Proyecto Final de Curso FADA

Profesor Mauricio López Benítez

Presentado Por:

Luis Camilo Urrea Monsalve – 1968207

Gustavo Adolfo Gutiérrez Agudelo - 1968208

SEDE TULUÁ

Facultad de Ingeniería

Curso de Fundamentos de Análisis y Diseño de Algoritmos

2022

Comentarios Sobre la Solución Dinámica Del Problema

Ya que el problema a solucionar se trata de buscar el menor costo de producción entre los meses dados, optamos por representar el problema mediante un grafo multietapas.

Al ser nuestra solución un grafo cuyos nodos varían para representar la cantidad de producto presente en cada mes, representamos el grafo y sus nodos utilizamos listas, en este caso usamos la clase ArrayList en la cual, para buscar un dato la complejidad es (1), esto optimiza el algoritmo ya que la parte más importante de este es comparar una lista con otra.

El array de etapas está formado por cada etapa, y cada etapa es un ArrayList con un objeto tipo nodo, el cual tiene 5 variable que son:

Unidades Actuales: uActuales

Próximo Nodo: proxNodo

Capacidad de inventario: capInv

Unidades para producir en el mes: producir

Costo Acumulado: cAcumulado

En el algoritmo primero creamos un ArrayList “Etapas” el cual se llena con n+1 ArrayList que a su vez estos contienen m+1 Objetos tipo Nodo.

Los Nodos se inicializan con todos sus componentes en 0, excepto uActuales y capInv.

Después de representar el grafo mediante listas, buscamos el camino mas optimo recorriendo las etapas desde atrás hacia adelante; allí comparamos todos los camino desde una etapa a otra,se compara cada nodo con los nodos siguiente y vamos almacenando en cada nodo el camino más óptimo, y se guardan el próximo nodo, las unidades a producir y el costo acumulado en el nodo presente.

Al final de todo, recorrer de nuevo la lista de etapas, empezando por el nodo i0 el cual tiene el costo total en su variable costo acumulado, además tiene la cantidad de unidades a producir para el primer mes, y por último pasamos a la siguiente etapa, al nodo que nos indica proxNodo de la etapa anterior y así, hasta llegar al último nodo, donde ya tendremos las unidades a producir y el costo total de producción.

Pseudocódigo Dinámica:

```
Metodo(n, meses, ci, cu, cv, u, m)
        Lista Etapas
        Lista primeraEtapa
        primeraEtapa.add(Nodo(0, 0, 0, 0, 0))
        Etapas.add(primeraEtapa)
        for ( i = 0; i < n - 1; i++)
            Etapas.add(CrearNodo(m, meses[i]))
        Lista ultimaEtapa
        ultimaEtapa.add(Nodo(meses[n - 1], 0, 0, 0, 0))
        Etapas.add(ultimaEtapa)
        for (i= length(Etapas)-1; i > 0; i--)
            Lista Lista1 = Etapas[i-1]
            Lista Lista2 = Etapas[i]
            for (k=0; k < length(Lista1); k++)
                min = INFINITY
                proximo = 0
                for (z=0; z < length(Lista2); z++)
                    costo = Ccosto(Lista2[z].uActuales, Lista1[k].CapInv(), …)
                    uProduccion = Lista2[z].getuActuales- Lista1[k].CapInv
                    if (uProduccion >= 0 & uProduccion <= u & min > costo)
                        min = min(costo, min)
                        proximo = z
    Resultado(Etapas);
```

Comentarios Sobre la Solución Voraz Del Problema

Para esta solución nos basamos en que iniciar la producción tiene un costo significativo, y es más costoso que almacenar, optamos por evitar iniciar producción, para ello, se produce para mantener el inventario lleno etapa tras etapa o hasta el máximo de producción, así, se logrará que posiblemente en uno de los meses no haya que iniciar producción.

