9-6-2023

Trabajo Práctico N2: Backtracking.

Análisis y Diseños de Algoritmos 2

UNIVERSIDAD DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES INGEÑIERIA EN SISTEMAS Profesores: Dra. Virginia Cifuentes, Ing. Diego Tolbaños, Dra. Virginia Yannibelli. Alumnas: Conti Verónica, Artaza Sheila.

**ÍNDICE:**

**INTRODUCCIÓN………………………………………………………………………………………………………………………….2**

**BACKTRACKING………………………………………………………………………………………………………………………….2**

**CARACTERISTICAS……………………………………………………………………………………………………………….3**

**PODA………………………………………………………………………………………………………………………………….3**

**PROBLEMAS DE OPTIMIZACION………………………………………………………………………………………….3**

**COTA GLOBAL……………………………………………………………………………………………………………….3**

**APLICACIONES DE BACKTRACKING………………………………………………………………………………………3**

**ESPACIO DE SOLUCION………………………………………………………………………………………………………………4**

**ALGORITMO PARA RESOLVER EL ENUNCIADO……………………………………………………………………………5**

**SEGUNDA VERSION DEL ALGORTIMO……………………………………………………………………………………6**

**VERSIÓN FINAL DE ALGORITMO BACKTRACKING………………………………………………………………….8**

**COMPARACIONES……………………………………………………………………………………………………………………..8**

**CON PODA…………………………………………………………………………………………………………………………..9**

**SIN PODA…………………………………………………………………………………………………………………………….9**

**CONCLUSIÓN……………………………………………………………………………………………………………………………10**

**BIBLIOGRAFIA…………………………………………………………………………………………………………………….......11**

**INTRODUCCIÓN**

En el siguiente informe realizaremos la técnica de programación Backtracking para resolver el siguiente enunciado:

* En una liga de fútbol participan n equipos (suponemos que n es par). En cada jornada se

juegan n/2 partidos, que enfrentan a dos equipos, dirigidos por un árbitro. Existen m árbitros disponibles, siendo m>n/2.

* Cada equipo i valora a cada árbitro j con una puntuación P[i, j] entre 0 y 10, indicando su preferencia por ese árbitro. Un valor alto indica que le gusta el árbitro y un valor bajo que no le gusta.
* Objetivo: (para cada jornada concreta) asignar un árbitro distinto a cada partido, de manera que se maximice la puntuación total de los árbitros asignados, teniendo en cuenta las preferencias de todos los equipos.

**BACKTRACKING**

Es un algoritmo basado en la técnica de búsqueda exhaustiva, una exploración sistemática del espacio de soluciones factibles de un problema.

Es común representar al espacio del problema por un árbol, el estado inicial corresponde a la raíz del árbol.

Realiza un recorrido sistemático del árbol n-ario en el que se representan las distintas soluciones.

El backtracking permite de una forma sistemática y organizada generar y recorrer estados del espacio de posibles soluciones factibles. Se basa en una generación en profundidad.

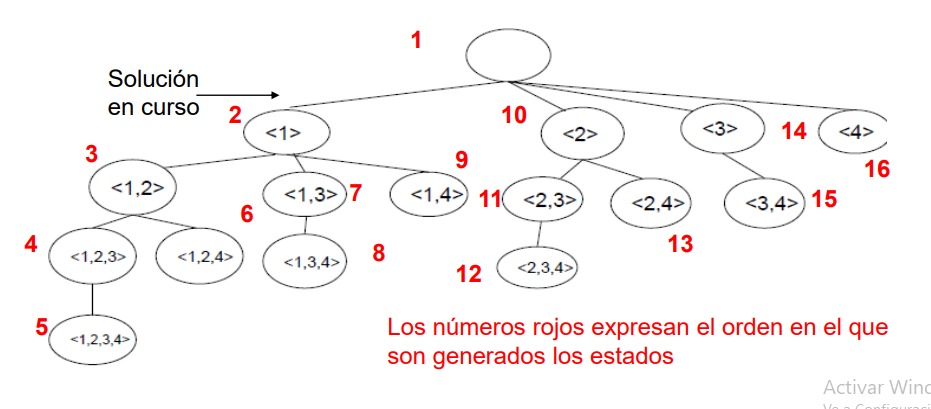


Imagen: árbol n-ario de solución generado por backtracking.

