UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Algoritmos Avanzados

Laboratorio N°3: Pseudocódigo

Alumnos: Israel Arias Panez.

Christian Mendez Acosta.

Sección: A-1.

Santiago -
Chile 1-
2021

Pseudocódigo del programa de backtracking:

1. Se Solicita el nombre del archivo por teclado.

2. Se verifica si existe el dataset según el nombre entregado, en caso de no existir vuelve al paso 1.

3. Se almacena la capacidad máxima del container por el nombre del archivo.

4. Se almacena la cantidad de paquetes por el nombre del archivo.

5. Se abre el archivo dataset (archivo .csv).

Nota: La estructura que se utilizará y que se le hará referencia a lo largo del pseudocódigo es la siguiente:

```
struct Paquete{
      int id; //id del paquete
      int beneficio; //beneficio del paquete
      int volumen; //volumen que ocupa el paquete
      float ponderación; //resultado de dividir beneficio entre volumen
};
```

6. Lectura, validación y creación de paquetes, creación del arreglo inicial que contiene todos los paquetes leídos y validados.

```
Se crea un arreglo de paquetes iniciales vacío

Por cada línea en el archivo:
      Se lee volumen del Paquete de la línea (segundo dígito)
      Si volumen Paquete < Capacidad Container:
            Calcular ponderación -> beneficio/volumen
            Generar paquete según estructura
            Guardar paquete en arreglo inicial
Cerrar archivo
```

7. Se verifica en caso de que no se haya guardado ningún paquete válido.

```
Si largo arreglo inicial == 0:
       Mostrar por pantalla “No hay solución válida”
Fin de Programa
```

8. Realizar un ordenamiento por **Quicksort** de la lista de paquetes de mayor a menor según su ponderación, usando la función nativa qsort(); disponible en la librería estándar de C <stdio.h>.

9. Se inicializan dos arreglos los cuales serán ocupados dentro del backtracking.

Se inicializan dos arreglos con la misma cantidad de espacio que el largo del arreglo de paquetes: mochila y mejorMochila

Se define una variable auxiliar i = 1

10. Se realiza el backtracking con poda para determinar la solución. **(nota: para entender mejor el algoritmo ver traza de ejemplo de programa al final del documento)**

```
Si i > largoArregloInicial:  //Condición de salida de la recursión
      Para cada elemento dentro de mochila:
            Insertar elemento dentro de mejorMochila
      retornar mejorMochila

Si pesoActual + peso del item i del arreglo inicial <= capacidad Total:
      Se agrega el item al arreglo mochila
      pesoActual = pesoActual + pesoItem
      beneficioActual = beneficioActual + beneficioItem

      Se llama recursivamente a esta función backtracking,
      aumentando en 1 el valor de i            // backtracking(i+1)

      Se quita el item del arreglo mochila
      pesoActual = pesoActual - pesoItem
      beneficioActual = beneficioActual - beneficioItem

Si posibleBeneficio al añadir próximo ítem > beneficioActual:
      Se quita el item del arreglo mochila

      Se llama recursivamente a esta función backtracking,
      aumentando en 1 el valor de i            // backtracking(i+1)
```

Nota: posibleBeneficio se calcula de la siguiente manera:

posibleBeneficio = beneficioActual + capacidadRestante * Pi

(Pi = Ponderación del item que se desea agregar)

11. Se escribe un archivo de salida .txt que posee la información de la solución encontrada, la cual fue determinada en el paso 9 según el índice de tope de la lista de paquetes calculado, el cual no excede el volumen máximo del container. Se escribirán los paquetes de la solución encontrada, incluyendo el beneficio y el volumen que ocupa.

```
Abrir archivo con nombre salida.txt

Se inicializan las variables beneficioTotal  y volumenTotal ambas en cero

Para cada elemento de mejorMochila:
      Leer numero de id del elemento actual
      Escribir “Paquete” + id en archivo
      beneficioTotal = beneficioTotal + beneficio elemento
      volumenTotal = volumenTotal + volumen elemento

Escribir “Beneficio: ” + beneficioTotal en archivo
Escribir “Volumen: “ + volumenTotal en archivo
Cerrar archivo

Fin de Programa
```

Ejemplo de traza del algoritmo:

Si existen los ítems:

