# Tabajo practico entregable

## Introduccion

En este informe se explicaran los detalles, inconvenientes y todo el progreso obtenido durante la realizacion de un ejercico resuelto aplicando los algoritmos Greedy y Backtracking.

La problematica planteada en el enunciado del trabajo practico fue la siguiente. Una biblioteca dispone de un conjunto de libros cada uno identificado por un codigo unico de libro, algunos libros disponen de mas de un ejemplar. Los alumnos tienen que leer libros para poder aprobar una asignatura pero no pueden leer dos veces el mismo libro, la asignatura tiene una nota de aprobación que los alumnos deben alacanzar leyendo libros para poder aprobar.

El objetivo es que los libros de la biblioteca se asignen a los alumnos y lograr que aprueben la asignatura la mayor cantidad posible de alumnos cumpliendo con las restricciones del problema.

A continuacion se explica el paso a paso y los detalles de las dos tecnicas utilizadas para encontrar una solucion al problema.

## Greedy

El algoritmo greedy debe tomar la mejor decision en un momento dado con la esperanza de que cuando se terminen de tomar todas las decisiones posibles se llegue a la mejor solucion general, no siempre se encuentra la mejor solucion general de un problema ya que al tomar una decision solo se evaluan las opciones a “corto plazo” y esta decision tomada no se vuelve a considerar. El uso de greedy para resover un problema tiene como objetivo encontrar una solucion de la forma mas eficiente y buscar la optimizacion.

Inicialmente el planteo fue seleccionar pares de candidatos pero la idea de generar todos los pares alumno-libro posibles resulto poco comprensible. Tambien se considero la idea de tomar un libro como candidato, pero se descarto la idea porque por la implementacion realizada, finalizaba con un numero muy alto de iteraciones.

Finalmente los candidatos fueron los alumnos, el criterio para seleccionar un alumno es evaluarlos en un orden determinado para optimizar la seleccion, primero se ubican los que tienen la nota mas alta. Una vez seleccionado se le asignan libros hasta que apruebe o que no queden libros que pueda leer. Al seleccionar un libro para el alumno se toman en cuenta primero los de mayor puntaje, en caso de que el alumno alcanze la nota de aprobacion se busca que apruebe de la forma menos holgada posible.

**Proceso de seleccion del libro para el alumno**

Se accede a los libros por medio de un indice que se manipula para avanzar y recorrer la lista de libros. Lo que se busca es evaluar los libros a partir de este indice, sumando el puntaje del alumno mas el puntaje de un libro.

Imaginando la situacion de que un alumno solo necesita 10 puntos para aprobar y el primer libro tiene 100 puntos, esta seleccion lograria hacer aprobar al alumno, pero quizas existe un libro que sume los 10 puntos que necesita el alumno y esta sería la mejor seleccion para poder guardar el libro de 100 puntos para otro alumno que requiera de mas puntaje para aprobar.

Se fuerza a entrar en el bucle inicializando con valor infinito a la variable **suma** que luego almacenara la suma de el puntaje del alumno y el de un libro seleccionado.

Si el primer libro en la lista que puede leer el alumno (el de mayor puntaje) **no** alcanza para que el alumno apruebe la asignatura, este bucle termina luego de la primer iteracion y retorna la posicion dentro de la lista donde se encuentra este libro.

Si el primer libro en la lista que puede leer el alumno (el de mayor puntaje) **si** alcanza para que el alumno apruebe la asignatura, el bucle continua iterando hasta encontrarse con una suma que no haga aprobar al alumno o hasta el final de la lista, dentro del bucle se va guardando la posicion del ultimo libro que hizo aprobar al alumno, el libro que lo hace aprobar de la forma menos holgada posible.

seleccionarLibro(indice)

suma = infinito

while (indice no termina la lista Y alumno aprueba la asignatura)

if (alumno puede leer el libro )

suma = puntaje del alumno + puntaje del libro

if (alumno aprueba la asignatura)

posicion de retorno = indice

indice++;

return posicion de retorno;}

**Optimizacion del salto**

Como la biblioteca puede tener miles de libros se tomaron medidas para optimizar la seleccion del libro, esta implementacion se llama ***optimizacion del salto***, en el peor de los casos se evaluan todos los libros, pero en la mayoria de las situaciones y especialmente cuando hay gran numero de libros, se reduce de manera considerable la cantidad de iteraciones sobre la lista de libros sin influir negativamente de forma notoria en el resultado final de la asignacion de libros a un alumno.

El indice ya no se se incrementa de uno en uno sino que da un salto, en el caso de que todos los libros logren hacer aprobar al alumno se daran 10 saltos como maximo porque estos saltos se calculan por el 10% del total de los libros disponibles.

Imaginando la situacion de que hay 1000 libros disponibles y todos estos hacen que el alumno apruebe, si el indice va incrementando de uno en uno se realizan 1000 iteraciones, en cambio con la optimizacion del indice que salta solo se realizan 10 iteraciones.

seleccionarLibro(indice)

suma = infinito

while (indice no termina la lista Y alumno aprueba la asignatura)

if (alumno puede leer el libro )

suma = puntaje del alumno + puntaje del libro

if (alumno aprueba la asignatura)

posicion de retorno = indice

**indice salta 1/10 del total de libros.**

else

indice++;

return posicion de retorno;}

Otro elemento de optimizacion fue no generar una instacia de Libro diferente para cada ejemplar, sino que una unica instancia del libro sabe cuantos ejemplares tiene y se van restando a medida que se asignan a los alumnos.