**CARACTERISTICAS:**

* Los nodos del árbol reflejan soluciones factibles. Si la solución actual en curso cumple las restricciones es factible.
* Existe una función que permite determinar si una solución actual debe ser podada o no.
* Existe una función solución que permite determinar si una secuencia de soluciones factibles es solución factible.
* Puede resolver problemas de optimización: maximización y minimización.

**PODA:** Función que permite eliminar zonas del espacio de búsqueda con el objetivo de no analizar estados que no son factibles porque no conducen a la solución final requerida.

**PROBLEMAS DE OPTIMIZACION:** Utiliza una cota global mediante la cual expresa que la solución óptima nunca será peor que la cota.

**COTA GLOBAL:** Es el valor de la mejor solución estudiada hasta el momento, se actualiza siempre que mejore su valor actual.

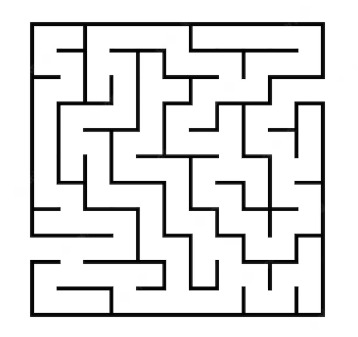
En nuestro caso realizamos un problema de maximización, por lo que:

**COTA GLOBAL <= ÓPTIMO**

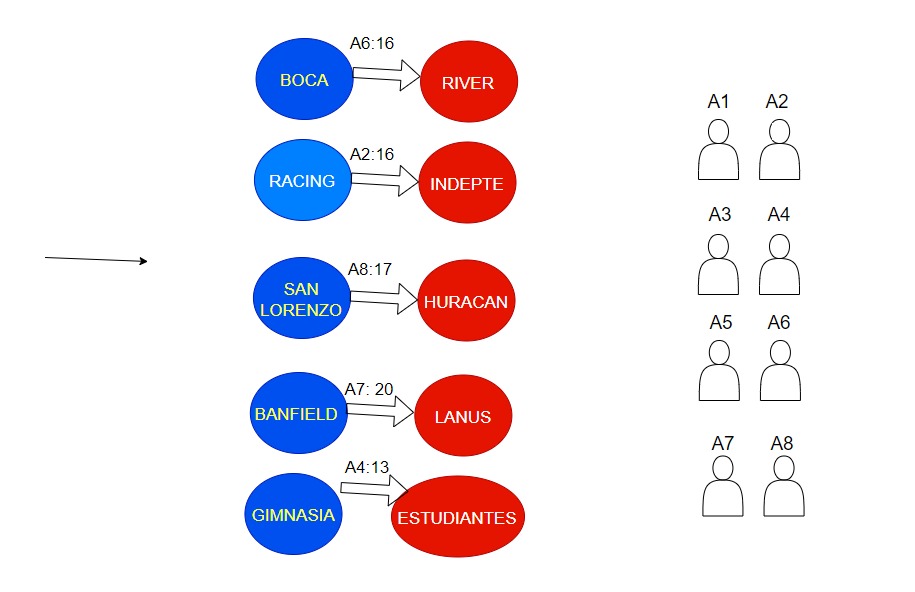
Buscamos obtener siempre el mayor puntaje del desempeño de los árbitros por jornada.

**APLICACIONES DE BACKTRACKING:** Nos permite resolver problemas como

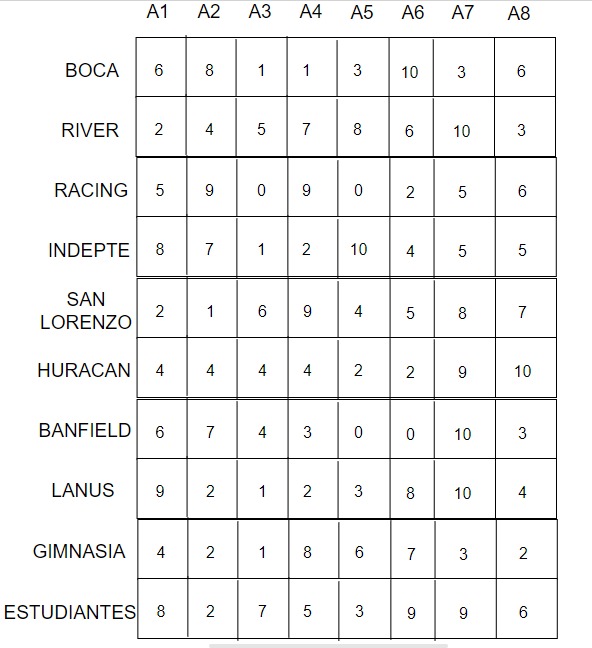
* Búsqueda de caminos (Pathfinding).
* Búsqueda con adversario. Juegos.
* Búsqueda con restricciones. Ej: Laberinto.



