Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-1221 Estructuras de Datos y Análisis de Algoritmos
Grupo 001

II Tarea programada

Profesora:
Sandra Kikut

Elaborado por:

Andreína Alvarado González | B40259
Otto Mena Kikut | B23753

3 de diciembre del 2015

Índice

1. Introducción

La importancia de esta tarea, radica en que forma una base de conocimientos básicos necesarios, para entrar en el mundo de las técnicas de resolución de problemas.

Específicamente, esta tarea estaría enfocada en la resolución de problemas NP-completos, que son algoritmos que no cuentan aún con un algoritmo eficiente para encontrar su solución óptima.

2. Objetivos

El objetivo de esta tarea, es implementar y de esta manera conocer, las técnicas de resolución de problemas conocidas como: búsqueda Greedy, búsqueda local, búsqueda exhaustiva pura y búsqueda exhaustiva con ramificación y acotamiento.

Los problemas a los que se aplicarían estas técnicas de resolución de problemas son: el problema de encontrar el circuito Hamilton de menor costo, un problema de asignación, un problema de distribución, el problema de las reinas, el problema del coloreo de grafos y un problema de mochila.

3. Enunciado

Implementar un programa que permita resolver problemas mediante búsqueda Greedy, búsqueda local, búsqueda exhaustiva pura y búsqueda exhaustiva con ramificación y acotamiento.

Las tres técnicas de búsqueda deben ser utilizadas para resolver problemas de los siguientes tipos:

- Problema del Vendedor
- Problema de Asignación
- Problema de Distribución

Además se debe usar exclusivamente búsqueda exhaustiva pura para resolver los siguientes tipos de problemas: Reinas, Coloreo de Grafos y Mochila.

4. Desarrollo

4.1. Modelos matemáticos

4.1.1 Modelo grafo no dirigido

4.1.1.1 Definición del modelo

Según el libro *“Estructuras de datos y algoritmos”*, de Alfred V. Aho, Jeffrey D. Ullman y John E. Hopcroft: un grafo no dirigido G = (V,A), es un grafo que consta de un conjunto finito de vértices V y de un conjunto de aristas A, en el que cada arista en A es un par no ordenado de vértices “t (buscar símbolo)”. Si (v,w) es una arista no dirigida, entonces (v,w) = (w,v).

4.1.1.2 Operadores básicos

```
Iniciar(grafo G)
```

Efecto	Iniciar el grafo G como vacío.
Requiere	No aplica.
Modifica	Grafo G.

Destruir(grafo G)

Efecto	Destruye el grafo G, dejándolo inutilizable.
Requiere	Grafo G inicializado.
Modifica	Grafo G.

Vaciar(grafo G)

Efecto	Vacía el grafo G, dejándolo con 0 vértices y 0 aristas.
Requiere	Grafo G inicializado.
Modifica	Grafo G.

Vacío(grafo G)

Efecto	Devuelve verdadero si grafo G está vacío y falso, sino.
Requiere	Grafo G inicializado.
Modifica	No aplica.

Agregar(v1, v2, p, grafo G)

Efecto	Agrega una arista entre el vértice v1 y el vértice v2, con peso p, el grafo G.
Requiere	Grafo G inicializado, v1 y v2 válidos en G y v1 y v2 no adyacentes.
Modifica	Grafo G.

EliminarArista(v1, v2, grafo G)

Efecto	Elimina la arista entre v1 y v2 en el grafo G.

Requiere	Grafo G inicializado, v1 y v2 válidos en G y v1 y v2 adyacentes.
Modifica	Grafo G.

ModifPeso(v1, v2, p, grafo G)

Efecto	Modifica el peso de la arista entre v1 y v2, en el grafo G.
Requiere	Grafo G inicializado, v1 y v2 válidos en G y v1 y v2 adyacentes.
Modifica	Grafo G.