I1: Beneficio = 9|Vol = 3 I2: Beneficio = 10|Vol= 5 I3: Beneficio = 7|Vol= 2 I4: Beneficio = 4|Vol = 1

Y el container tiene una capacidad máxima de 9 kg:

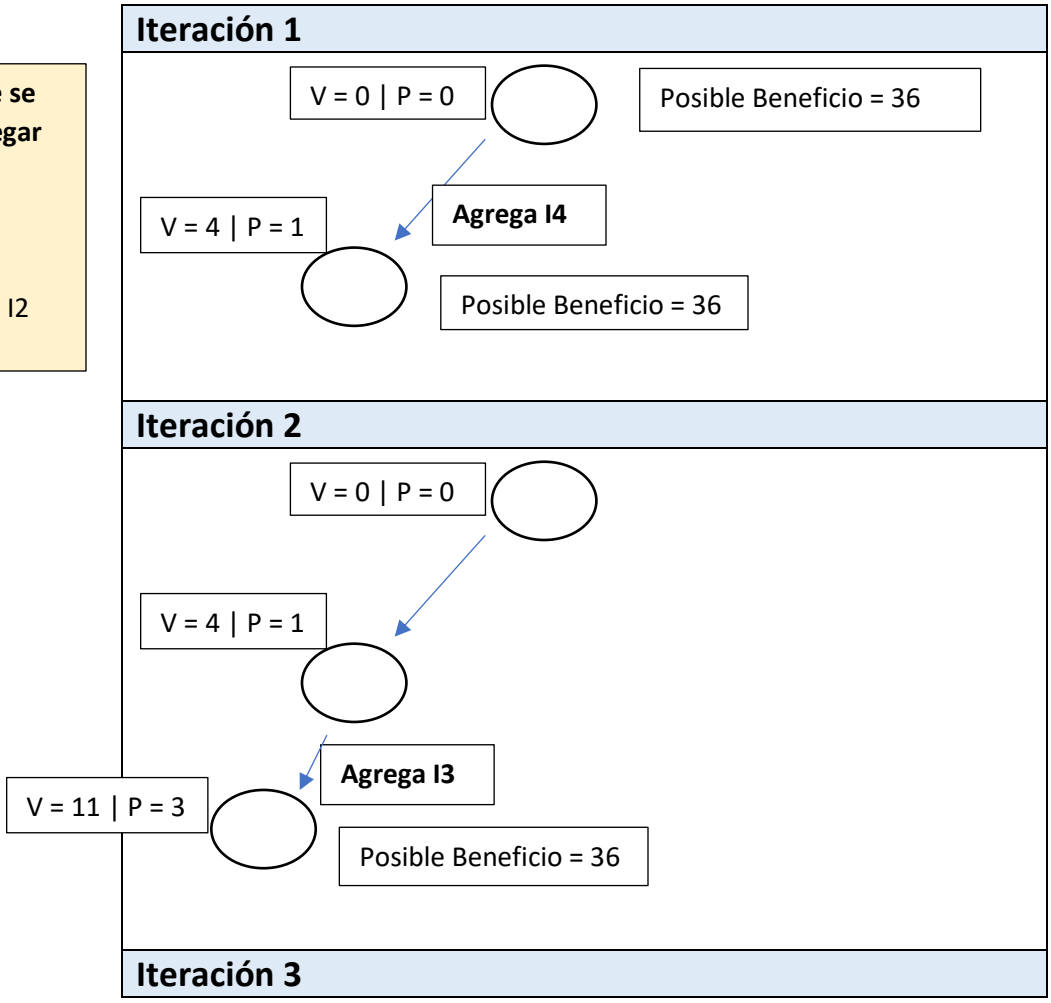
Se ordenan de mayor a menor los ítems según su ponderación beneficio/volumen