**ESPACIO DE SOLUCION**

****

**Imagen: demostración de asignación de árbitros por partido con su mayor puntaje en base a la imagen de abajo.**

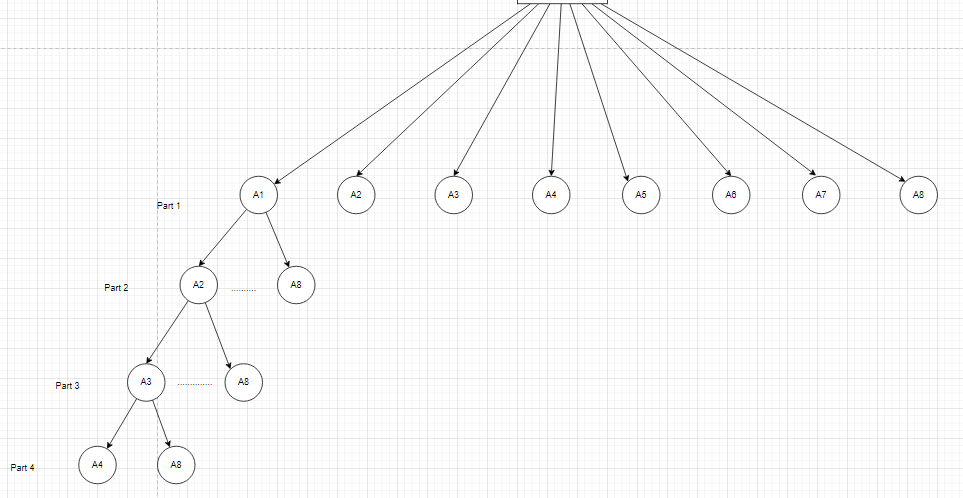
****

**Imagen: puntajes para cada árbitro dada por cada equipo.**

En la imagen que se muestra a continuación podemos ver cómo se va creando el árbol del espacio de solución, en el cual partimos de la cantidad de árbitros totales y cada nivel del árbol corresponde a un partido, en este ejemplo, 7 partidos, ya que la cantidad de árbitros corresponde a (cantidad de equipos/2) + 1.

El primer partido elije el árbitro A1, (que se asume que es el de mayor puntaje), luego el partido dos al árbitro A2, (que se asume que es el de mayor puntaje), sin repetir árbitros, y así sucesivamente hasta ocupar todos los árbitros.

Luego lo que realiza backtracking es consultar volviendo al paso anterior del estado hoja, si existe otro arbitro para seleccionar y si este genera un nuevo valor máximo en dicha jornada.

****

**Imagen: demostración de cómo se va creando el árbol n-ario del espacio de solución.**

**ALGORITMO PARA RESOLVER EL ENUNCIADO:**

Para poder resolver el trabajo decidimos utilizar las estructuras:

* Matriz para el puntaje de los equipos(filas) a los árbitros(columnas).
* Par de dos enteros, jornadas, con la cantidad de rondas y la cantidad de partidos.
* Arreglo que guarda el puntaje máximo por cada jornada, siendo su tamaño la cantidad de rondas. Cantidad de rondas = cantidad de equipos – 1.
* Lista de lista de enteros que guarda los árbitros por jornada.

Con el objetivo de resolver el problema propuesto comenzamos efectuando el algoritmo recursivo BACKTRACKING enseñado por la cátedra. A continuación, la primera versión del algoritmo:

**void Back (estado d, solucion &sol)**

**{ int nrohijo;**

**estado sig;**

**if (hoja(d))**

**calcular-solucion(d,sol);**

**else {**

**nrohijo =1;**

**while (hijos(d, nrohijo, sig) && !podado(sig, sol)) {**

**Back(sig, sol);**

**nrohijo= nrohijo +1;**

**}**

**}**

**}**

**SEGUNDA VERSION DEL ALGORTIMO:**

Finalmente agregamos una función para calcular todas las jornadas posibles, sistema de todos contra todos, (round-robin), que es la que invoca al backtracking para resolver cada fecha.

