# Proyecto final - Problema CalDep
# Análisis y diseño de algoritmos I

1st Cristián David Zúñiga Gutierrez
*202259425 - 2724*

2nd Alejandro Marín Hoyos
*202259353 - 3743*

3rd Juan José Marín Arias
*202259337 - 3743*

## I. Solución ingenua

### A. función "backtrack"

La función **"backtrack"** implementa un algoritmo de backtracking para encontrar combinaciones óptimas de partidos. Esta función esta diseñada para explorar diferentes configuaciones de partidos, calculando costos de viaje y manteniendo un registro de los equipos que han jugado.
Su funcionamiento consta de un **caso base** donde si "n" es $\mathcal{O}$, significa que se ha llegado a una configuración completa, luego se verifica si esta solución es valida y de ser así se agrega a "rs". Si aún quedan decisiones por tomar ("$n$" $>$ **O**), la funcion itera sobre cada equipo "team" y explora las posibles combinaciones de partidos, llamandose recursivamente para este fin. Cuando se toma una decisión (por ejempllo, programar un partido entre dos equipos), se actualizan varios estados como: "played", `"current_post"`, `"current_post"`, "travel" y se modifica `"remanning_matches"` para reflejar esta decisión. Después de explorar una rama de decisiones, la función "retrocede", deshaciendo las últimas decisiones y restaurando el estado anterior para explorar diferentes ramas de decisiones. Esto se logra mediante la reversión de los cambios en las variables de estado y la eliminación de las últimas decisiones de "tmp", finalmente una vez que se han explorado todas las posibilidades, la función retorna "rs", que contiene todas las soluciones válidas encontradas.

### B. función `"find_all"`

La función **`"find_all"`** se encarga de encontrar la solución óptima para el calendaario, llamando a **"backtrack"** para cada semana del calendario y manteniendo un registro del costo total óptimo y la solución óptima encontrada hasta el momento. Recibe como parámetros un "N" que es el número total de equipos, un "K" que es el contador de las semanas del calenadrio, "res" que es una lista para almacenar las combinaciones de partidos y "tmp" que es una lista temporal para lamacenar información mientras se exploran combinaciones de partidos.
La función comienza verificando si ha alcanzado el final de la programación (todas las semanas han sido programadas). Si se ha llegado al final y el costo reciente de la configuración `"f_recent"` es mejor que el óptimo encontrado hasta el momento `"f_opt"`, se actualiza la solución óptima. De lo contrario la función incrementa el contador de semanas "k" y llama a la función "backtrack" para encontrar todas las posibles configuraciones de partidos para la semana actual, este proceso se realiza recursivamente para cada semana hasta que todas las semanas han sido programadas. Dentro de cada llamada a "backtrack", se explora el espacio de soluciones para esa semana en particular, probando diferentes combinaciones de partidos y calculando sus costos asociados, para cada configuración válida devuelta por "backtrack", la función `find_all` actualiza el costo reciente `"f_recent"` y agrega la configuración a la lista de resultados "res". Luego, realiza llamadas recursivas a sí misma para continuar construyendo la solución en semanas subsiguientes. Después de explorar una configuración, la función retrocede, es decir, revierte los cambios realizados en el estado del programa (como las posiciones actuales y previas de los equipos) y retira la última configuración de partidos de la lista de resultados "res", esto permite que la función explore otras configuraciones de partidos alternativas en las semanas siguientes, finaalmente una vez exploradas todas las configuraciones para todas las semanas, la función devuelve la mejor solución encontrada (solution) junto con su costo `"f_opt"`.

### C. Análisis temporal y espacial

Para el análisis temporal tenemos que:

- La lectura del archivo tiene una complejidad de $\mathcal{O}(n^2)$ donde n es el número de equipos. Cada línea después de la primera se lee y se procesa en $\mathcal{O}$(n), y esto se repite n veces.
- La complejidad de la función "backtrack" es de $\mathcal{O}(2^n)$ ya que la función itera sobre cada equipo y realiza backtracking para los partidos restantes.
- La función `"find_all"` tiene una complejidad tempral similar a la función "backtrack" ya que itera sobre todas las posibles semanas (hasta 2n - 2) y llama a "backtrack". La complejidad es exponencial y posiblemente mayor a $\mathcal{O}(2^n)$ dado a que itera en el número de equipos y el número de semanas.
- Para la complejidad temporal de la construcción de la matriz sera de $\mathcal{O}(n*w)$ donde n es el número de equipos y W es el número de semanas, ya que se iteran todas las semanas y todos los equipos para cada semana.

