Asignatura Programación y Algoritmos

Proyecto de Curso - Primer Semestre

El Traveling Tournament Problem desde la Serie Nacional de Béisbol cubana

Sofía Albizu-Campos Rodríguez

Grupo M1

Curso 2019-2020

Tarea 1

Esta operación es sencilla y se puede ejecutar a mano. El resultado se ilustra en la primera sección del código (matriz *Cal9*), junto con el calendario en la forma 5 (matriz *Cal*), la matriz de distancias entre provincias (matriz *DistProv*) y la lista de las subseries de la primera ronda, que determina la distribución Local/Visitador que se empleará durante todo el torneo (matriz *Subseries*). Todos estos datos están proporcionados en el enunciado del problema.

Tarea 2

En esta tarea debemos convertir un calendario dado en cualquier forma (5 o 9) a la otra. Para ello crearemos dos funciones que se encuentran en la segunda sección del código.

a) La función *C5aC9* convierte un calendario de la forma 5 a la forma 9. Recibe para ello como parámetros un calendario dado en la forma 5, la cantidad de equipos en cuestión y el número de rondas que tiene el campeonato. Por ejemplo, podemos convertir el calendario *Cal* en el calendario *Cal9* ejecuntando: *C5aC9(Cal,6,3)*.

b) La función *C9aC5* convierte un calendario de la forma 9 a la forma 5, recibiendo para ello los mismos parámetros que *C5aC9*, pero en este caso el calendario dado en la forma 9. Por ejemplo, convertimos *Cal9* en el calendario *Cal* ejecuntando: *C5aC9(Cal,6,3)*.

La implementación de ambas funciones es muy sencilla, valiéndose cada una de un doble ciclo *for*.

Tarea 3

Nuevamente crearemos dos funciones, esta vez para calcular las distancias que recorren los equipos en el torneo.

a) La función *distC9* es la encargada de realizar la tarea en el caso de un calendario expresado en la forma 9. Recibe como parámetros un calendario en la forma 9, la cantidad de equipos que se encuentran compitiendo, el número rondas que posee el torneo y una matriz de distancias, en este caso *DistProv* pues se trata de Cuba. Devuelve en primer lugar una lista de igual longitud que la cantidad de equipos, donde el elemento i representa la distancia total recorrida por el equipo i, y seguidamente un número, que será la distancia total recorrida por todos los equipos (en ambos casos en kilómetros). Por ejemplo: *distC9(Cal9,6,3,DistProv)=[[7982, 3244, 5556, 5986, 5288, 4965], 33021]*.
Para la implementación del código nos valemos de una lista auxiliar l para cada equipo i, donde el elemento j de esta lista es igual al número que representa al equipo en cuyo estadio se encuentra jugando el equipo i en la jornada j. Luego de la creación de estas listas es muy sencillo calcular las respectivas distancias.

b) La función *distC5* recibe los mismos parámetros que *distC9*, pero esta vez un calendario en la forma 5, y devuelve lo mismo que la función explicada anteriormente. Su implementación se vale del uso de la función *C5aC9* para expresar la matriz recibida en la forma 9 y luego emplea *distC9* para realizar el cálculo requerido.

Tarea 4

Para completar la tarea en cuestión, notemos que un calendario dado es

válido (asumimos que la matriz tiene el tamaño apropiado y que sus elementos son números enteros no negativos inferiores a la cantidad de equipos, o números enteros de módulos inferiores a la cantidad de jornadas) si y solo si verifica las siguientes condiciones:

1) *Un equipo no juega en la misma jornada con dos equipos distintos.* Para verificarlo se encuentran las funciones *verifbasicC9* y *verifbasicC5*, que reciben como parámetros el calendario (en la forma 5 y en la 9 respectivamente), la cantidad de equipos y la cantidad de rondas. Devuelven *True* o *False* en dependencia de si el calendario cumple o no la restricción. Estas características son análogas para las funciones que se mencionarán a continuación.

2) *Se cumple la restricción TTP-k, es decir, un equipo no puede celebrar más de* $k > 1$ *subseries consecutivas como local o visitador.* Para ello se tienen las funciones *verif1C5* y *verif1C9*, que reciben además el parámetro k para realizar sus cometidos.

3) *Cada par de equipos juega exactamente una vez en cada ronda y se alterna en cuanto a local y visitador cada vez que juega.* Encontramos para esto las funciones *verif21C5* y *verif21C9*. Estas no reciben el parámetro que indica la cantidad de rondas, puesto que solo se pueden utilizar para un torneo de 3.