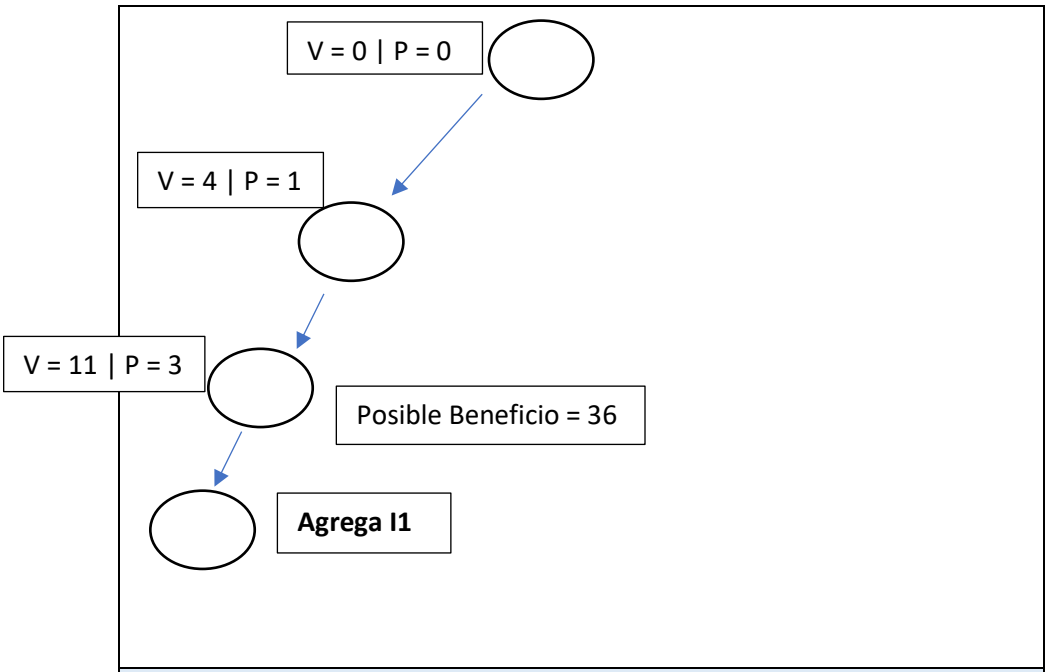
Ordenados: I4 -> I3 -> I1 -> I2

A continuación, se efectúa el algoritmo de backtracking

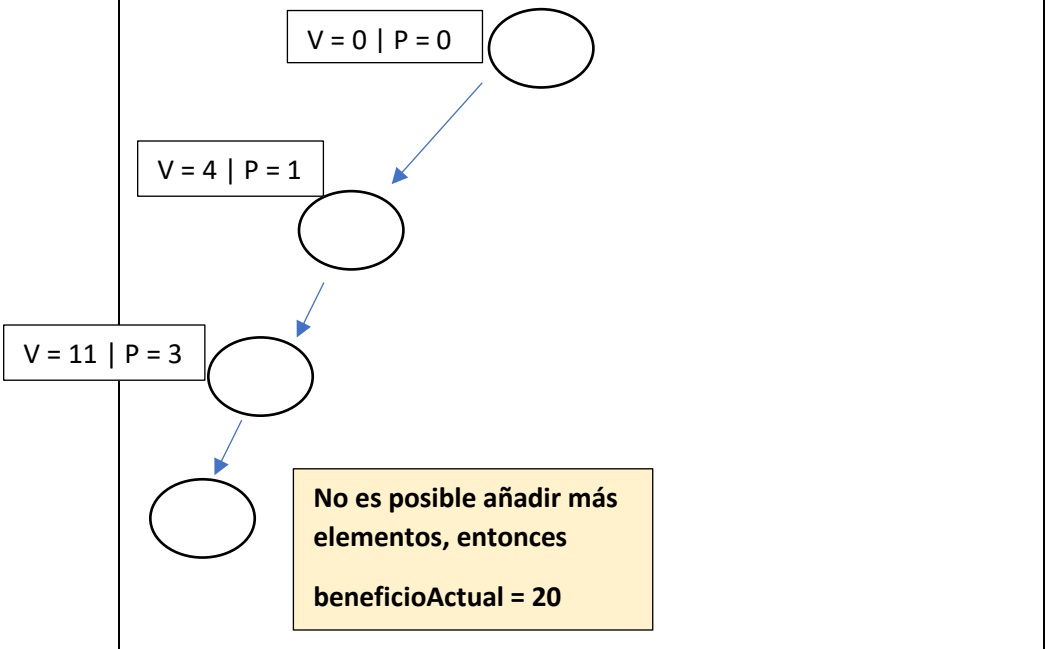
Nota: siempre se intentará agregar de acuerdo al orden de los ponderados:

I4 -> I3 -> I1 -> I2

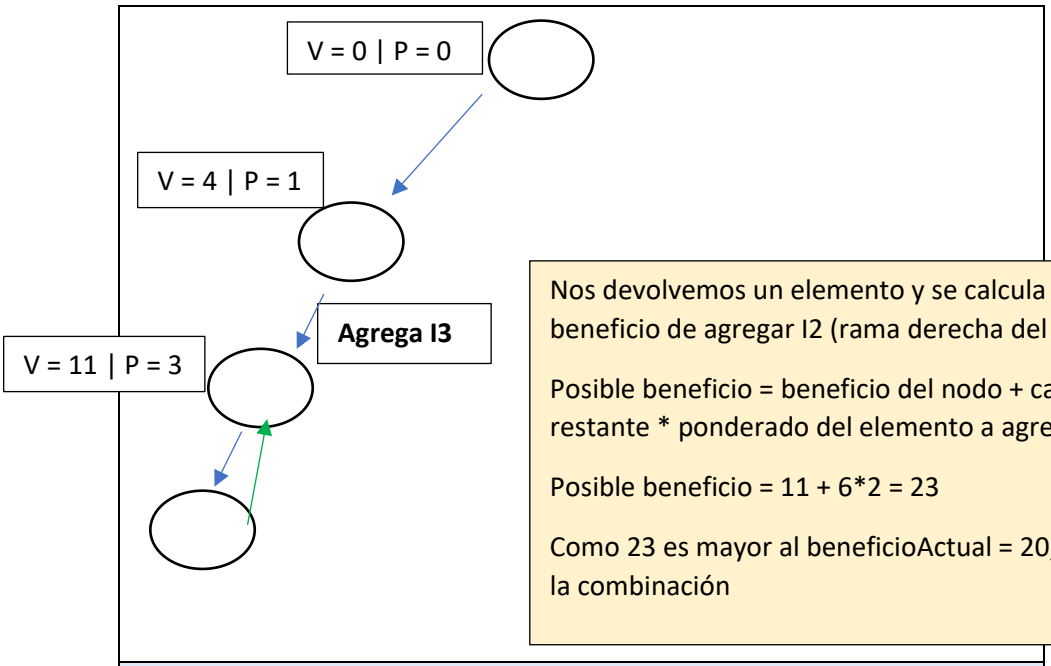




Iteración 4



Iteración 5



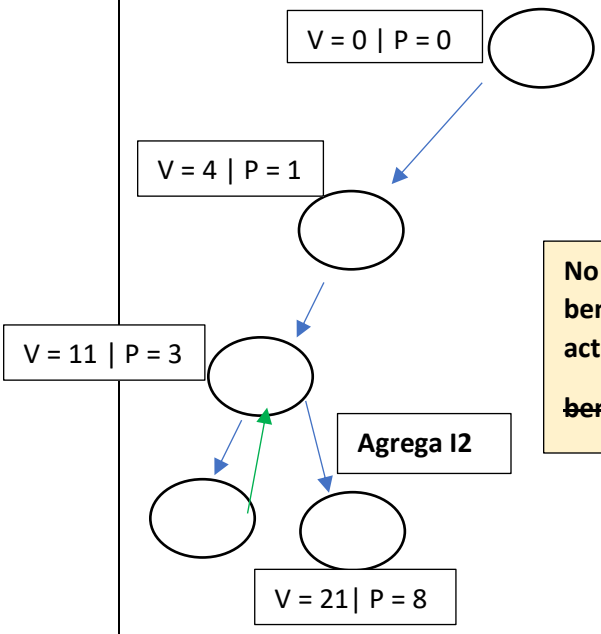
Nos devolvemos un elemento y se calcula el posible beneficio de agregar I2 (rama derecha del árbol)

Posible beneficio = beneficio del nodo + capacidad restante * ponderado del elemento a agregar

