Tecnológico de Costa Rica

Segundo Proyecto Programado

## Algoritmos y estructuras de Datos 2

#### Profesora: Ana Lorena Valerio

Realizadores del proyecto:

* Diego Alfaro
* Alonso Vega

# Descripción del problema

Para el proyecto presente se pidió que se cumplieran varios objetivos y que se pudieran realizar diversas operaciones y consultas. Básicamente, el proyecto consiste en una aplicación de escritorio que se encarga de manejar sistemas de grafos, utilizando estructuras conocidas como vértices y arcos, además de objetos, para la solución del problema del viajero y de la mochila; lectura de archivos de texto plano, algoritmos de varias categorías y mediciones en ellos. El código fuente de la aplicación debe estar escrito en el lenguaje de programación orientado a objetos conocido como C++, y se debe aprovechar esta característica mencionada.

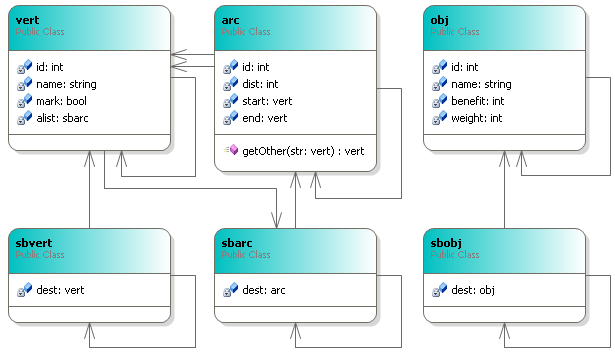
La idea general es que la aplicación pueda abrir archivos de texto plano con un formato definido, del cual se leerán varias listas: vértices, arcos y objetos. Usando esta información, se construirá un grafo y una lista de objetos. Una vez que están listos, el usuario puede seleccionar una serie de algoritmos para buscar la solución de cada problema.

En el caso del viajero, se debe recorrer el grafo dado un punto inicial y se deben recorrer todos los vértices, llegando al origen nuevamente. Lo que se busca es encontrar la ruta con la que se obtiene la menor distancia. El problema de la mochila consiste en seleccionar de una lista de objetos con una cantidad definida de beneficio y de peso, los que puedan brindar la mayor cantidad de beneficio sin pasarse de un máximo establecido para la mochila.

# Solución del problema

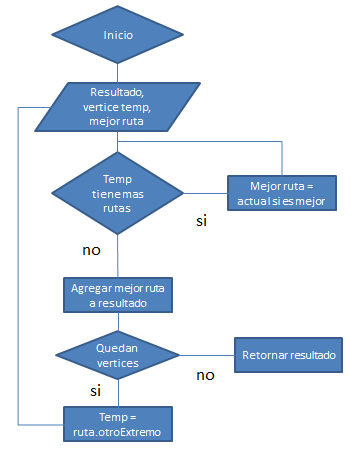
## Diagrama de clases

Estas son las clases básicas utilizadas. Consisten en 3 clases principales: vértice, arco y objeto; y otras 3 clases asociadas como sublistas, correspondientes a las principales.

