Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y
Cs. de la Computación

Informe proyecto 3

Análisis de Algoritmos (503309-1)
Profesora Cecilia Hernández

Alumnos:

Nicolás Araya

Martina Cádiz

Ayudante:

Manuel Díaz

Fecha:

06/08/21

Descripción del informe

El presente informe tiene el objetivo de describir la implementación de algoritmos aproximados, y técnicas de programación dinámica tales como Top-Down o Bottom-Up en la resolución de los problemas planteados.
Para la implementación de los algoritmos se trabajó utilizando C++, utilizando distintas funciones y estructuras de datos para abordar los problemas planteados.

Descripción de las soluciones

A continuación, se describirán las soluciones en el orden de aparición del listado entregado.

1.
a) Se pretende obtener el conjunto de vértices independientes de mayor peso (CVIMP) de un grafo lineal G(V,E), con n = V vértices y m = E aristas.

Dado que un grafo lineal es aquel en el que cada vértice sigue un orden. Se utiliza la siguiente notación
v_i: vértice i-ésimo del grafo.

La solución al problema planteado viene dado por la siguiente expresión, la cual divide en subproblemas más pequeños:

$$CVIMP(i)\begin{cases} -1 & i \leq 0 \\ V[i] & i = 1,\ \ i = n-1,\ \ i = n \\ \max(V[i-1], V[i]) & i = 2 \\ \max\left(CVIMP(i+2) + V[i],\ \ CVIMP(i+3) + V[i]\right) & 2 < i < n-1 \end{cases}$$

Demostración:
Suponiendo que tenemos un conjunto de elementos de `1...n` vértices. Y tenemos una solución óptima `CVIMP(n)` la cual incluye el ultimo elemento n .

Se puede obtener la solución óptima `X=CVIMP(n)-{n}` para los elementos `1...n-2`

Esto solo es verdad si y solo si `CVIMP (n)` es una solución optima para el conjunto completo, ya que al eliminar el elemento `{n}` tampoco podemos considerar `{n-1}`, por lo que si eliminamos `{n}` de la solución optima, el resultado tiene que ser si o si una solución optima de los elementos `1...n-2`.

b) **Pseudocódigo función generador**

```
generador(n, a, b):
	Graph <- vector de int donde se almacena el grafo lineal.
	maximo = Max(a, b)
	minimo = Min(a, b)

	for i = 0 to n-1 do:
		Graph[i] = rand()%(maximo-minimo) + minimo //un numero entre a y b

	return Graph
```

c) **Pseudocódigo función Top Down inicial.**

```
TopDownInicial(Graph):
	if V[Graph] == 0 then:
		return -1; //Si el grafo esta vacío.
	if V[Graph] == 1 then:
		return Graph[1] //Si el grafo solo tiene un elemento
						solo retorna ese elemento.
	if V[Graph] == 2 then:
		return Max(Graph[0], Graph[1]) //Si hay dos elementos
						en el grafo se retorna el mayor.

	Memoizacion <- Vector de int utlizado para la recursividad.

	for i = 0 in V[Graph] do: //se recorre desde i=0 hasta la cantidad de vert
		Memoizacion[i] = -1


	a = TopDownRecursivo(Graph, 0, memoizacion)
	b = TopDownRecursivo(Graph, 1, memoizacion)

	Solucion = Max(a, b)
	return Solucion
```

Pseudocódigo función Top Down recursiva

```
TopDownRecursivo(Graph, pos, Memoizacion):

	if Memoizacion[pos] != -1 then:
		return Memoizacion[pos]

	if pos == V[Graph] then:
		Memoizacion[pos] = Graph[pos]
		return Graph[ V[Graph] ]
	if pos == V[Graph] - 1 then:
		Memoizacion[pos] = Graph[pos]
		return Graph[ V[Graph] - 1]

	a = 0
	b = 0
	if pos+2 < V[Graph] then:
		a = TopDownRecursivo(Graph, pos + 2, Memoizacion)
	if pos+3 < V[Graph] then:
		b = TopDownRecursivo(Graph, pos + 3, Memoizacion)

	Memoizacion[pos] = Max(a,b) + Graph[pos]

	return Memoizacion[pos]
```

```
BottomUp(Graph):
	sub <- vector en el que se almacenaran los pesos máximos de los sub grafos.
	if V[Graph] == 0 then:
		return -1
	if V[Graph] == 1 then:
		sub[0] = Graph[0]
	if V[Graph] == 2 then:
		sub[1] = Graph[1]

	for i = 2 in V[Graph] do: //Se recorre cada vértice restante
		if Graph[i] + sub[i-2] > sub[i-1] then:
			sub[i] = Graph[i] + sub[i-2]
		else:
			sub[i] = sub[i-1]

	return sub[V[graph]] //retornar el último elemento
```