Posible beneficio = 11 + 6*2 = 23

Como 23 es mayor al beneficioActual = 20, se construirá la combinación

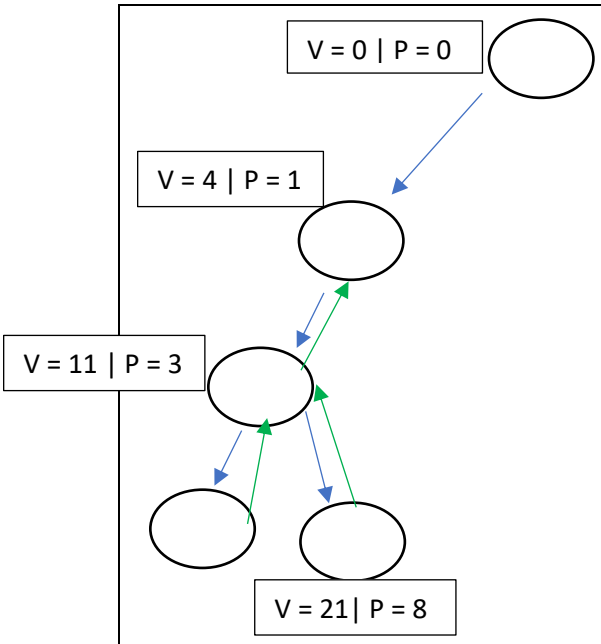
Iteración 6



No es posible añadir más elementos, además beneficioNodo > beneficioActual, entonces se actualiza el beneficioActual

~~**beneficioActual = 20**~~ **beneficioActual = 21**

Iteración 7

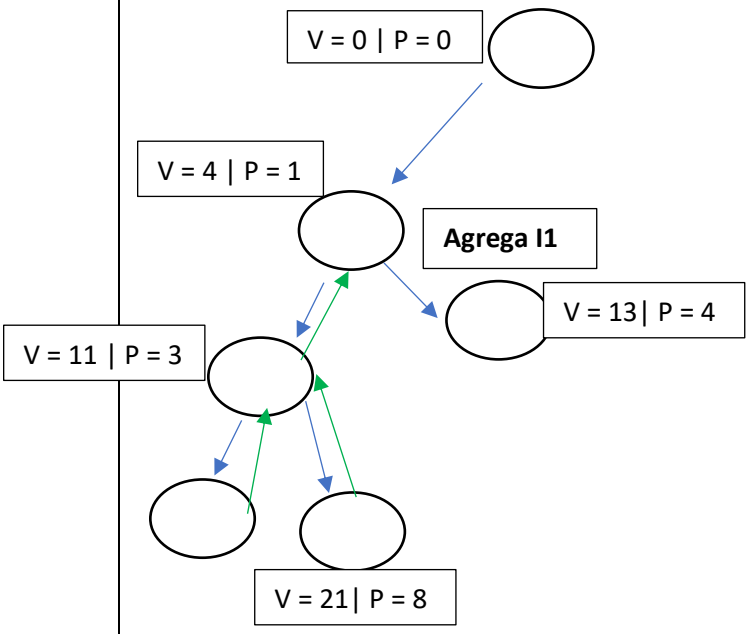


Como no es posible seguir combinando, nos devolvemos hasta un punto donde sea posible hacerlo y al igual que antes, se calcula el posible beneficio de agregar I1:

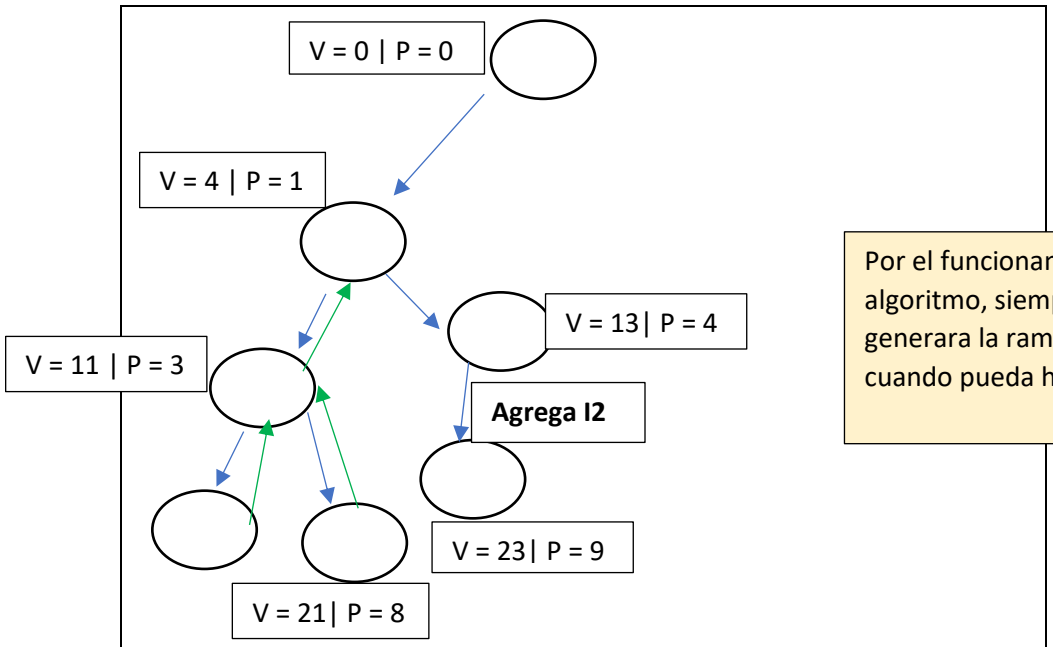
Posible beneficio = 4 + 8*3= 28

Como 28 > a 21 (beneficioActual) se efectuará la combinación

Iteración 8

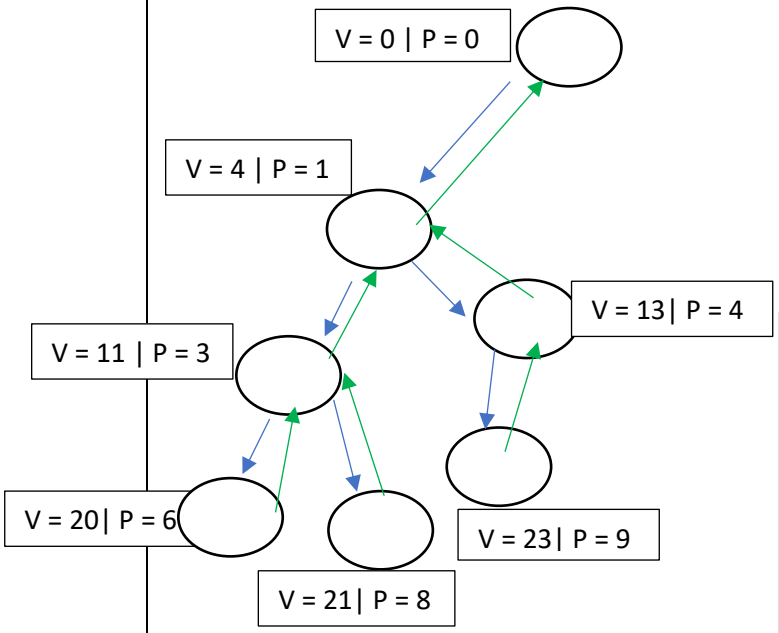


Iteración 9



Por el funcionamiento del algoritmo, siempre generara la rama izquierda cuando pueda hacerla