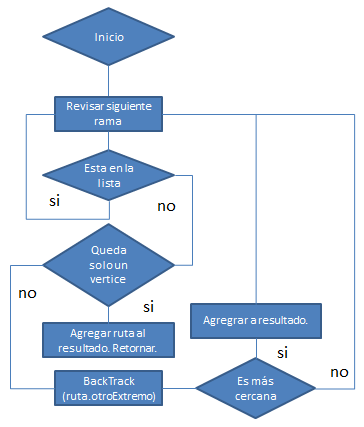


## Diagramas de flujo de algoritmos

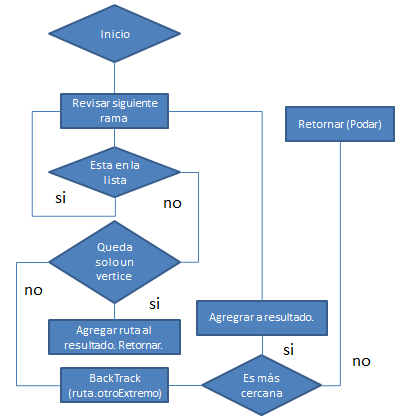
**Problema del viajero: Método voraz**



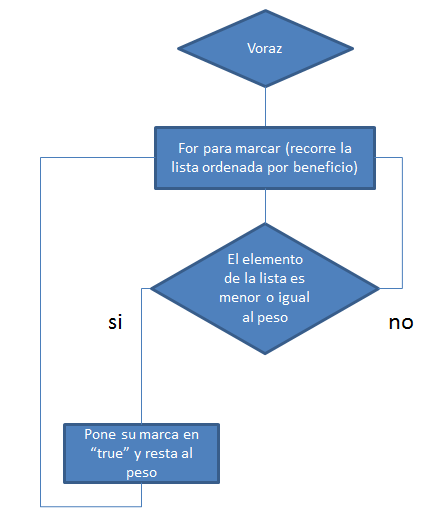
**Problema del viajero: Método de Back Tracking**



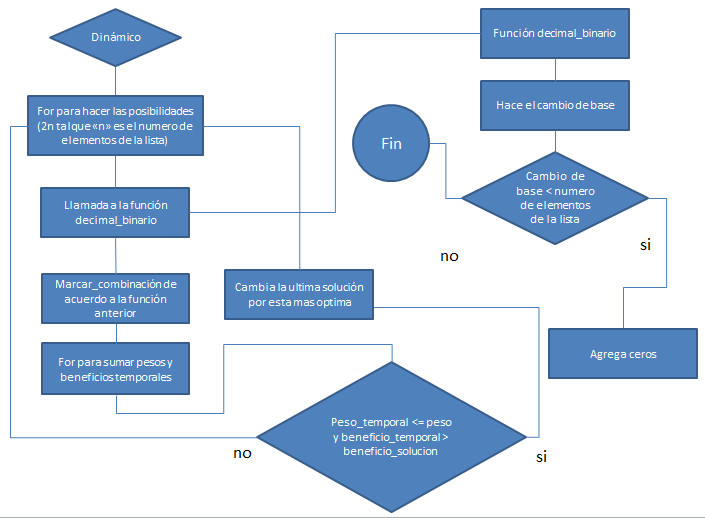
**Problema del viajero: Método de Ramificación y poda**

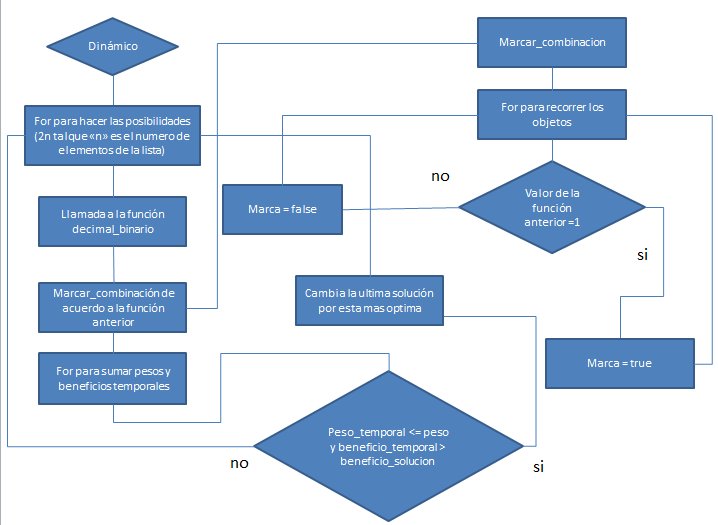


**Problema de la mochila: Voraz**



**Problema de la mochila: Programación dinámica**





## Realización

Fecha de inicio del proyecto: 17 de mayo, 2011.

Fecha de última modificación: 8 de junio, 2011.

En esta sección procederemos a explicar las estrategias, técnicas y métodos que utilizamos para cumplir con cada objetivo planteado en el problema. Subdividiremos esta sección en cada parte para que sea más modular y fácil de comprender.

### Clases y estructuras de datos

Las estructuras de datos principales utilizadas fueron las siguientes:

**Problema del viajero**

Vértice, Arco, Sub-vértice (para sublistas) y Sub-arco (para sublistas).

En vértice se utilizaron la propiedad básica de nombre, además de un ID y una marca para las funciones de los algoritmos. Además se agregaron los punteros propios de listas dinámicas utilizados para generar los enlaces para el grafo.