Para el análisis espacial tenemos que:

- Se inicializan varias listas de tamaño n y su complejidad espacial sera de $\mathcal{O}(n)$ para cada una totalizando una

complejidad de $\mathcal{O}(n)$ ya que crecen linealmente con el número de equipos y un¡ complejidad de $\mathcal{O}(2^n)$ para la matriz.

- La función "backtrack" Utiliza espacio adicional para el conjunto "tmp" y la recursión, el espacio utilizado depende del número de llamadas recursivas y el tamaño del conjunto "tmp", que puede crecer con el número de equipos. La complejidad espacial puede ser significativa, especialmente debido a la profundidad de la pila de llamadas en la recursión.
- La función `"find_all"` tiene una complejidad similar a backtrack, pero se agrega la lista "res" para mantener las soluciones. La complejidad espacial es alta debido a la naturaleza recursiva y al almacenamiento de soluciones intermedias.
- Para la complejidad espacial del almacenamiento de la matriz sera de $\mathcal{O}(n * w)$ donde n es el número de equipos y W es el número de semanas, ya que esta matriz almacena los resultados de todos los partidos para cada semana.

Al centrarnos en las dos funciones principales vamos a encontrar que la complejidad computacional es de $\mathcal{O}(n!)$ ya que se deben explorar todas las posibles permutaciones de los n equipos y encontramos que hay factorial de n formas de ordenar los n equipos.

### *D. Análisis de cercanía al optimo*

Teniendo en cuenta que una solución óptima es aquella que minimiza el costo total de viajes bajo las restricciones dadas. Teóricamente el algoritmo tiene la capazidas de encontrar una solución optima ó cerca al óptimo porque explora exhaustivamente todo el espacio de soluciones. Sin embargo, la eficiencia de este enfoque es una preocupación significativa, especialmente para un gran número de equipos, ya que la complejidad temporal del algoritmo es muy alta.
En el caso ideal, donde el tiempo y los recursos no son limitantes, el algoritmo debería ser capaz de encontrar la solución óptima, ya que no deja ninguna configuración sin explorar, pero en la práctica, es posible que el programa no pueda completar su ejecución en un tiempo razonable para instancias grandes debido a su complejidad exponencial, en tales casos, no podemos garantizar que la solución encontrada sea la óptima o incluso esté cerca de ella.
En resumen podemos decir que el algoritmo tiene la capacidad de encontrar la solución óptima o cerca al óptimo siempre y cuando la cantidad de equipos sea prudente, pues en instancias muy grandes el algoritmo pierde eficiencia.

### *E. Descripción y análisis de las pruebas realizadas*

Para la primer prueba donde n = 4, Min =1, Max = 3,

$$D = \begin{bmatrix} 0 & 745 & 665 & 929 \\ 745 & 0 & 80 & 337 \\ 665 & 80 & 0 & 380 \\ 929 & 337 & 380 & 0 \end{bmatrix}$$

Obtuvimos la salida esperada que corresponde a:

$$Cal = \begin{bmatrix} 3 & 4 & -1 & -2 \\ 2 & -1 & 4 & -3 \\ 4 & -3 & 2 & -1 \\ -3 & -4 & 1 & 2 \\ -2 & 1 & -4 & 3 \\ -4 & 3 & -2 & 1 \end{bmatrix}$$

Con un costo total de 8276 y un tiempo de ejecución de 0.8562 segundos.