Iteración 11

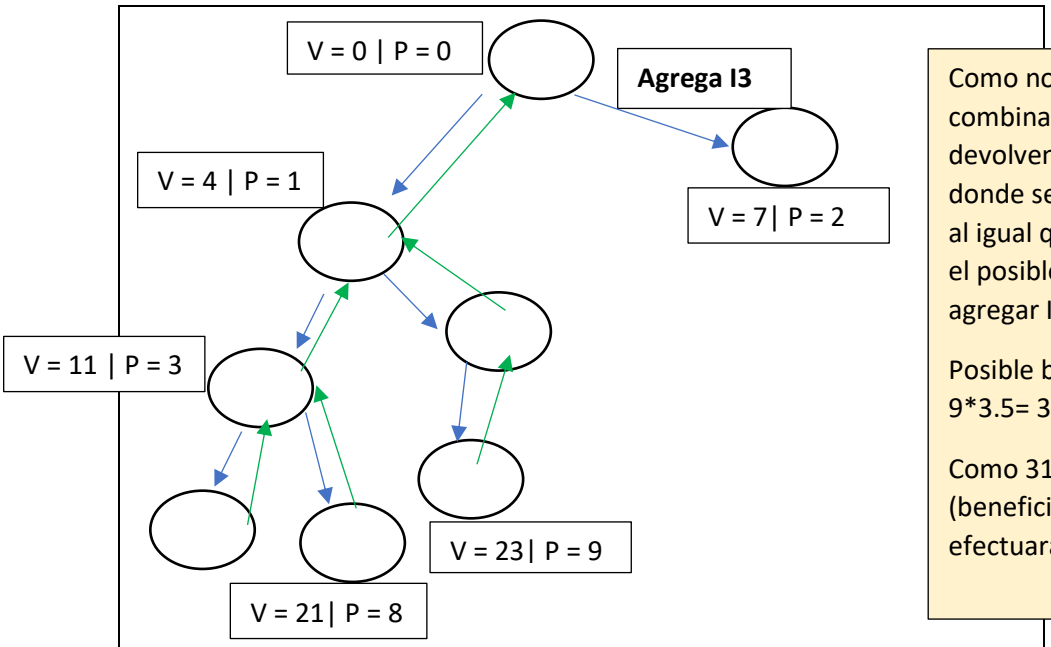


No es posible añadir más elementos, además beneficioNodo > beneficioActual, entonces se actualiza el beneficioActual

~~**beneficioActual = 21**~~ **beneficioActual = 23**

Al igual que antes, al no poder añadir más elementos, se devuelve a un punto en el que pueda volver a hacerlo

Iteración 12

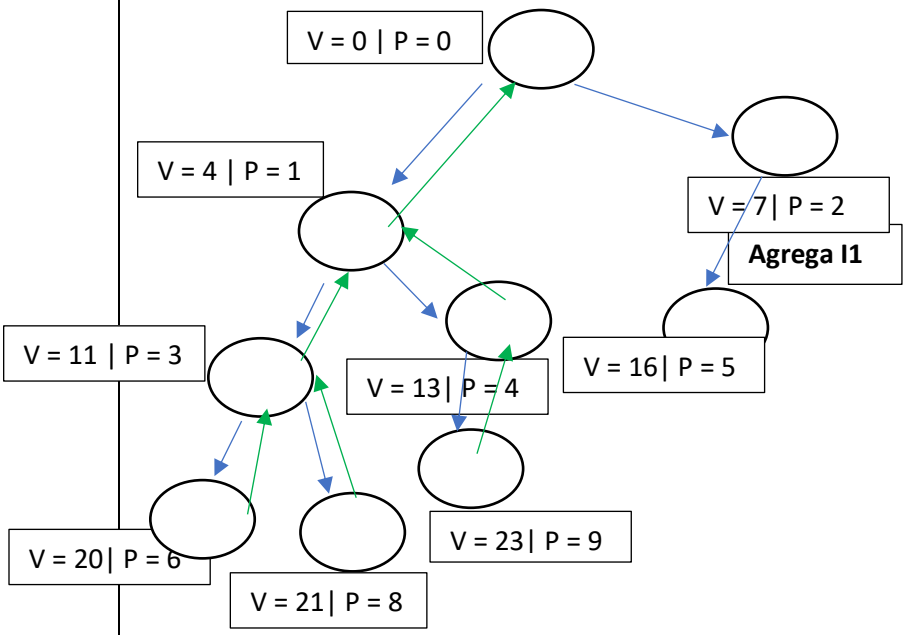


Como no es posible seguir combinando, nos devolvemos hasta un punto donde sea posible hacerlo y al igual que antes, se calcula el posible beneficio de agregar I3 (Rama derecha):