En este sistema cada equipo del torneo se enfrenta contra todos los demás en jornadas que involucran una competencia de pares, es decir un equipo contra otro equipo, y en un número constante de oportunidades. Este sistema se diferencia del torneo de eliminación, donde el perdedor no participa más y el ganador pasa a una siguiente fase; en el sistema de liga el perdedor sigue participando hasta jugar contra cada uno de los competidores.

Este tipo de torneo se utiliza habitualmente en las ligas nacionales de deportes de pelota como baloncesto, béisbol, fútbol, tenis y rugby, donde varios equipos se enfrentan en temporadas de medio año o año completo.

**void calcularArbitrosJornadas (pair<int,int>jornadas[][CANT\_PARTIDOS] , int puntajes\_arbitros [][CANT\_ARBITROS] ,**

**list < list <int> > & arbitros\_jornadas , int puntaje\_maximo\_jornadas[] ){**

**for (int i = 0 ; i < CANT\_RONDAS ; i++){**

**int puntaje\_maximo = INT\_MIN;**

**list<int> arbitros;**

**backMaxArbitros(jornadas , i , 0 , puntajes\_arbitros , 0 , puntaje\_maximo , arbitros , arbitros);**

**cout << ".................."<<endl;**

**puntaje\_maximo\_jornadas[i] = puntaje\_maximo;**

**arbitros\_jornadas.push\_back(arbitros);**

**}**

**}**

**void backMaxArbitros (pair<int,int> jornadas [][CANT\_PARTIDOS] , int fila , int num\_partido , int puntajes\_arbitros [][CANT\_ARBITROS] ,**

**int puntaje\_actual , int&puntaje\_maximo , list<int> arbitros\_actuales , list<int> &arbitros){**

**if( arbitros\_actuales.size() == CANT\_PARTIDOS ){//Estado hoja**

**if(puntaje\_actual > puntaje\_maximo){//Consulta sobre maximización**

**puntaje\_maximo = puntaje\_actual;**

**arbitros = arbitros\_actuales;**

**mostrarArbitros(arbitros);**

**cout<< "puntaje maximo"<<puntaje\_maximo<<endl;**

**}**

**}**

**else {**

**for (int i = 0; i < CANT\_ARBITROS ; i++) //Vario cada arbitro**

**if(!arbitroElegido(arbitros\_actuales, i ) ) { //Poda por arbitro no elegido**

**arbitros\_actuales.push\_back(i);**

**//cout << "puntaje actual" <<puntaje\_actual << endl;**

**puntaje\_actual += puntajes\_arbitros[ jornadas[fila][num\_partido].first][i] + puntajes\_arbitros[jornadas[fila][num\_partido].second][i];**

**backMaxArbitros (jornadas , fila , num\_partido+1 , puntajes\_arbitros , puntaje\_actual , puntaje\_maximo , arbitros\_actuales , arbitros);**

**arbitros\_actuales.pop\_back();**

**puntaje\_actual -= puntajes\_arbitros[ jornadas[fila][num\_partido].first][i] + puntajes\_arbitros[jornadas[fila][num\_partido].second][i];**

**}**

**}**

**}**

**VERSIÓN FINAL DE ALGORITMO BACKTRACKING:**

**void backMaxArbitros (pair<int,int> jornada [] , int cant\_partidos , int num\_partido , int puntajes\_arbitros [][8] ,**

**int puntaje\_actual , int&puntaje\_maximo , list<int> arbitros\_actuales , list<int> &arbitros){**

**if( arbitros\_actuales.size() == cant\_partidos ){//Estado hoja**

**if(puntaje\_actual > puntaje\_maximo){//Consulta sobre maximización**

**puntaje\_maximo = puntaje\_actual;**

**arbitros = arbitros\_actuales;**

**mostrarArbitros(arbitros);**

**cout<< "puntaje maximo"<<puntaje\_maximo<<endl;**

**}**

**}**

**else {**

**for (int i = 0; i < CANT\_ARBITROS ; i++) //Vario cada arbitro**

**if(!arbitroElegido(arbitros\_actuales, i ) ) { //Poda por arbitro no elegido**

**arbitros\_actuales.push\_back(i);**

**//cout << "puntaje actual" <<puntaje\_actual << endl;**

**puntaje\_actual += puntajes\_arbitros[ jornada[num\_partido].first][i] + puntajes\_arbitros[jornada[num\_partido].second][i];**

**backMaxArbitros (jornada , cant\_partidos , num\_partido+1 , puntajes\_arbitros , puntaje\_actual , puntaje\_maximo , arbitros\_actuales , arbitros);**