En el arco se agregaron los valores de distancia, origen y destino (definidos como punteros a vértices), un ID, una marca y los punteros de siguiente para tener una lista por amplitud.

**Problema de la mochila**

Objeto, Sub-objeto (para sublistas).

En este caso, el objeto debía implementar las propiedades de nombre, peso y beneficio. Adicional a estos valores, se agregaron un ID, una marca y los punteros para generar listas dinámicas.

### El archivo de grafo y su lectura

Para almacenar un grafo en un archivo plano se utilizó un formato sencillo que consiste en 3 listas, una para cada clase principal (vértice, arco y objeto). Cada una de las 3 listas debe finalizar con un carácter de “:” (dos puntos). Dentro de las listas, cada elemento está terminado por el símbolo “;” (punto y coma). Al abrir un archivo, el texto dentro de este es procesado, y dependiendo de la lista que se está leyendo actualmente, cada elemento es tratado ya sea como vértice, como arco o como objeto. Según sea el tipo, se separan las propiedades dentro de este con una “,” (coma).

Un ejemplo de cómo se vería un archivo para un grafo pequeño (y una lista de objetos), es el siguiente:

0,Javillos;1,Los Ángeles;2,Fortuna;3,Chachagua;4,La Tigra;:10,3,2;15,1,2;6,0,4;14,0,1;14,3,4;:Botella de Agua,1,50;Bolso,2,60;Cuchilla,1,45;Foco,3,30;Cuerda,2,75;:

**Vértices**

**Arcos**

**Objetos**

### Problema del viajero

#### Método voraz

Usando una variable temporal que almacene el vértice actual recorrido, y un ciclo que revise sus rutas buscando la menor, este método realiza un proceso pequeño y rápido en el cual solo se prueba una ruta total posible. Esto causa que el tiempo de ejecución del mismo y el consumo de memoria sean muy pequeños. El método se detiene cuando logra determinar que todos los vértices están marcados, o cuando llega a un vértice en el cual no puede determinar una ruta que lo conecte con otro vértice que no esté marcado como visitado.

#### Método de Back Tracking

El proceso utilizado es un poco extenso, pero consiste en los siguientes pasos: A partir de un vértice determinado, buscar las rutas a las que está conectado. Por cada una de esas rutas, se busca que no esté marcado entre la lista de vértices visitados (una lista temporal para cada ejecución recursiva del método). Si se comprueba que el vértice sí está marcado, entonces se salta a la siguiente rama. Cuando el vértice no está visitado, se determina si seguir esa ruta devuelve una distancia total menor que la temporal almacenada (usando la llamada recursiva). Si efectivamente es menor, se guarda como “menor temporal”, ya que puede que se encuentre una menor en las siguientes rutas. Una vez que se revisan las ramas se devuelve la menor temporal, a menos que esta sea inválida (nula), lo cual significa que no se logró determinar una ruta posible con este método.

#### Método de Ramificación y poda

Este método es muy similar al anterior (Back Tracking), a excepción de que está mejor optimizado, y la idea es evitar realizar posibles soluciones que se sabe que no tendrán éxito. De esta manera se realiza un consumo menor de tiempo de ejecución y de memoria en general.

La técnica es simple: una vez que se encuentra una posibilidad, se comprueba su distancia total y se verifica si es menor a una previamente almacenada. En el caso de que no sea menor, entonces la posibilidad actual no tiene oportunidad de superar a la otra (en ese caso, la que haya obtenido la distancia menor). Es innecesario seguir procesando las posibilidades de la descartada, por lo que se evitan más procesos.

### Problema de la mochila

#### Método voraz

Al leer en archivo e insertar en el la lista de objetos, estos se insertan de forma ordenada por beneficio. Al realizarse de esta manera el método para programación voraz es muy sencillo, se ve revisa si el peso cabe en la mochila, si se puede agregar se marca una variable y se resta el peso y luego se sigue con todas las variables hasta que termine de recorrer la lista. Y después se recorre la lista imprimiendo solo los que tienen esta variable marcada con un “true”.