Posible beneficio = 0+ 9*3.5= 31.5

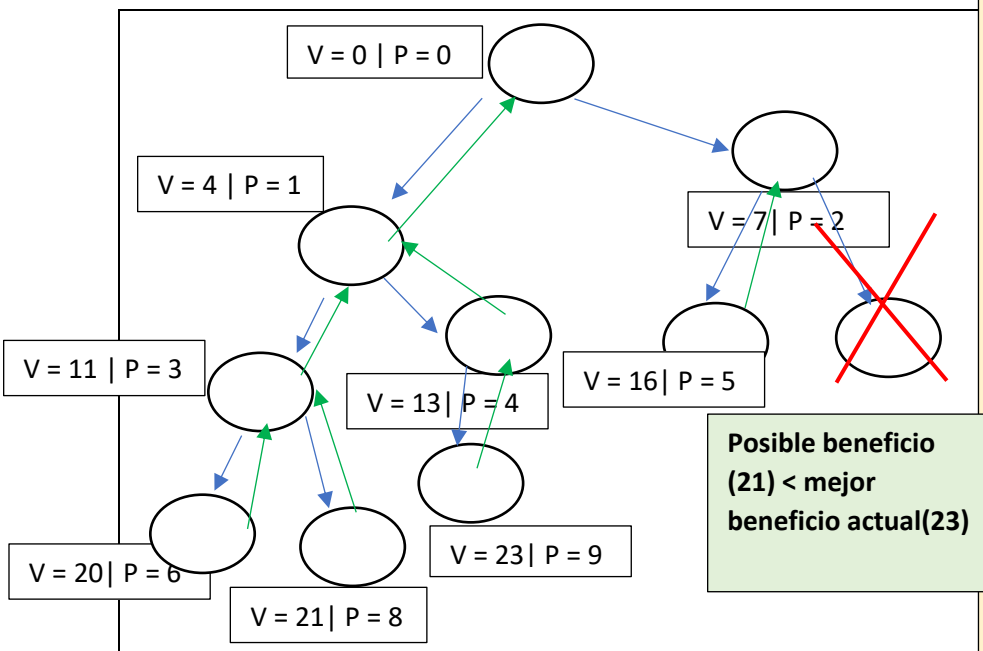
Como 31.5 > a 23 (beneficioActual) se efectuará la combinación

Iteración 13



Por el funcionamiento del algoritmo, siempre generara la rama izquierda cuando pueda hacerla

Iteración 14



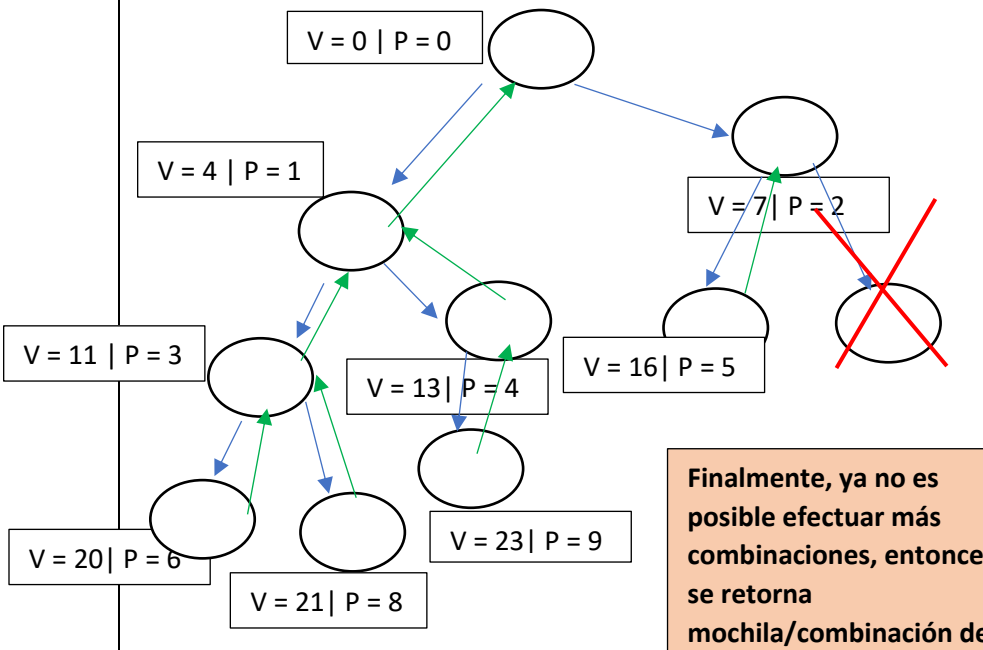
Posible beneficio (21) < mejor beneficio actual(23)

Como no es posible seguir combinando, nos devolvemos hasta un punto donde sea posible hacerlo y al igual que antes, se calcula el posible beneficio de agregar I2 (Rama derecha):

Posible beneficio = 7+ 7*2= 21

Como 21 < 23 (beneficioActual) No se efectuará la combinación y se eliminaran todas las posibles combinaciones que pudieran derivar de seguir esa rama, ya que nunca podrán ser mayor al mejor beneficio actual

Iteración 15



Finalmente, ya no es posible efectuar más combinaciones, entonces se retorna mochila/combinación de beneficio 23 y peso 9

Finalmente, se escribe el archivo de salida.txt con la mejor combinación obtenida

Fin de programa