**arbitros\_actuales.pop\_back();**

**puntaje\_actual -= puntajes\_arbitros[ jornada[num\_partido].first][i] + puntajes\_arbitros[jornada[num\_partido].second][i];**

**}**

**}**

**}**

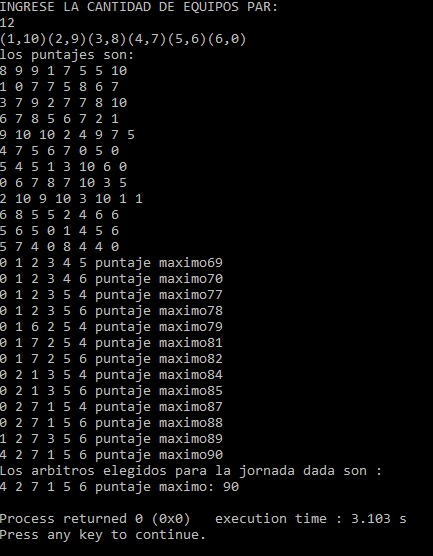
En esta última versión se modificó el cargado de la cantidad de equipos de manera dinámica, se ingresa por teclado, pero el ingreso de numero de árbitros se mantiene estático, por errores de memoria en el heap.

**PODA:** En este backtracking, esta función booleana, consiste en no elegir un árbitro para determinado partido si ya fue elegido anteriormente.

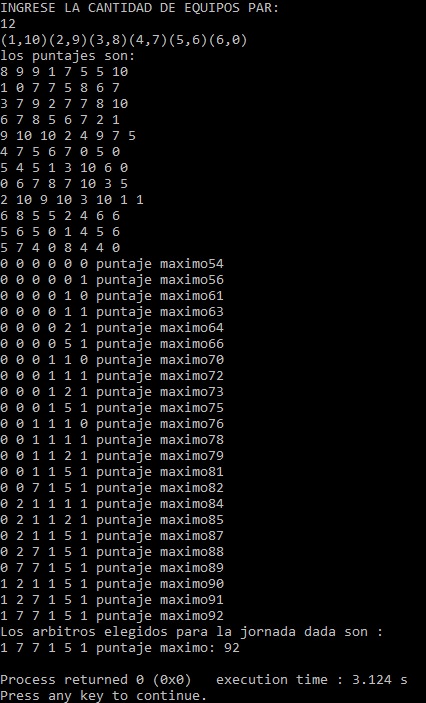
**COMPARACIONES:** A continuación, vamos a comparar la cantidad de veces que el programa se ejecuta teniendo en cuenta dos casos: **con poda y sin poda.**

Ingresando la misma cantidad de árbitros y de equipos en ambas ejecuciones para comparar los resultados.

* **CON PODA:** En la siguiente imagen observamos que realiza una iteración de 13 veces con un puntaje máximo de 90 y no se repiten árbitros.

****

* **SIN PODA:** En la siguiente imagen observamos que realiza una iteración de 23 veces, además se repiten los árbitros en la misma jornada y el puntaje máximo obtenido es de 92.

****

Finalmente viendo esta comparación podemos concluir que los resultados sin poda no cumplen con el problema, debido a que un mismo árbitro no puede dirigir dos partidos en una misma fecha.

Por otro lado, sin poda solo se incrementa por 2 puntos el puntaje máximo, pero se itera 10 veces más que poda, aumentando así el tiempo de ejecución del algoritmo.

**CONCLUSIÓN:**

Se tine un listado de arbitros\_actuales, y una variable entera cant\_partidos, y el objetivo es maximizar el puntaje de una jornada. Lo que se debe entender es que a un partido no se le puede asignar más de un árbitro, y ademas los árbitros solo pueden dirigir un partido en esa fecha, y el puntaje máximo saldrá de las puntuaciones que los equipos asignaron a los árbitros. Entonces de la matriz que se genera con las puntaciones, **puntajes\_arbitros,**  se toma el mayor puntaje entre los dos equipos y de toda la jornada.

Por lo tanto la complejidad temporal pertenece a O(max(P, A))

Siendo P la cantidad de partidos y A la cantidad de árbitros.

**BIBLIOGRAFIA:**

* [**https://es.wikipedia.org/wiki/Sistema\_de\_todos\_contra\_todos**](https://es.wikipedia.org/wiki/Sistema_de_todos_contra_todos)
* **Filminas de la cátedra, análisis y diseños de algoritmos II. Virginia Cifuentes.**