#### Método de Programación dinámica

Para realizar este algoritmo se optó por un razonamiento lógico muy efectivo, se explica de la siguiente manera: usted tiene que realizar las combinaciones posibles para determinar cuál es la correcta. Vemos esto en un proceso de conteo binario en el cual mientras uno cuenta en forma binaria va generando combinaciones para ser probadas. Entonces ¿en qué consiste el método? Es fácil; se cuenta en decimal de uno hasta “2n” (donde “n” representa la cantidad de objetos). Estas combinaciones se prueban en la lista de objetos, si el peso y el beneficio temporal, es decir, el de la solución que se está analizando, cumplen con la condición: de que el peso temporal de ser menor o igual al peso que se da para cargar en la mochila, y además son mayores que el beneficio temporal, se sustituye la solución. Así se hace hasta llegar a dos 2n que es lo máximo de combinaciones. Al terminar el proceso usted tiene el beneficio máximo, y una combinación binaria la cual se usa para marcar en la lista de objetos. Y igualmente al voraz, solo se recorre y se imprime.

# Análisis de resultados

## Resultados finales

|  |  |  |  |
| --- | --- | --- | --- |
| Tarea a realizar | Realizado con éxito | No realizado | Defectuosas |
| Abrir y leer archivos de texto desde cualquier unidad | X |  |  |
| Imprimir la Ruta Corta encontrada para el problema del viajero | X |  |  |
| Imprima todas las variables de medición para cada una de las estrategias de diseño | X |  |  |
| Imprimir el MÁXIMO beneficio en problema de la mochila para cada una de las estrategias e indicar cuáles fueron los OBJETOS seleccionados | X |  |  |
| El programa debe crear un grafo de transporte (arcos y vértices) obteniendo los datos desde un archivo. | X |  |  |
| Realizar el algoritmo de obtener la solución del problema del viajero mediante la estrategia voraz e imprimir todas las variables de las mediciones empíricas. | X |  |  |
| Realizar el algoritmo de obtener la solución del problema del viajero mediante la estrategia back tracking e imprimir todas las variables de las mediciones empíricas. | X |  |  |
| Realizar el algoritmo de obtener la solución del problema del viajero mediante la estrategia ramificación y poda e imprimir todas las variables de las mediciones empíricas. | X |  |  |
| Realizar las soluciones para el problema de la mochila | X |  |  |

## Tablas de mediciones

### Análisis empírico de costes del problema del viajero en un grafo de transporte conexo implementado con varias estrategias de diseño de algoritmos:

### Voraz, Back tracking, Ramificación y Poda

*Tabla 1.1: Método de Viajero:* ***Estrategia voraz***

| Operaciones |
| --- |
| Cantidad de arcos | | |
| Primer tamaño | Segundo tamaño | Tercer tamaño |
| Asignaciones | 51 | 60 | 137 |
| Comparaciones | 45 | 65 | 135 |
| Cantidad de líneas de código ejecutadas | 96 | 125 | 272 |
| Cantidad de líneas de código. | 29 | | |
| Cantidad de memoria consumida | 64 bytes | 64 bytes | 124 bytes |

¿Según los resultados que has obtenido, cómo se comporta el algoritmo del viajero en diseño voraz en función de de los tamaños del grafo? Marcar con X la respuesta que coincida con tu caso.

| (a) | El tamaño no importa a nivel de costes |  |
| --- | --- | --- |
| (b) | El tamaño afecta sólo al número de comparaciones |  |
| (c) | El tamaño afecta sólo al número de asignaciones |  |
| (d) | El tamaño afecta a los dos tipos de operaciones | **X** |
| (e) | El tamaño afecta la cantidad de memoria consumida | **X** |
|  |

*Tabla 1.2: Factor de incremento del número de operaciones al crecer el tamaño (en cantidad de arcos)*

| Talla | Factor talla | Factor Asig | Factor Comp | Cantidad de memoria consumida | Cantidad de líneas ejecutadas |
| --- | --- | --- | --- | --- | --- |
| Del primer al segundo tamaño | 2 | 1 | 2 | 1 | 1 |
| Del segundo al tercer tamaño | 1.5 | 2 | 2 | 2 | 2 |

