es programación dinámica.

Se llamará A_i a la tira de números inicial, con la que el juego comienza. También se define, coloquialmente, el término "juego óptimo"para referirse al desarrollo de un juego en el cual cada jugador juega según la estrategia dada por el enunciado. Esta estrategia es: jugar en cada jugada de manera de maximizar la diferencia de puntos a favor, suponiendo que el otro jugador en cada jugada hará lo mismo. El "juego óptimo"queda entonces definido recursivamente a partir del caso base: un juego que comienza con una única ficha, que solo puede jugarse de una manera (el primer jugador agarrando esa ficha).

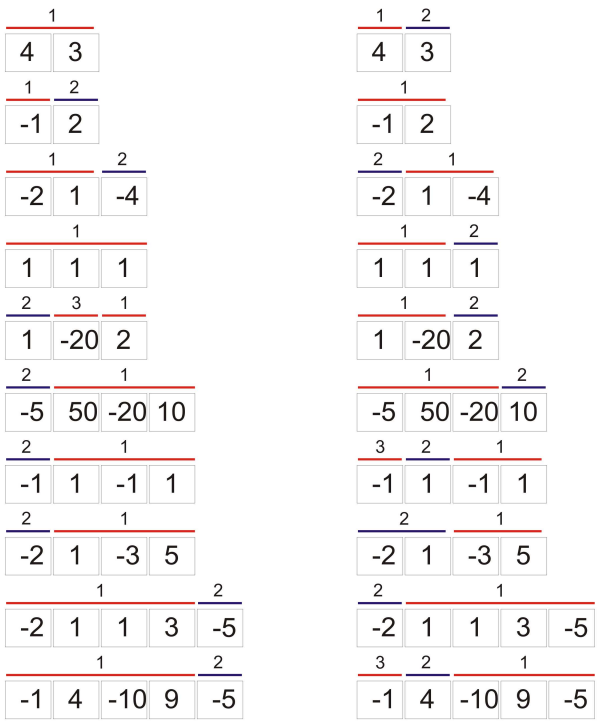

Figura 1: Soluciones óptimas del juego (izquierda) y no óptimas(derecha).

Se definen las siguientes funciones:

$$S(i,j) = \sum_{i \leq k \leq j} A_k$$

$$P(i,j,h) = p1 - p2$$

Donde $p1$ y $p2$ son los puntajes sumados por los jugadores 1 y 2 respectivamente, luego de jugar óptimamente la subinstancia dada por la subtira $A_k; i \leq k \leq j$, comenzando por el jugador h (jugador1 $= 1$, jugador2 $=$ -1).

El algoritmo se divide en tres partes:

Parte 1: Se calcula la función S. Se hace por programacíon dinámica guardando en una tabla los valores $S(i,j)$ y usando la fórmula de recursión

$$S(i,j) = S(i,j-1) + A_j.$$

(con $S(i,i-1) = 0$).

```
for i=1 to n do
  for j=i to n do
    S(i,j) = S(i,j-1) + A_j
  end for
end for
```

Parte 2: Se calcula la función P, también por programación dinámica, usando la S ya calculada. Se guarda en una tabla $P(i,j,1)$, pudiendo calcularse $P(i,j,-1) = -P(i,j,1)$ a partir de ésta. La regla de recursión es:

$$P(i,j,1) = max\{\, max\{S(i,k)+P(k+1,j,-1) \mid i \leq k \leq j\}, max\{S(k,j)+P(i,k-1,-1) \mid i \leq k \leq j\}\,\}$$

con $P(i,i-1,-1) = 0$.

Ya que la recursión requiere para el cálculo de $P(i,j)$ el cálculo de $P(i,j'); j' \leq j$ y de $P(i',j); i' \geq i$ la tabla se calcula entonces de menor a mayor en las columnas, y por cada columna de mayor a menor en las filas.
La tabla contiene en el lugar (i,j) la tupla $(res(i,j), P(i,j))$ donde $res(i,j)$ son los indices que limitan la subtira que queda por jugar despues de jugar la subtira (i,j). Si ya no se juega mas, $res(i,j) = (0,0)$.

```
1:  for j=1 to n do
2:    for i=j to 1 do
3:      MAX = -∞
4:      for  k =i-1 to j do
5:        if  S(i,k) + P(k+1,j) > MAX  then
6:          MAX ← S(i,k) + P(k+1,j)
7:          if  k ≠ j  then
8:            res(i,j) ← (k+1,j)
9:          else
10:           res(i,j) ← (0,0)
11:         end if
12:       end if
13:     end for
14:     for  k = j+1 to i do
15:       if  S(k,j) + P(i,k-1) > MAX  then
16:         MAX ← S(k,j) + P(i,k-1)
17:         if  k ≠ i then
18:           res(i,j) ← (i,k-1)
19:         else
20:           res(i,j) ← (0,0)
21:         end if
22:       end if
23:     end for
24:   end for
25: end for
```

Parte 3: Se calcula, a partir de las tablas ya calculadas, el desarrollo del juego a partir de su instancia inicial. Se guarda la cantidad de turnos jugados y las fichas robadas por izquierda o derecha en cada turno.