Para la primer prueba donde n = 6, Min =1, Max = 5,

$$D = \begin{bmatrix} 0 & 184 & 222 & 177 & 216 & 231 \\ 184 & 0 & 45 & 123 & 128 & 200 \\ 222 & 45 & 0 & 129 & 121 & 203 \\ 177 & 123 & 129 & 0 & 46 & 83 \\ 216 & 128 & 121 & 46 & 0 & 83 \\ 231 & 200 & 203 & 83 & 83 & 0 \end{bmatrix}$$

Obtuvimos la salida esperada que corresponde a:

$$Cal = \begin{bmatrix} 5 & 3 & -2 & -6 & -1 & 4 \\ 4 & -6 & -5 & -1 & 3 & 2 \\ 3 & -4 & -1 & 2 & 6 & -5 \\ 2 & -1 & 6 & 5 & -4 & -3 \\ 6 & 5 & 4 & -3 & -2 & -1 \\ -4 & 6 & 5 & 1 & -3 & -2 \\ -5 & -3 & 2 & 6 & 1 & -4 \\ -6 & -5 & -4 & 3 & 2 & 1 \\ -2 & 1 & -6 & -5 & 4 & 3 \\ -3 & 4 & 1 & -2 & -6 & 5 \end{bmatrix}$$

Con un costo total de 8276 y un tiempo de ejecución de 45 minutos con 21.5086 segundos.

Para la primer prueba donde n = 8, Min =1, Max = 7,

$$D = \begin{bmatrix} 0 & 184 & 222 & 177 & 216 & 231 & 120 & 60 \\ 184 & 0 & 45 & 123 & 128 & 200 & 52 & 100 \\ 222 & 45 & 0 & 129 & 121 & 203 & 15 & 300 \\ 177 & 123 & 129 & 0 & 46 & 83 & 250 & 15 \\ 216 & 128 & 121 & 46 & 0 & 83 & 100 & 7 \\ 231 & 200 & 203 & 83 & 83 & 0 & 20 & 10 \\ 120 & 52 & 15 & 250 & 100 & 20 & 0 & 441 \\ 60 & 100 & 300 & 15 & 7 & 10 & 441 & 0 \end{bmatrix}$$

Obtuvimos la salida esperada que corresponde a:

$$Cal = \begin{bmatrix} 7 & 5 & 4 & -3 & -2 & -8 & -1 & 6 \\ 6 & 3 & -2 & -8 & -7 & -1 & 5 & 4 \\ 2 & -1 & -7 & -6 & -8 & 4 & 3 & 5 \\ 3 & -8 & -1 & -7 & -6 & 5 & 4 & 2 \\ 4 & -6 & -5 & -1 & 3 & 2 & 8 & -7 \\ 8 & -7 & -6 & -5 & 4 & 3 & 2 & -1 \\ -5 & -4 & -8 & 2 & 1 & 7 & -6 & 3 \\ -7 & -5 & -4 & 3 & 2 & 8 & 1 & -6 \\ -6 & -3 & 2 & 8 & 7 & 1 & -5 & -4 \\ -2 & 1 & 7 & 6 & 8 & -4 & -3 & -5 \\ -3 & 8 & 1 & 7 & 6 & -5 & -4 & -2 \\ -4 & 6 & 5 & 1 & -3 & -2 & -8 & 7 \\ -8 & 7 & 6 & 5 & -4 & -3 & -2 & 1 \\ 5 & 4 & 8 & -2 & -1 & -7 & 6 & -3 \end{bmatrix}$$

Con un costo total de 8276 y un tiempo de ejecución de mas de 2 horas.

### *F. Análisis de los resultados y conclusiones*

Análisando los resultados obtenidos podemos observar el crecimiento exponencial con resecto al tiempo cuando aumentamos la cantidaad de equipos, confirmando la complejidad teorica calculada anteriormente, esto demuestra que en un escenario real esta solución no es la mas eficiente.

La implementación tiene sus fortalezas pues suponiendo recursos ilimitados el backtraking nos garantiza una exploración exhaustiva del espacio de soluciones, lo que teóricamente siempre permite encontrar la solución óptima. Tambien la estructura del código nos permite adaptarlo a diferentes escenarios de programación de partidos con distintas restricciones y requisitos. Aunque tambien se deben considerar sus debilidades pues la principal limitación es su complejidad computacional exponencial, lo que hace que el programa sea poco práctico para ligas con un gran número de equipos o para escenarios donde el tiempo de ejecución es un factor crucial. Esta no es la solución mas optimizada dado que no se implementan técnicas avanzadas de optimización como podas o algoritmos heurísticos, que podrían mejorar significativamente la eficiencia del algoritmo.

Por ultimo como aspecto a mejorar se podria considerar la implementación conjunta de otro algoritmo para garantizar la aproximación al óptimo.