El comportamiento esperado del algoritmo será que, en general:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | O(1) | O(n) | Otro\_\_\_\_\_\_\_\_ |  |
| Las asignaciones crecen |  | **X** |  | Con respecto a la talla |
| Las comparaciones crecen |  | **X** |  | con respecto a la talla |
| Cantidad de memoria consumida |  | **X** |  | con respecto a la talla |
| Cantidad de líneas ejecutadas crece |  | **X** |  | con respecto a la talla |

*Tabla 2.1: Método del viajero:* ***Back tracking***

| Operaciones |
| --- |
| Cantidad de arcos | | |
| Primer tamaño | Segundo tamaño | Tercer tamaño |
| Asignaciones | 112 | 868 | 3470 |
| Comparaciones | 139 | 1255 | 6641 |
| Cantidad de líneas de código ejecutadas | 251 | 2123 | 10111 |
| Cantidad de líneas de código. | 34 | | |
| Cantidad de memoria consumida | 228 bytes | 1880 bytes | 5068 bytes |
| Cantidad de combinaciones posibles. | 2 | 24 | 12 |

¿Según los resultados que has obtenido, cómo se comporta el algoritmo del viajero con el diseño back tracking en función de de los tamaños del grafo? Marcar con X la respuesta que coincida con tu caso.

| (a) | El tamaño no importa a nivel de costes |  |
| --- | --- | --- |
| (b) | El tamaño afecta sólo al número de comparaciones |  |
| (c) | El tamaño afecta sólo al número de asignaciones |  |
| (d) | El tamaño afecta a los dos tipos de operaciones | **X** |
| (e) | El tamaño afecta la cantidad de memoria consumida | **X** |
|  |

.

*Tabla 2.2: Factor de incremento del número de operaciones al crecer el tamaño de los arcos.*

| Talla | Factor talla | Factor Asig | Factor Comp | Cantidad de memoria consumida | Cantidad de líneas ejecutadas |
| --- | --- | --- | --- | --- | --- |
| Del primer al segundo tamaño | 2 | 8 | 9 | 8 | 8 |
| Del segundo al tercer tamaño | 1.5 | 4 | 5 | 3 | 5 |

El comportamiento esperado del algoritmo será que, en general:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | O(1) | O(n) | Otro 5n |  |
| Las asignaciones crecen |  |  | **X** | Con respecto a la talla |
| Las comparaciones crecen |  |  | **X** | con respecto a la talla |
| Cantidad de memoria consumida |  |  | **X** | con respecto a la talla |
| Cantidad de líneas ejecutadas crece |  |  | **X** | con respecto a la talla |

*Tabla 3.1: Método del viajero:* ***Ramificación y Poda.***

| Operaciones |
| --- |
| Cantidad de arcos | | |
| Primer tamaño | Segundo tamaño | Tercer tamaño |
| Asignaciones | 87 | 654 | 2715 |
| Comparaciones | 116 | 981 | 4718 |
| Cantidad de líneas de código ejecutadas | 203 | 1635 | 7433 |
| Cantidad de líneas de código. | 41 | | |
| Cantidad de memoria consumida | 188 | 1528 | 4340 |
| Cantidad de ramas (combinaciones) | 13 | 134 | 439 |
| Cantidad de podas | 5 | 70 | 269 |

¿Según los resultados que has obtenido, cómo se comporta el algoritmo del viajero con ramificación y poda en función de de los tamaños del grafo? Marcar con X la respuesta que coincida con tu caso.

| (a) | El tamaño no importa a nivel de costes |  |
| --- | --- | --- |
| (b) | El tamaño afecta sólo al número de comparaciones |  |
| (c) | El tamaño afecta sólo al número de asignaciones |  |
| (d) | El tamaño afecta a los dos tipos de operaciones | **X** |
| (e) | El tamaño afecta la cantidad de memoria consumida | **X** |
|  |

*Tabla 3.2: Factor de incremento del número de operaciones al crecer el tamaño de la cantidad de arcos.*

| Talla | Factor talla | Factor Asig | Factor Comp | Cantidad de memoria consumida | Cantidad de líneas ejecutadas |
| --- | --- | --- | --- | --- | --- |
| Del primer al segundo tamaño | 2 | 8 | 8 | 8 | 8 |
| Del segundo al tercer tamaño | 1.5 | 4 | 5 | 3 | 5 |