d) Para ambos ejemplos se considerará el mismo Grafo, asumiendo que fue generado por el generador.

Grafo = {6, 1, 2, 11, 5, 4}
V[G] = 6

Enfoque Top-Down con Memoización

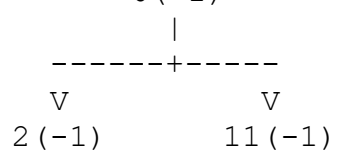
La idea del algoritmo es explorar todas las posibilidades partiendo o bien desde el primer elemento (elemento cero) o desde el segundo elemento. Ya que para una solución valida solo puede pertenecer uno de ellos. Y utilizar el vector Memoización para almacenar el máximo peso que se puede obtener desde ese punto hacia la derecha.

Cuando se llama a TopDownInicial con Grafo, en el vector Memoización se almacenan 7 valores de -1. Este valor sirve para saber si ya se calculó el mayor peso en ese punto o no.

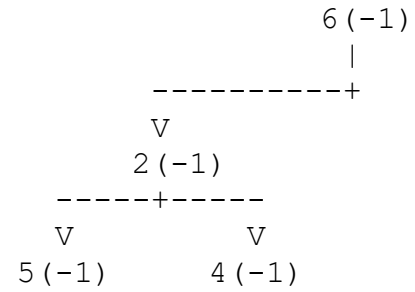
Cada elemento i del vector Memoización representa el peso máximo entre la posición i y el final del vector, es decir, considerando el elemento i-ésimo del grafo, hasta el final.

Ahora se continua, llamando a la función TopDownRecursiva, con el primer elemento.

Este a su vez, llama a sus dos posibilidades



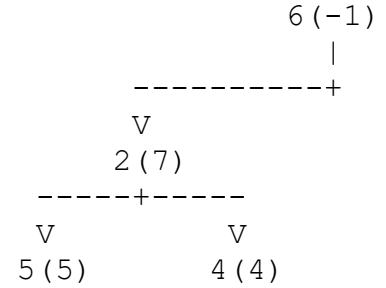
Y luego 2 llama a sus sucesores posibles.



Como 5 y 4 corresponden al final del grafo, ese es el peso maximo que se puede obtener en esa posicion, por lo tanto escriben en el vector Memoizacion, su valor

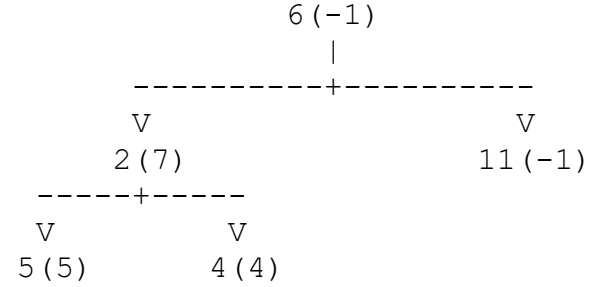
```
Memoizacion = {-1, -1, -1, -1, 5, 4}
```

Luego de eso se retorna al Nodo 2, en cual se almacena el peso máximo obtenido en esa rama, el cual será el nodo mas el mayor obtenido en sus ramas hijas

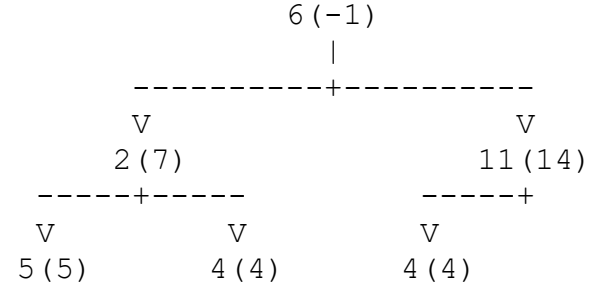



```
Memoizacion = {-1, -1, 7, -1, 5, 4}
```