4) *No hay subseries de ida y vuelta (dos equipos no pueden celebrar dos subseries consecutivas).* Para ello proponemos *verif22C5* y *verif22C9*. Como en el caso anterior, estas también solo se utilizan para un torneo de 3 rondas, luego no reciben este parámetro.

Finalmente, tenemos las funciones que *verifC5* y *verifC9*, que realizan todas las verificaciones anteriores para un torneo de 3 rondas. Reciben el calendario (en la forma 5 y en la 9 respectivamente), la cantidad de

equipos y el parámetro k de la restricción *TTP-k*, devolviendo *True* o *False*, en dependencia de si el calendario es válido o no. Por ejemplo: *verifC5(Cal,6,3)=True.*

Tarea 6

Tenemos varias propuestas en relación con el problema de la optimización, pero primeramente definimos las siguientes funciones auxiliares:

a) *Mascara*: Recibe como parámetro una lista de subseries y transforma la lista dada en una matriz de la forma 9 con elementos 0, 1 y -1, que no ofrece información sobre los días en que se realiza cada subserie, sino solo la información acerca de qué equipo será local en la ronda en cada subserie. Por ejemplo:
Mascara(Subseries)=[[0, 1, 1, -1, -1, -1], [-1, 0, 1, -1, 1, 1], [-1, -1, 0, -1, -1, 1], [1, 1, 1, 0, -1, -1], [1, -1, 1, 1, 0, -1,], [1, -1, -1, 1, 1, 0]].

b) *ListasPosibles*: Función que dada una lista, devuelve la lista de todas las listas que son desordenaciones de la dada. Por ejemplo:
ListasPosibles([1,2,3])=[[3, 2, 1], [2, 3, 1], [3, 1, 2], [1, 3, 2], [2, 1, 3], [1, 2, 3]].
Utiliza para ello el método recursivo.

c) *Ubicar*: Función que dados un número, una lista ordenada de números, el índice del primer elemento de la lista (0) y la longitud de esta, devuelve la posición que debería ocupar el número si se insertara en la lista. Por ejemplo: *Ubicar(3,[1,2,4],0,3)=2*. La función utiliza para esto el método de búsqueda binaria.

La función *Calendario*, con ayuda de las funciones auxiliares *Mascara* y *ListasPosibles* y de las diferentes funciones de verificación, devuelve un

calendario válido en la forma 9 a partir de la lista de subseries, que además cumple que las tres rondas solo difieren entre sí por la alternancia de Local/Visitador, es decir, el orden de los enfrentamientos no cambia. Esto es una condición deseable puesto que resulta más justo para los equipos.

Ahora, para optimizar el calendario *Cal* proponemos las siguientes funciones (todas trabajarán a partir de ahora con calendarios de la forma 9 y utilizarán casi todas las funciones concebidas anteriormente):

1) *Optimo*: Esta función consigue recorrer todos los posibles calendarios y devuelve aquel en que la distancia total recorrida sea mínima. No obstante, es demasiado costosa. Se calcula que tardaría horas, casi medio día, en realizar su objetivo. Por lo tanto, no es aconsejable.

2) *OptimoYparejo*: Esta función recorre todos los calendarios válidos que son del tipo de aquel que devuelve la función *Calendario*, es decir, cuyas rondas solo varían en la alternancia Local/Visitador. Finalmente, devuelve *distC9(C,6,3,DistProv)* y *C*, donde *C* es el calendario de este tipo que minimiza la distancia total recorrida. Esta es una función instantánea, de modo que en menos de un segundo obtenemos:
OptimoYparejo(Subseries)=([[4917, 3407, 5347, 4303, 3783, 3595], 25352], [[0, 3, 2, -1, -4, -5, 0, -8, -7, 6, 9, 10, 0, 13, 12, -11, -14, -15], [-3, 0, 4, -2, 5, 1, 8, 0, -9, 7, -10, -6, -13, 0, 14, -12, 15, 11], [-2, -4, 0, -5, -1, 3, 7, 9, 0, 10, 6, -8, -12, -14, 0, -15, -11, 13], [1, 2, 5, 0, -3, -4, -6, -7, -10, 0, 8, 9, 11, 12, 15, 0, -13, -14], [4, -5, 1, 3, 0, -2, -9, 10, -6, -8, 0, 7, 14, -15, 11, 13, 0, -12], [5, -1, -3, 4, 2, 0, -10, 6, 8, -9, -7, 0, 15, -11, -13, 14, 12, 0]]).