El comportamiento esperado del algoritmo será que, en general:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | O(1) | O(n) | Otro 5n |  |
| Las asignaciones crecen |  |  | **X** | Con respecto a la talla |
| Las comparaciones crecen |  |  | **X** | con respecto a la talla |
| Cantidad de memoria consumida |  |  | **X** | con respecto a la talla |
| Cantidad de líneas ejecutadas crece |  |  | **X** | con respecto a la talla |

*Tabla 4.1: Método de Mochila:* ***Voraz***

| Operaciones |
| --- |
| Cantidad de objetos (vértices) | | |
| Primer tamaño | Segundo tamaño | Tercer tamaño |
| Asignaciones | 8 | 8 | 10 |
| Comparaciones | 11 | 11 | 21 |
| Cantidad de líneas de código ejecutadas | 19 | 19 | 31 |
| Cantidad de líneas de código. | 5 | | |
| Capacidad mochila | 5 | 20 | 20 |
| Cantidad de objetos dentro de la mochila | 3 | 3 | 4 |
| Máximo beneficio dentro de la mochila | 185 | 255 | 139 |
| Suma de los pesos dentro de la mochila. | 5 | 16 | 20 |
| Suma de todos los beneficios. | 260 | 355 | 612 |
| Cantidad de memoria consumida | 4 bytes | 4 bytes | 4 bytes |

¿Según los resultados que has obtenido, cómo se comporta el algoritmo de la mochila con el diseño voraz en función de la cantidad objetos? Marcar con X la respuesta que coincida con tu caso.

| (a) | El tamaño no importa a nivel de costes |  |
| --- | --- | --- |
| (b) | El tamaño afecta sólo al número de comparaciones |  |
| (c) | El tamaño afecta sólo al número de asignaciones |  |
| (d) | El tamaño afecta a los dos tipos de operaciones | **X** |
| (e) | El tamaño afecta la cantidad de memoria consumida |  |
|  |

*Tabla 4.2: Factor de incremento del número de operaciones al crecer el tamaño de la cantidad de objetos.*

| Talla | Factor talla | Factor Asig | Factor Comp | Cantidad de memoria consumida | Cantidad de líneas ejecutadas |
| --- | --- | --- | --- | --- | --- |
| Del primer al segundo tamaño | 1 | 1 | 1 | 1 | 1 |
| Del segundo al tercer tamaño | 2 | 1 | 2 | 2 | 1 |

El comportamiento esperado del algoritmo será que, en general:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | O(1) | O(n) | Otro\_\_\_\_\_\_\_\_ |  |
| Las asignaciones crecen |  | **X** |  | Con respecto a la talla |
| Las comparaciones crecen |  | **X** |  | con respecto a la talla |
| Cantidad de memoria consumida |  | **X** |  | con respecto a la talla |
| Cantidad de líneas ejecutadas crece |  | **X** |  | con respecto a la talla |

*Tabla 5.1: Método de Mochila:* ***programación dinámica.***

| Operaciones |
| --- |
| Cantidad de objetos (vértices) | | |
| Primer tamaño | Segundo tamaño | Tercer tamaño |
| Asignaciones | 1159 | 1159 | 62584 |
| Comparaciones | 753 | 753 | 39911 |
| Cantidad de líneas de código ejecutadas | 1912 | 1912 | 102459 |
| Cantidad de líneas de código. | 20 | | |
| Capacidad mochila | 5 | 25 | 50 |
| Cantidad de objetos dentro de la mochila | 3 | 4 | 8 |
| Máximo beneficio dentro de la mochila | 185 | 275 | 568 |
| Suma de los pesos dentro de la mochila. | 5 | 22 | 46 |
| Suma de todos los beneficios. | 260 | 355 | 612 |
| Cantidad de posibilidades analizadas | 32 | 32 | 1024 |
| Cantidad de memoria consumida | 1392 | 1392 | 65520 |

¿Según los resultados que has obtenido, cómo se comporta el algoritmo de la mochila con el diseño de programación dinámica en función de la cantidad objetos? Marcar con X la respuesta que coincida con tu caso.

