Resumen

Introducción

Marco Teórico

Knapsack

Knapsack o el problema de la mochila, es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y valor especifico. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.

Clasificado en 1972 por Richard Karp, un informático teórico, entre los 21 problemas Np-completos, que más adelante en este informe explicaremos como llegamos a esta clasificación.

Si se trata de solucionar este problema de forma recursiva daría un O(2^n) pero si se usa un algoritmo iterativo se obtiene un O(nW).

Y por su forma de desarrollo es un algoritmo de Programación Dinámica.

Metodología

Knapsack

Partiendo de que este problema se puede plantear de dos maneras distintas (se toma todo el objeto o se toma parte del objeto para llenar la bolsa) nosotros usaremos en donde tomamos toda la cantidad de un solo objeto.

También existen dos maneras de resolver dicho problema, en este caso para nosotros es más importante el tiempo de ejecución que tendrá, aun con el conocimiento que de cualquier manera solo se llega a una aproximación, es por ello que usamos un algoritmo iterativo ya que es más eficiente en cuestión de tiempo al algoritmo recursivo.

Ilustración 1 Algoritmo Iterativo Knapsack

knapsack = () => {

var k = [];

for (var i = 0; i <= this.productList.length; i++) {

for (var j = 0; j <= this.weight; j++) {

if (i === 0 || j === 0) {

k[i][j] = 0;

} else if (this.productList[i - 1].amount <= j) {

k[i][j] = this.max(parseInt(this.productList[i - 1].value) + parseInt(k[i - 1][j - this.productList[i - 1].amount]), k[i - 1][j]);

} else {

k[i][j] = k[i - 1][j];

}

}

}

return k[this.productList.length][this.weight];

}

Como muestra el algoritmo en la Ilustración 1 donde se usa el tamaño de la lista de productos antes ingresada, así mismo el peso total que soporta la mochila, luego ya dentro de la función se crea una matriz donde se guardaran el máximo de cada iteración. Luego para iterar dentro de la matriz ya creada usamos dos For’s siendo el exterior con recorrido de la lista de valor monetario y el interno del máximo peso de la mochila previamente pedido.

Adentro de los dos for tenemos un if donde primero verificaremos el valor de i y de j poniendo en cero el primer elemento de la matriz igual a 0 y esto para poder después usar dicho valor en una función “max (a, b)”.

Sino y si el valor monetario en el arreglo con la posición j-1 es mayor o igual que j entonces usamos una función “max (a, b)”, que nos sirve para saber el mayor entre dos números, en el cual le enviamos como parámetro ‘a’ la suma del valor monetario y la cantidad y como ‘b’ el valor en la matriz en la posición [i-1][j]. Y la ponemos en la posición [i][j] de la matriz.

Sino no se cumple ninguno de los casos antes descritos entonces solo colocamos el valor en k[i-1][j] en la matriz con posición [i][j].

Y lo que retorna nuestra función es el elemento en la posición [tamaño del arreglo] [el peso de la mochila].

Es por eso que este algoritmo es tiempo O(nW), donde n es el número de productos y W el peso máximo de la mochila.