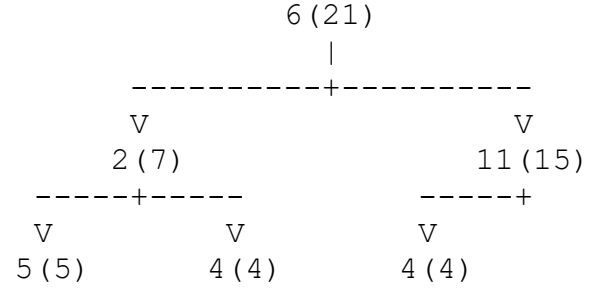
Ahora siguiendo con el otro la rama con el Nodo 11



Este nodo solo cuenta con una sola posibilidad, baja con el Nodo 4, pero como este ya se ha calculado previamente su peso maximo se retorna y actualiza el peso maximo de 11



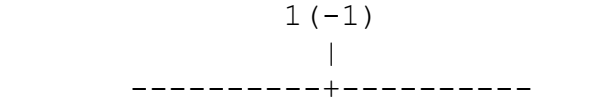
Finalmente volviendo al Nodo 6, solo restaria elegir la rama de mayo peso y sumarle 6, correspondiente al valor del Nodo.



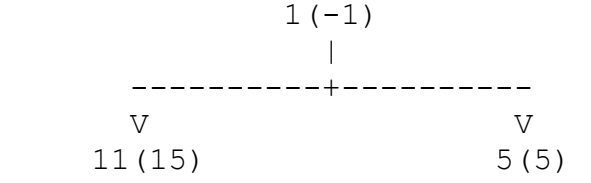
Con esto, el vector Memoizacion quedaria se la siguiente manera:

```
Memoizacion = {21, -1, 7, 15, 5, 4}
```

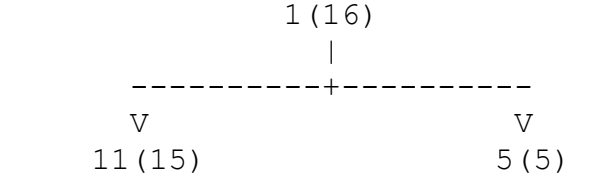
Ahora si se connsidera la llamada a TopDownRecursivo desde el Nodo con valor 1 quedaria:



El cual también tiene dos posibilidades, continuar con 11 o 5, pero ya se tienen guardados previamente los valores de los pesos máximos en el vector Memoizacion,



Asi que no es necesario seguir bajando explorando el arbol, sino que se puede subir inmediatamente y actualizar el peso maximo posible incluyendo al Nodo 1.




```
Memoizacion = {21, 16, 7, 15, 5, 4}
```

Finalmente volviendo a la funcion TopDownInicial se debe conservar el resultado de mayor peso. En este ejemplo bajando por el numero 6, se obtiene el peso maximo y la solucion al problema, es decir 21.

Enfoque Bottom-Up

De manera similar a como se hace en Top-Down, en este caso se recorre el grafo y a medida que se va avanzando se va almacenando en un vector, el mayor peso que se puede ir obteniendo hasta ese punto.

El algoritmo inicia, y se tiene un vector sub, en el que cada posición i representa la mayor cantidad de peso que se puede obtener entre los elementos 0, 1, …, i. Por lo que en la posición 0 del vector, siempre se tiene el elemento 0 del grafo, y en la posición 1 se tiene el máximo entre la posición 0 y la posición 1 del grafo.

Por lo tanto, antes de comenzar con el ciclo iterativo, para este ejemplo nuestro vector sub estaría de la siguiente manera:

```
Sub = {6, 6}
```

En cada iteración se evalúa el candidato de peso máximo que podría haber considerado los elementos por lo que se han recorrido, para esto se consideran dos opciones

1. Si la suma entre el elemento actual y el peso máximo en la iteración i-2 es mayor al peso máximo en la iteración i-1, entonces se almacena en la posición i del vector sub la suma obtenida.
2. En caso contrario, es decir, si la suma es menor al peso máximo obtenido en la iteración i-1. Se conserva este valor en la posición i del vector sub.