Pseudocódigo Voraz:

```
Metodo (int n, list(meses), ci, cu, cv, u, m)
    listaResult
    for (i=0; i < length(meses) ; i++)
        if (meses[i] > inv)
            producir = min(u, meses[i]-inv)
            costo += ci+(producir + cu)+(cv * inv)
            listaResult[i] = producir
            inv = producir + inv - meses[i]
        else
            listaResult[i] = 0
            costo += (inv * cv)
            inv -= meses[i]
```

Complejidad Computacional

Dinámica: $T(n) = O(n*m^2)$

El costo computacional de la solución dinámica es constante o lineal en casi todas las líneas de pseudocódigo, así que lo que realmente vamos a estudiar y nos dará la cota superior aproximada será la parte del triple for, que compara todos los nodos de una lista con los de la otra.

	Costos	Instrucciones
for Etapas i to 0)	C1	n
Lista Lista1 = Etapas[i-1]	C2	n-1
Lista Lista2 = Etapas[i]	C3	n-1
for Lista1 k to length	C4	(n-1)* m
min = INFINITY	C5	(n-1)*(m-1)
proximo = 0	C6	(n-1)*(m-1)
for lista2 z to length	C7	(n-1)(m-1)m
costo = Ccosto(Lista2[z].uActuales, Lista1[k].CapInv(), ...)	C8	(n-1)(m-1)(m-1)
uProduccion = Lista2[z].getuActuales- Lista1[k].CapInv	C9	(n-1)(m-1)(m-1)
if (uProduccion >= 0 & uProduccion <= u & min > costo)	C10	(n-1)(m-1)(m-1)
min = min(costo, min)	C11	(n-1)(m-1)(m-1)
proximo =	C12	(n-1)(m-1)(m-1)

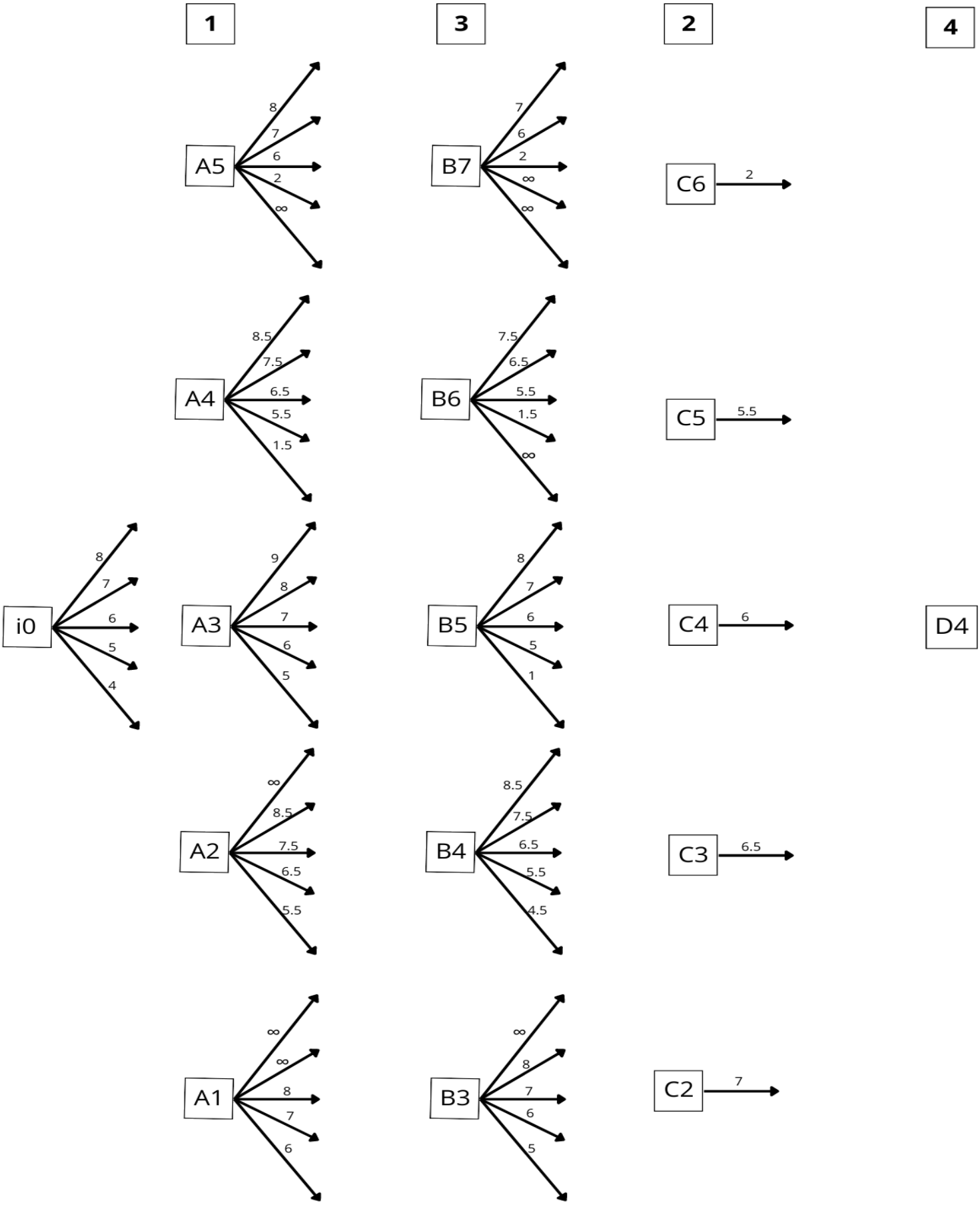
Voraz:

$T(n) = O(n)$

	Costos	Instrucciones
Metodo (int n, list(meses), ci, cu, cv, u, m)		
listaResult	C1	1
for (i=0; i < length(meses) ; i++)	C2	n+1
if (meses[i] > inv)	C3	n
producir = min(u, meses[i]-inv)	C4	n
costo += ci+(producir + cu)+(cv * inv)	C5	n
listaResult[i] = producir	C6	n
inv = producir + inv - meses[i]	C7	n
else		
listaResult[i] = 0	C9	n
costo += (inv * cv)	C10	n
inv -= meses[i]	C11	n

Ejemplo

Entrada : Método(4, {1,3,2,4}, 3.0, 1.0, 0.5, 5, 4)



S	f4(s)	x4
C6	2	D4
C5	5.5	D4
C4	6	D4
C3	6.5	D4
C2	7	D4

S/X	C6	C5	C4	C3	C2	f3(s)	x3
B7	9	11.5	11.5	8	∞	∞	C4
B6	9.5	12	12	11.5	8	∞	C3
B5	10	12.5	12.5	12	11.5	8	C2
B4	10.5	13	13	12.5	12	11.5	C6
B3	∞	13.5	13.5	13	12.5	12	C2

S/X	B7	B6	B5	B4	B3	f2(s)	x2
A5	16	15	14	12.5	∞	12.5	B4
A4	16.5	15.5	14.5	16	13.5	13.5	B3
A3	17	16	15	16.5	17	15	B5
A2	∞	16.5	15.5	17	17.5	15.5	B5
A1	∞	∞	16	17.5	18	16	B5

S/X	A5	A4	A3	A2	A1	f1(s)	x1
i0	20.5	18.5	17.5	16.5	15.5	15.5	A1

Respuesta dinámica: Camino = i0 - A1 - B5 - C2 - D4

Producción= {1 - 5 - 0 - 4}

Costo = 20

Respuesta Voraz:

Esta solución no tiene representación en tablas, el criterio se basa en producir al máximo o no producir, la respuesta dada es:

R/ producción mensual = { 5,0,5,0 }

costo = 20.5

Análisis de los Resultados

La programación voraz por lo general nos lleva a soluciones bastante óptimas, cercanas a la solución más óptima y en algunas ocasiones hasta la solución más óptima. Pero en cambio la programación dinámica siempre nos proporciona la solución más óptima. Estas dos estrategias son muy diferentes hasta en el enfoque que se les da al momento de abordar el problema, ya que con la voraz tratamos de dar una respuesta muy optima y lo más rápido posible, en cambio con la dinámica se trata de analizar todas las posibilidades hasta llegar a la mejor solución posible. Por esta razón la solución voraz en términos de complejidad computacional siempre termina siendo menos costosa que la programación dinámica.

Con esto lo que podemos concluir es que la programación voraz es menos costosa computacionalmente que la dinámica, y nos entrega buenas soluciones, en cambio la programación dinámica siempre nos entrega la mejor solución posible.