Resultados

Knapsack

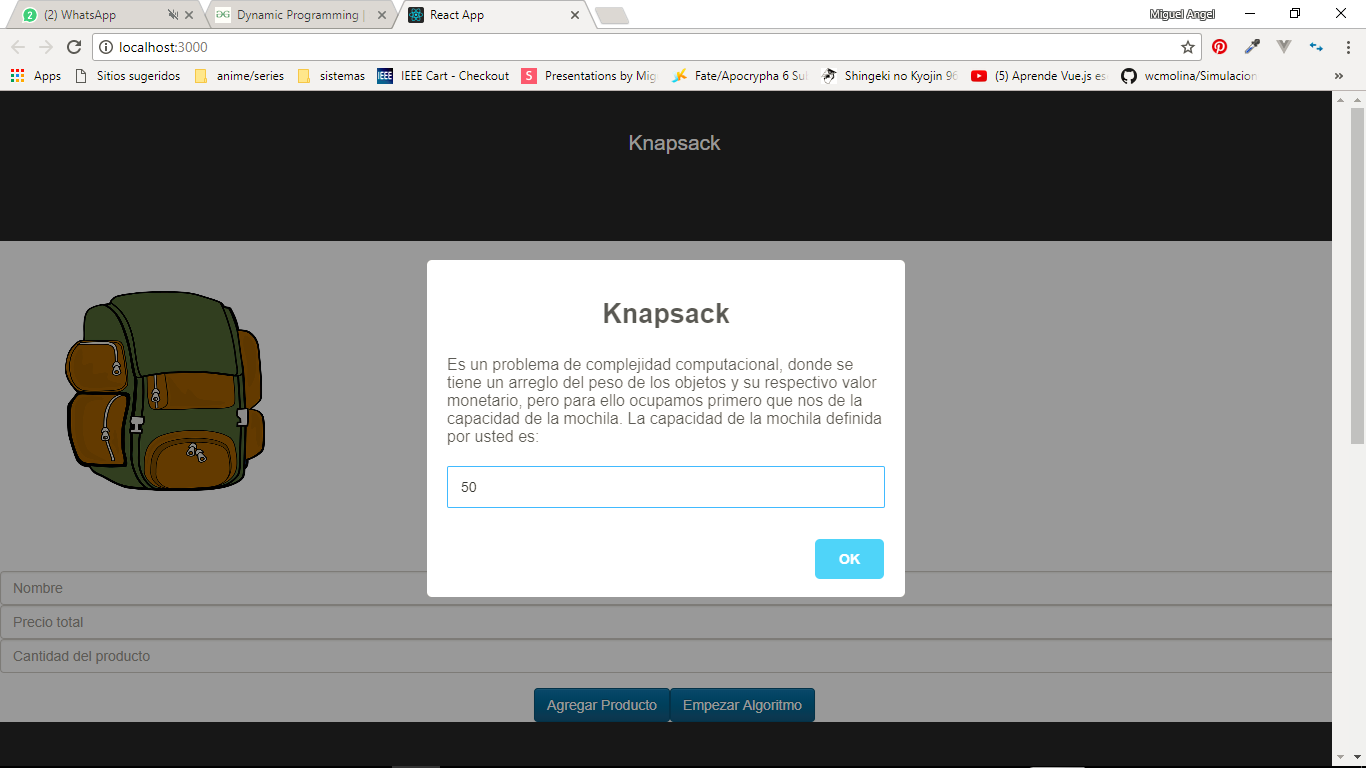


Ilustración 2 Inserción del peso de la mochila

Al cargar la página, explica el problema y pide el peso de la mochila deseada. Usaremos de ejemplo 50.