3) *Caltimus1*: Esta función devuelve un buen calendario en cuanto a la distancia total recorrida, recibiendo como parámetros la lista de las subseries y tres números (*m1, m2, m3*). Estos últimos tres números sirven para determinar cuán bueno será el calendario que se obtie-

ne, pero por supuesto, mientras mayores sean más tiempo tomará el algoritmo en ejecutarse. En efecto, *Caltimus1* se dedica a combinar los *m1* mejores calendarios de la primera ronda con los mejores *m2* mejores de la segunda con los *m3* mejores de la tercera. Esto lo hace a partir de las listas ofrecidas por las funciones auxiliares *ListOrdR_1* y *ListOrdR_2* definidas antes de *Caltimus*, las cuales devuelven todos los calendarios posibles de la primera y la segunda ronda respectivamente, ordenados en dos listas por la distancia total que recorren los equipos en cada uno. Nótese que no es necesaria una función *ListOrdR_3*, pues devolvería lo mismo que *ListOrdR_1*.
Para *m1, m2, m3* lo suficientemente grandes, *Caltimus1* devuelve el mismo calendario que *Optimo*, pero este no es el objetivo. Sin embargo en aproximadamente media hora, podemos obtener:
Caltimus1(Subseries,200,200,100)=([[5068, 2754, 4720, 4350, 3993, 2775], 23660], [[0, 1, 5, -2, -4, -3, 0, -10, -8, 9, 7, 6, 0, 14, 13, -15, -11, -12], [-1, 0, 4, -3, 2, 5, 10, 0, -7, 6, -9, -8, -14, 0, 11, -13, 12, 15], [-5, -4, 0, -1, -3, 2, 8, 7, 0, 10, 6, -9, -13, -11, 0, -12, -15, 14], [2, 3, 1, 0, -5, -4, -9, -6, -10, 0, 8, 7, 15, 13, 12, 0, -14, -11], [4, -2, 3, 5, 0, -1, -7, 9, -6, -8, 0, 10, 11, -12, 15, 14, 0, -13], [3, -5, -2, 4, 1, 0, -6, 8, 9, -7, -10, 0, 12, -15, -14, 11, 13, 0]])
Donde *[[5068, 2754, 4720, 4350, 3993, 2775], 23660]* es la lista de distancias recorrida por cada equipo más la distancia total.

4) *Caltimus2*: Es muy similar a *Caltimus1*, pero recibe dos párametros *m1, m2* en vez de tres, pues se dedica a combinar los mejores *m1* calendarios de las dos primeras rondas con los mejores *m2* de la tercera, utilizando las funciones *ListOrdR_1* y *ListOrdR_12*, donde *ListOrdR_12* ofrece las lista de todos los posibles calendarios de la primera y la segunda rondas, ordenados por distancia total recorrida. En alrededor de 15 minutos obtenemos:
Caltimus2(Subseries,200,200)=([[5290, 2426, 4511, 4350, 3895, 3244], 23716], [[0, 2, 3, -1, -5, -4, 0, -6, -8, 7, 9, 10, 0, 15, 11, -14, -12,

-13], [-2, 0, 5, -3, 4, 1, 6, 0, -9, 10, -7, -8, -15, 0, 12, -13, 14, 11], [-3, -5, 0, -4, -1, 2, 8, 9, 0, 6, 10, -7, -11, -12, 0, -15, -13, 14], [1, 3, 4, 0, -2, -5, -7, -10, -6, 0, 8, 9, 14, 13, 15, 0, -11, -12], [5, -4, 1, 2, 0, -3, -9, 7, -10, -8, 0, 6, 12, -14, 13, 11, 0, -15], [4, -1, -2, 5, 3, 0, -10, 8, 7, -9, -6, 0, 13, -11, -14, 12, 15, 0]])

Conclusiones

Hemos llegado a generar tres posibles calendarios válidos, donde en cada uno se recorren en total 25352km, 23660km y 23716km respectivamente. Estas distancias son muchísimo menores que 33021km. De hecho, es posible verificar que le calendario ofrecido por la Comisión Nacional es de los peores que existen en cuanto a distancia total recorrida y además, es también uno de los más desiguales en cuanto a distancia recorrida por equipo.