El ciclo iterativo parte desde i = 2, ya que ya se consideraron los casos anteriores.

Para i = 2:

```
Grafo = {6, 1, 2, 11, 5, 4}
Sub = {6, 6}
```

La suma entre el elemento i y el peso máximo en i – 2 es: `6 + 2`
Lo cual es mayor al peso máximo en la iteración i – 1: `6 + 2 > 6`
Por lo tanto, se guarda en el vector sub ese peso máximo

Para i = 3:

```
Grafo = {6, 1, 2, 11, 5, 4}
Sub = {6, 6, 8}
```

La suma entre el elemento i y el peso máximo en i – 2 es: `11 + 6`
Lo cual es mayor al peso máximo en la iteración i – 1: `17 > 8`
Por lo tanto, se guarda en el vector sub ese peso máximo

Para i = 4:

```
Grafo = {6, 1, 2, 11, 5, 4}
Sub = {6, 6, 8, 17}
```

La suma entre el elemento i y el peso máximo en i – 2 es: `5 + 8`
Lo cual es menor al peso máximo en la iteración i – 1: `13 < 17`
Por lo tanto, se guarda en el vector sub el peso máximo de la iteración i – 1

Para i = 5:

```
Grafo = {6, 1, 2, 11, 5, 4}
Sub = {6, 6, 8, 17, 17}
```

La suma entre el elemento i y el peso máximo en i – 2 es: `4 + 17`

Lo cual es mayor al peso máximo en la iteración i – 1: `21 > 17`
Por lo tanto, se guarda en el vector sub ese peso máximo

Finalmente terminamos el ciclo iterativo, y el vector sub queda de la siguiente manera:

`Sub = {6, 6, 8, 17, 17, 21}`

Como se puede ver, siempre en la ultima posición esta el mayor peso que se puede obtener considerando todos los elementos, por lo cual el resultado es 21, concordando con la solución de Top-Down

2.

a) El problema Set cover es una generalización de Vertex cover. Para resolver el problema especial de Set cover propuesto, es necesario hacer unas asunciones para la transformación:

1. Se considerará que Set cover está dado por (U,S,K), donde U corresponde al universo, S a los subconjuntos y K a la respuesta.
2. Además se tomará G = (V,E) como un grafo no dirigido, donde V corresponde a todos los vértices y E a las aristas.

Para poder asociar (U,S,K) a un grafo, se considerará que los elementos de U se corresponderán a las aristas (E) y los subsets (S) a los vértices (V), también que cada arista es incidente a un vértice de los subsets.
Teniendo en cuenta los factores anteriores, se considerará el problema de Set cover propuesto en la Figura 1.

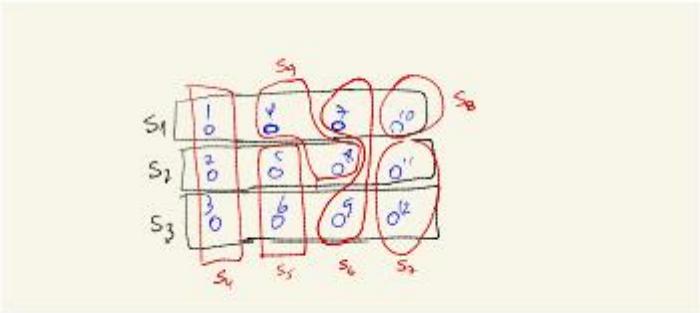


Figura 1. Set cover especial propuesto.

Gracias a la transformación es posible aplicar el algoritmo Vertex cover al problema propuesto, ya que cumple con las asunciones mencionadas. Por lo que se tiene lo siguiente:

1. En el caso del Set cover:
   U = {1,2,3,4,5,6,7,8,9,10,11,12}

   S1 = {1,4,7,10} S2= {2,5,8,11} S3 ={3,6,9,12} S4 = {1,2,3}
   S5 = {5,6} S6={7,9} S7={11,12} S8= {10} S9 = {4,8}
2. Aplicando la transformación propuesta:
   E = U y v ∈ V: $S_v = \{e \in E: e\ incidente\ a\ v\}$