| (a) | El tamaño no importa a nivel de costes |  |
| --- | --- | --- |
| (b) | El tamaño afecta sólo al número de comparaciones |  |
| (c) | El tamaño afecta sólo al número de asignaciones |  |
| (d) | El tamaño afecta a los dos tipos de operaciones | **X** |
| (e) | El tamaño afecta la cantidad de memoria consumida | **X** |
|  |

*Tabla 4.2: Factor de incremento del número de operaciones al crecer el tamaño de la cantidad de objetos.*

| Talla | Factor talla | Factor Asig | Factor Comp | Cantidad de memoria consumida | Cantidad de líneas ejecutadas |
| --- | --- | --- | --- | --- | --- |
| Del primer al segundo tamaño | 1 | 1 | 1 | 1 | 1 |
| Del segundo al tercer tamaño | 2 | 54 | 53 | 47 | 54 |

El comportamiento esperado del algoritmo será que, en general:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | O(1) | O(n) | Otro n6 |  |
| Las asignaciones crecen |  |  | **X** | Con respecto a la talla |
| Las comparaciones crecen |  |  | **X** | con respecto a la talla |
| Cantidad de memoria consumida |  |  | **X** | con respecto a la talla |
| Cantidad de líneas ejecutadas crece |  |  | **X** | con respecto a la talla |

¿Los datos de la tabla 1.1, 2.1, 3.1, coinciden en? Marcar con X la respuesta que coincida con tu caso.

| (a) | Coinciden todos los resultados |  |
| --- | --- | --- |
| (b) | Sólo coinciden las asignaciones |  |
| (c) | Sólo coinciden las comparaciones |  |
| (e) | Sólo coincide en la cantidad de memoria consumida. |  |
| (f) | No coinciden ninguno de los resultados | **X** |

Justifica el resultado obtenido en la cuestión anterior

| Al analizar las tablas nos damos cuenta de que los valores no coinciden puesto que cada uno de ellos se ejecuta de diferente manera, haciendo así el mejor resultado según lo planteado en el problema, aunque en el método voraz no siempre va tener una respuesta requerida, pero los otras dos algoritmos utilizados deberían ser igual en cuanto a resultado; pero en asignaciones, comparaciones y cantidad de memoria no lo son puesto que el proceso para llegar a la respuesta que requiere el problema es diferente, es decir, mientras se recorre el grafo en el voraz de una forma en el back tracking lo hace de otras y estas de otra diferentes al ramificación y poda, siendo así las asignaciones, comparaciones y cantidad de memoria, en el ramificación y poda deben de disminuir de una forma gradual conforme aumenta la cantidad de arcos y esto también ocurre en voraz. |
| --- |

¿Los datos de la tabla 4.1, 5.1 coinciden en? Marcar con X la respuesta que coincida con tu caso.

| (a) | Coinciden todos los resultados |  |
| --- | --- | --- |
| (b) | Sólo coinciden las asignaciones |  |
| (c) | Sólo coinciden las comparaciones |  |
| (e) | Sólo coincide en la cantidad de memoria consumida. |  |
| (f) | No coinciden ninguno de los resultados | **X** |

Justifica el resultado obtenido en la cuestión anterior

| Según lo analizado podemos ver que el voraz tiene menos asignaciones, comparaciones y cantidad de memoria que el programación dinámica, ya que el dinámico a pesar de que es el que da la respuesta más óptima sacrifica el uso de almacenamiento, entonces por ende, muchas veces el voraz no va dar la respuesta correcta, contrario al dinámico que siempre lo hace. Y no coinciden puesto que el manejo de las asignaciones, comparaciones y cantidad de memoria es de forma diferente. |
| --- |

# Conclusiones

Al realizar las comprobaciones de los resultados de los algoritmos, podemos observar una gran diferencia entre los resultados. Esta consiste mayoritariamente en la bipolaridad que generan métodos muy distintos. Por un lado el voraz realiza un procedimiento muy lineal y rápido, su consumo de memoria y de líneas totales es bastante bajo, mientras que los métodos de Back Tracking, ramificación y poda y programación dinámica son más lentos y consumen más memoria, tiempo y recursos, pero obtienen un resultado optimizado.