var $turno \leftarrow 1$
var $puntj1 \leftarrow 0$
var $puntj2 \leftarrow 0$
var $i \leftarrow 1$
var $j \leftarrow n$
var $jugadas(lado, cant)$
loop
  if $turnos = impar$ **then**
    $puntj1 \leftarrow puntj1 + puntoslevantados$
  else
    $puntj2 \leftarrow puntj2 + puntoslevantados$
  end if
  if $levantoporizq$ **then**
    $jugadas.lado[turno - 1] \leftarrow izq$
    $jugadas.cant[turno - 1] \leftarrow cantcartaslevantadas$
  else
    $jugadas.lado[turno - 1] \leftarrow der$
    $jugadas.cant[turno - 1] \leftarrow cantcartaslevantadas$
  end if
  if $res(i, j) = (0, 0)$ **then**
    break
  end if
  $(i, j) \leftarrow res(i, j)$
  $turno \leftarrow turnos + 1$
end loop

Donde *puntoslevantados*, *cantcartaslevantadas* y *levantoporizq* se obtienen como función $O(1)$ de $res(i, j)$, y no se explicitan para simplificar la exposición. Las variables $puntj1$, $puntj2$, *turno* y *jugadas* se usan luego para construir la salida.

3. Correctitud del algoritmo

A continuación se da una demostración de por qué la estrategia elegida cumple el requerimiento del enunciado. Este requerimiento es: "maximizar la diferencia de puntos a favor al final del partido, asumiendo que el otro jugador tiene la misma estrategia".
La estrategia elegida es: para cada posible tira que se puede extraer, calcular la suma de los números de la tira y sumarle la diferencia que se consigue a favor al jugar de forma óptima durante el resto del juego (que ya está definida por ser el resto del juego una instancia más pequeña). Extraer la tira que maximice este número.
Sea entonces A_k una tira inicial de cartas. Si $A_k = (c1)$ hay una única manera

de jugar, y por lo tanto la estrategia cumple el requerimiento. Supóngase que la estrategia cumple el requerimiento si A_k es de longitud $\leq m$. Para A_k de longitud $m+1$, una forma genérica de jugar el juego es una elección de cartas a robar inicialmente, de puntuación total S, seguida de un cierto desarrollo en el cual la diferencia conseguida a favor es P. Llámese P' a la diferencia a favor conseguida como resultado de jugar óptimamente el juego que comienza a partir de la primer elección de cartas (comenzado por el jugador2). Por hipótesis inductiva se cumple que $P \leq P'$ luego la diferencia a favor obtenida en este juego es $S + P \leq S + P'$. Además, por definición, $S + P' \leq M$ donde M es la diferencia a favor que se consigue haciendo la primer elección según la estrategia, y jugando óptimo durante el resto del juego.

Por último resta ver que esta estrategia se traduce formalmente en la recursión dada para $P(i, j, h)$ en la descripción del algoritmo, pero esto es muy fácil de ver por inducción y no se escribe.

4. Complejidad

La parte 1 del algoritmo es $O(n^2)$, ya que son 2 bucles for anidados con interior $O(1)$ con a lo sumo n iteraciones cada uno. La parte 2 del algoritmo es $O(n^3)$ ya que son 2 bucles for anidados con a lo sumo n iteraciones cada uno, y dentro de ellos hay dos bucles for sucesivos, cada uno de ellos $O(n)$. La parte 3 del algoritmo es O(n) ya que el loop se ejecuta a lo sumo tantas veces como la cantidad de turnos, que es a lo sumo n, y el interior del loop es $O(1)$.
Por lo tanto el tiempo total del algoritmo es $O(n^3)$

5. Código fuente

Ver Apéndice.

6. Verificación

Para la verificación se emplearon las mismas instancias que fueron dadas como ejemplo del problema y su resolución. Éstas fueron calculadas a mano, y luego se verificó que efectivamente el algoritmo daba el mismo resultado (la misma diferencia de puntos a favor) en cada caso.

7. Análisis de performance

La segunda parte del algoritmo, el cálculo de la tabla P, es de complejidad $O(n^3)$ y se lleva la mayor parte del tiempo de ejecución para n grande. El cálculo

de esta tabla, que tiene $n^2/2$ posiciones, involucra para cada posición el cálculo del máximo de una cantidad de elementos que depende únicamente de la posición que se cálcula, no del contenido de las cartas. Como el tiempo que demora el algoritmo está dominado por esta parte, es de esperar que el contenido de las cartas no cambie apreciablemente los tiempos de ejecución. Ésto se puede constatar por medio de la experimentación.

Se hizo una experimentación con tamaños n de entrada entre 100 y 400. Se distinguió la distribución de los valores de las cartas en tres casos : uniforme en $[-50; 50]$, uniforme en $[-100; 0]$ y uniforme en $[0; 100]$. Todos los casos dieron tiempos similares. En la gráfica puede observarse una curva similar a las de la forma ax^m. Se puede comprobar que al variar de $n = 200$ a $n' = 300$, $n' = (3/2)n$, el tiempo pasa de un promedio de 200 a un promedio de $650 \sim 200.(3/2)^3$. Ésto comprueba que la curva es cúbica. En el apéndice se adjuntan las gráficas experimentales de $tiempo(n)$.