   V = {S1,S2,S3,S4,S5,S6, S7, S8}
   E = {1,2,3,4,5,6,7,8,9,10,11,12}

3. Visualización:

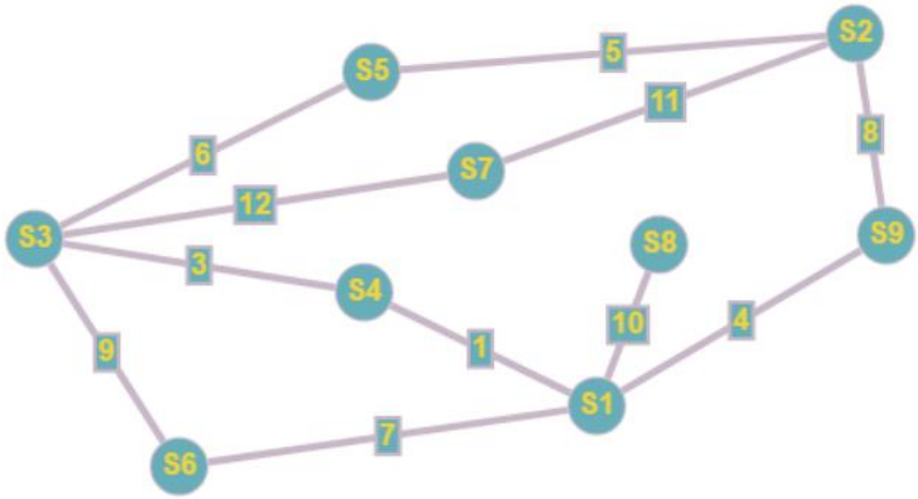


Figura 2. Set cover post transformación.

Aplicando Vertex cover, podemos ver en la Figura 2 que S1, S2, S3 entregan la cardinalidad mínima que cubra todo el gráfo. Dado el ejemplo de Set cover, se pudo apreciar que el mínimo corresponde a K = 3. Por lo que es posible decir que el cálculo mediante Vertex cover si fue posible y entregó los mismos resultados.

b) Si el problema es posible resolverlo con vertex cover, por ende, debe ser calculable con 2-apróximado vertex cover. En el ejemplo anterior se tiene que cada elemento del universo está contenido en 2 conjuntos, por lo que, si fue posible llevarlo directamente a vertex cover, sin embargo, el problema general propuesto menciona que cada elemento del universo puede estar contenido en a lo más 2 conjuntos. Considerando lo anterior, puede caer el caso que un elemento pueda estar contenido en un único subset. Si es que llegase a ocurrir lo mencionado, el valor mínimo que entregue k con el algoritmo de set cover será distinto al de vertex cover, a raíz de que no se considerará la arista sin incidencia, por lo tanto, en ese caso claramente no podemos llevarlo a 2-apróximado, entonces para que esto no ocurra y sea posible llevarlo a 2-apróximado, se puede asumir que el vértice pueda tener autociclos, por ende la arista incide sobre ese mismo vértice.

Ejecución del código

Para ejecutar el código, se recomienda lo siguiente:
Paso 1. Abrir la carpeta Proyecto3
Paso 2. Para compilarlo, escribir en consola lo siguiente: `g++ main.cpp`
Paso 3. Para ejecutarlo, escribir en consola siguiendo la siguiente estructura: `./a.out n a b`

Siendo **n** la cantidad de vértices que se quieren añadir al grafo, y **a** y **b** los límites del intervalo en el que se encontraran los valores de cada vértice.

Paso 4. Ver resultados en consola.
En ella se mostrará el grafo generado, y el resultado obtenido por ambas soluciones.

Ejemplo:

Paso 1.

```
nicolas@DESKTOP-TQ97LDH:/mnt/c/Users/nicolas/Desktop/U/analisis/Proyecto3$ g++ main.cpp
```

Paso 2.

```
nicolas@DESKTOP-TQ97LDH:/mnt/c/Users/nicolas/Desktop/U/analisis/Proyecto3$ ./a.out 10 25 40
```

Grafo de 10 nodos entre 25 y 40.

Paso 3.

```
nicolas@DESKTOP-TQ97LDH:/mnt/c/Users/nicolas/Desktop/U/analisis/Proyecto3$ ./a.out 10 25 40
29 30 32 27 25 33 26 26 28 25
Solucion con TopDown 145
Solucion con BottomUp 145
```

Resultados obtenidos.