Para especificar un poco más, en el método del viajero se nota mucho la diferencia de resultados, a pesar de que este dependa mucho en el origen, en la mayor parte de los casos se notó que la ruta final variaba con respecto a la ruta generada por los métodos de Back Tracking y de ramificación y poda, los cuales por su parte generan siempre soluciones similares, lo que se puede suponer debido a que son métodos bastante parecidos, en que ramificación y poda es simplemente una optimización del otro. El consumo de tiempo y memoria es más alto con estos últimos dos, mientras que en voraz se consume mucho menos, sin embargo el resultado casi siempre es más óptimo en los primeros, mientras que en voraz algunas veces ni siquiera se logra encontrar ninguna ruta.

Por el lado del problema de la mochila, la polaridad es similar, pues el método voraz se comporta igual que en el problema anterior, realizando la solución muy rápidamente, pero con un resultado que no siempre es el mejor. El otro método, programación dinámica, actúa de forma contraria. Consume más tiempo y recursos, pero logra encontrar un resultado más acertado.

# Recomendaciones

* Una de las recomendaciones más importantes que se nos vienen a la mente es que el proyecto tenga un poco más de sentido práctico o útil. Sin embargo se puede destacar que este se veía más útil que el anterior, ya que al menos se basaba en solución de problemas. Aun así se podría realizar algo más práctico que todavía sirva para evaluar lo que se necesita.
* Con respecto a la explicación en clase, creo que se podría discutir un poco más la utilidad de estos algoritmos, así como el funcionamiento preciso de cada uno y estrategias que se puedan utilizar en el momento de implementarlos.
* Además de lo anterior se puede recomendar que se entregue un adelanto o el proyecto terminado, pero antes de revisarlo hacer observaciones del mismo sin calificación, para el día de entrega general se puede entregar al 100%, todo lo que por ciertas razones a veces se olvida como son las variables de medición.
* Con respecto al lenguaje podemos decir que bastante interesante y amigable para la programación orientada a objetos y estructuras de almacenamiento como arreglos, estructuras, variables entre otros, se recomienda la explicación más de clases propias del programa y su funcionamiento interno además de la diferencia entre este y sus sucesores C, J# y C# entre otros.

# Literatura citada

* Ejemplo animado de Back Tracking. Revisado en miércoles 1 de junio, 2011.
  + <http://www.hbmeyer.de/backtrack/backtren.htm>
  + Muestra gráficamente como se realiza el método de back tracking sobre un problema sencillo en un tablero con varias fichas.
* Tutorial de Back Tracking con recursión. Consultado el miércoles 1 de junio, 2011.
  + <http://compsci.ca/v3/viewtopic.php?t=21497>
  + Ejemplo de cómo implementar correctamente el algoritmo mencionado con funciones recursivas, escrito en java.
* Algoritmo de Branch And Bound (Ramificar y Acotar) Para la Programación Lineal Entera. Consultado el viernes 3 de junio, 2011.
  + <http://www.arquimedex.com/index.php?accion=1&id=41>
  + Explicación del método de ramificación y poda en programación lineal. Se muestra un árbol de soluciones para ejemplificar gráficamente el proceso normal de este método.
* Ejemplo de Ramificar y acotar. Consultado el viernes 3 de junio, 2011.
  + <http://www.arquimedex.com/index.php?accion=1&id=72>
  + Otro ejemplo práctico del uso del método. Basado en un problema de álgebra lineal de inecuaciones.
* Problema de la mochila en programación voraz. Consultado el 4 de junio de 2011.
  + <http://es.wikipedia.org/wiki/Problema_de_la_mochila>
  + Explicación general de cómo desarrollar el problema de la mochila y como ha sido planteado por varios matemáticos.
* Problema de la mochila (programación voraz). Consultado el 2 de junio de 2011.
  + <https://sercastro.wordpress.com/tag/mochila-por-estimacion-voraz/>
  + Explica como ver el método en una representación en una estructura de datos alternativa.
* Problema de la mochila en programación dinámica. Consultado el 4 de junio de 2011.
  + <http://webysw.blogspot.com/2009/05/pro.html>
  + Resolución de cómo hacer la programación dinámica en problemas como la mochila.
* Función ITOA. Consultado el 5 de junio de 2011 de
  + <http://www.cplusplus.com/forum/beginner/1057/>
  + Convertir de entero a string utilizando cambio de base. Utilizada para método dinámico cuando se realizan las posibles combinaciones.