Considerando que la consigna es maximizar la cantidad de alumnos aprobados y no dice explicitamente que la solucion es que aprueben todos, entiendo que hay tres posibles condiciones de finalizacion del algoritmo. La primera condicion de corte es que apueben todos los alumnos, la segunda es que se asignen todos los libros, la tercera es que no aprueben todos los alumnos pero al mismo tiempo los libros que queden disponibles no puedan ser asignados debido a que ya fueron leidos por todos los alumnos.

Es factible agregar un candidato a la solucion cuando a este candidato (alumno) se le asigna un libro, teniendo en cuenta que el algoritmo debe informar que libros se deben asignar a cada alumno.

## Backtacking

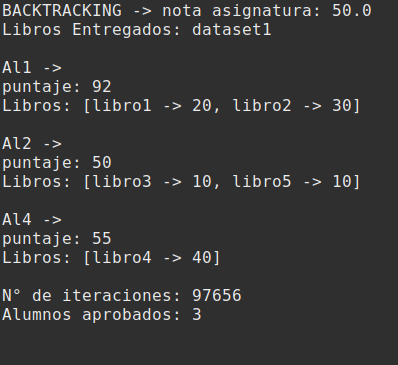
El algoritmo backtracking evalua todos los resultados posibles de un problema dado e ira acumulando la mejor solucion o soluciones del problema, en caso de no existir una solucion este algoritmo pasara por todas las combinaciones posibles de decisiones porque a diferencia de greedy tiene la posibilidad de volver a cosiderar una decision tomada.

Con cada decision tomada se genera un nuevo estado, este estado se compone del conjunto de alumnos evaluados con los libros que le fueron asignados, se considera un estado final cuando ya no hay mas libros para asignar a los alumnos ya sea porque no quedan libros o porque los alumnos no pueden leer los libros disponibles. El estado solucion es cuando el estado en tiene mas aprobados que estado solucion anterior. Los hijos de un estado son todas las decisiones tomadas a partir de este estado hasta llegar a los diferentes estados finales posibles. Para generar un estado hijo se debe modificar el estado actual asignandole un libro a un alumno.

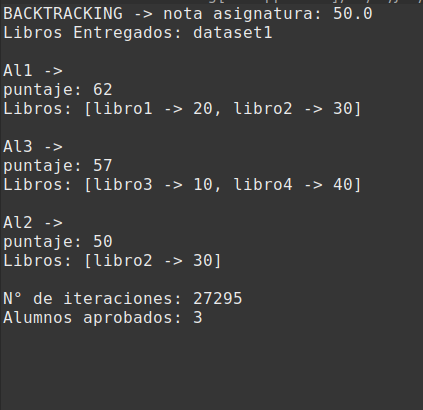
La poda se realiza para no generar ramas innecesarias en el arbol de recursion. La poda se hace efectiva cuando un alumno ya acumulo los libros suficientes para aprobar la asignatura, tambien resuelve las restricciones del problema como por ejemplo el libro que no tiene mas ejemplares y el alumno que no puede leer un libro.

En las pruebas realizadas se puede ver una diferencia en la eficiencia a la hora de la ejecucion del codigo.

**Backtracking sin poda:**



**Backtracking con poda:**

****

##### Resultados dataset1.csv

Nota para aprobar asignatura: **30**

El contador finaliza con un total de 20 iteraciones.

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **50**

El contador finaliza con un total de 23 iteraciones.

**Resutado:** Aprueban 3 alumnos.

Nota para aprobar asignatura: **100**

El contador finaliza con un total de 30 iteraciones.

**Resutado:** Aprueba 1 alumno.

##### Resultados dataset2.csv

Nota para aprobar asignatura: **30**

El contador finaliza con un total de 39 iteraciones.

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **50**

El contador finaliza con un total de 34 iteraciones.

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **100**

El contador finaliza con un total de 27 iteraciones.

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **200**

El contador finaliza con un total de 45 iteraciones.

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **500**

El contador finaliza con un total de 110 iteraciones.

**Resutado:** Aprueban 3 alumnos.

##### Resultados dataset3.csv

Nota para aprobar asignatura: **500**

El contador finaliza con 101 iteraciones. \*(sin optimizacion del salto son 2654)

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **1000**

El contador finaliza con 166 iteraciones. \*(sin optimizacion del salto son 2018)

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **2000**

El contador finaliza con 346 iteraciones. \*(sin optimizacion del salto son 3430)

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **5000**

El contador finaliza con 800 iteraciones. \*(sin optimizacion del salto son 3135)

**Resutado:** Aprueban 5 alumnos.

Nota para aprobar asignatura: **20000**

El contador finaliza con 4441 iteraciones. \*(sin optimizacion del salto son 5280)

**Resutado:** Aprueban 4 alumnos.

Nota para aprobar asignatura: **30000**

El contador finaliza con 6343 iteraciones. \*(sin optimizacion del salto son 6547)

**Resutado:** Aprueban 2 alumnos.

## Conclusion

Backtracking asegura encontrar la mejor solucion pero el riesgo es que el espacio de busqueda sea demasiado grande y jamas finalize la ejecucion por el numero incalculable de posibles combinaciones, incluso con la optimizacion de la poda.

Greedy resulta muy practico ya que da una respuesta rapida a riesgo de que sea una respuesta mala, de todas formas el algoritmo puede dar buenas soluciones en ralacion “costo-calidad” hablando del costo computacional y la calidad de la respuesta.