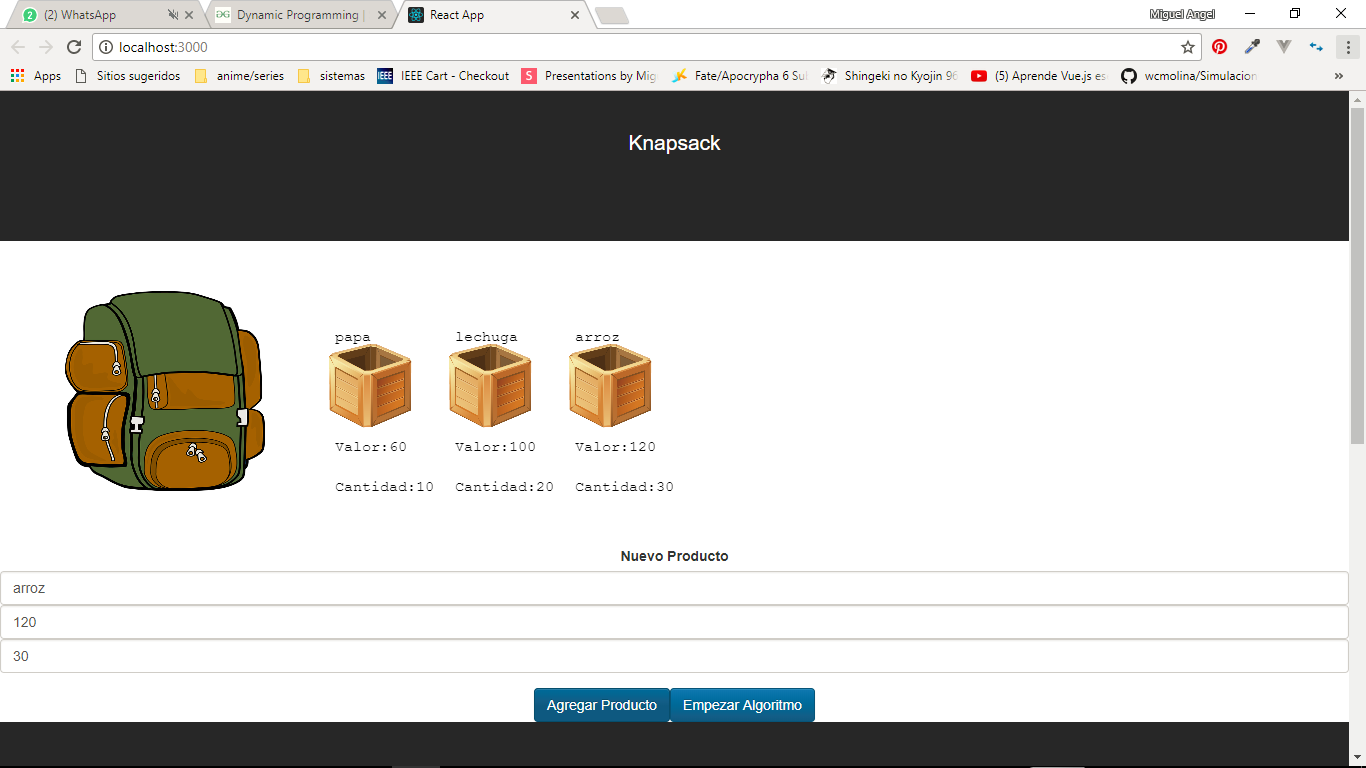


Ilustración 3 Elementos en la lista de productos

En este caso usaremos una lista pequeña para poder explicar y que se vea bien la ejecución del algoritmo en la Ilustración 1

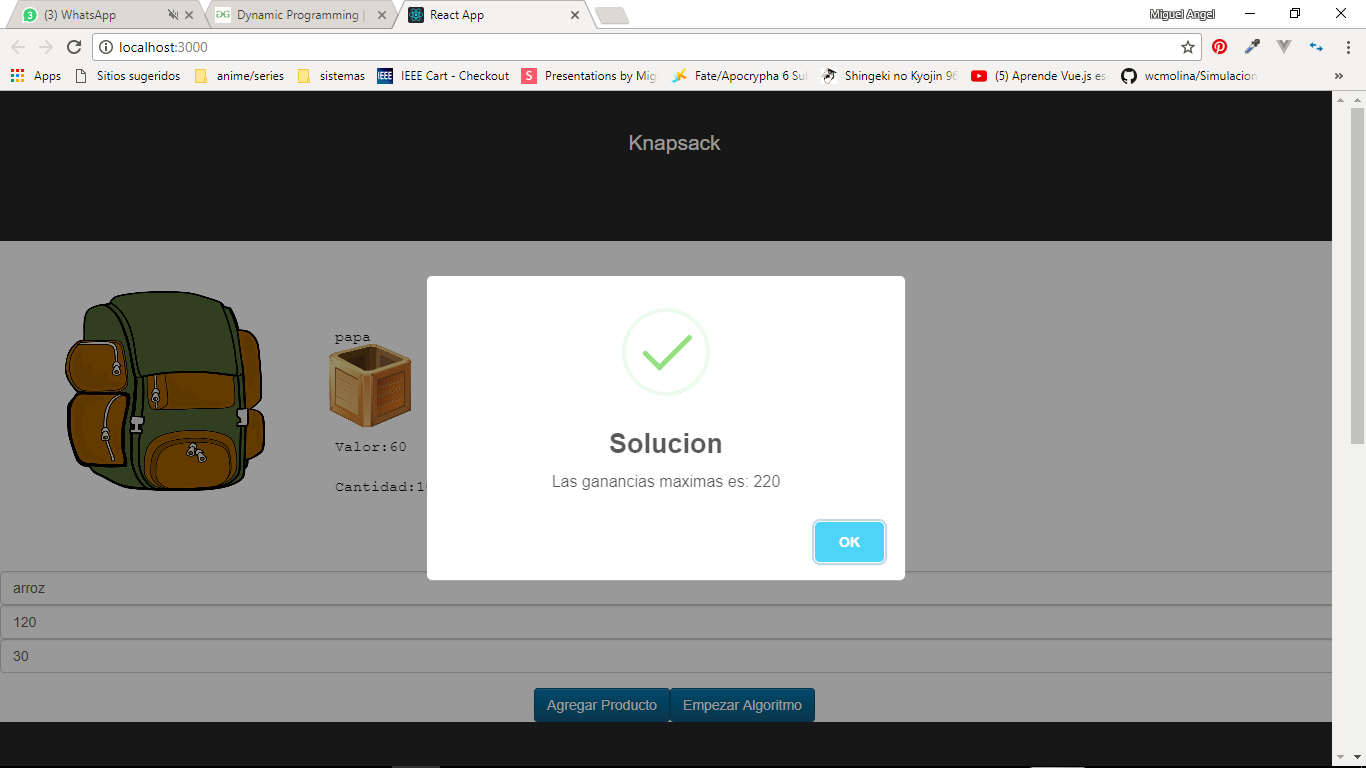


Ilustración Solucion de ejemplo

Como resultado del algoritmo nos retorna el 220 que es la máxima ganancia adquirida con la lista antes ingresada

Conclusiones y Recomendaciones

Knapsack

Como única recomendación y al mismo tiempo conclusión es que el algoritmo antes mostrado, con relación en tiempo de ejecución es más eficiente el iterativo aun cuando la lista es grande.