Peso(v1, v2, grafo G)

Efecto	Devuelve el peso de la arista entre v1 y v2, en el grafo G.
Requiere	Grafo G inicializado, v1 y v2 válidos en G y v1 y v2 adyacentes.
Modifica	No aplica.

ExisteArista(v1, v2, p, grafo G)

Efecto	Devuelve verdadero si existe arista entre v1 y v2, y falso sino.
Requiere	Grafo G inicializado y, v1 y v2 válidos en G.
Modifica	No aplica.

AgregarVértice(e, grafo G)

Efecto	Agrega un nuevo vértice con etiqueta e, en el grafo G.
Requiere	Grafo G inicializado.
Modifica	Grafo G.

EliminarVértice(v, grafo G)

Efecto	Elimina el vértice v en el grafo G.
Requiere	Grafo G inicializado, v válido en G y v aislado.
Modifica	Grafo G.

ModifEtiq(v, e, grafo G)

Efecto	Modifica la etiqueta de v por e, en el grafo G.
Requiere	Grafo G inicializado y v válido en G.
Modifica	Grafo G.

Etiqueta(v, grafo G)

Efecto	Devuelve la etiqueta del vértice v, en el grafo G.
Requiere	Grafo G inicializado y v válido en G.
Modifica	No aplica.

NumVert(grafo G)

Efecto	Devuelve el número de vértices, en el grafo G.
Requiere	Grafo G inicializado.
Modifica	No aplica.

NumVertSalida(v, grafo G)

Efecto	Devuelve el número de vértices adyacentes al vértice v, en el grafo G.
Requiere	Grafo G inicializado y v válido en G.
Modifica	No aplica.

NumAristas(grafo G)

Efecto	Devuelve el número de aristas, en el grafo G.
Requiere	Grafo G inicializado.
Modifica	No aplica.

EstaAislado(v, grafo G)

Efecto	Devuelve verdadero si el vértice v está aislado y falso, sino.
Requiere	Grafo G inicializado y v válido en G.
Modifica	No aplica.

PrimerVert(grafo G)

Efecto	Devuelve el primer vértice, en el grafo G.
Requiere	Grafo G inicializado y con al menos un elemento.
Modifica	No aplica.

SigVer(v, grafo G)

Efecto	Devuelve el vértice siguiente al vértice v, en el grafo G. Si no existe vértice, se devolverá vérticeNulo.
Requiere	Grafo G inicializado y v válido en G..
Modifica	No aplica.

PrimerVertAdya(v, grafo G)

Efecto	Devuelve el primer vértice adyacente a v, en el grafo G. Si no existe el vértice, se devolverá vérticeNulo.
Requiere	Grafo G inicializado y v válido en G.
Modifica	No aplica.

SigVertAdya(v1, v2, grafo G)

Efecto	Devuelve el siguiente vértice adyacente a v1, después de v2, en el grafo G.
Requiere	Grafo G inicializado y v válido en G. Si no existe el vértice, se devolverá vérticeNulo.
Modifica	No aplica.

4.1.2 Modelo conjunto de conjuntos

4.1.2.1 Definición del modelo

El modelo conjunto de conjuntos, es un modelo matemático en el que cada elemento es un conjunto en sí mismo.

4.1.2.2 Operadores básicos

Iniciar(conjunto de conjuntos CC)

Efecto	Iniciar el conjunto de conjuntos CC como vacío.
Requiere	No aplica.
Modifica	Conjunto de conjuntos CC.

Destruir(conjunto de conjuntos CC)

Efecto	Destruye el conjunto de conjuntos CC, dejándolo inutilizable.
Requiere	Conjunto de conjuntos CC inicializado.
Modifica	Conjunto de conjuntos CC.

Vaciar(conjunto de conjuntos CC)

Efecto	Vacía el conjunto de conjuntos CC, dejándolo con 0 conjuntos.
Requiere	Conjunto de conjuntos CC inicializado.
Modifica	Conjunto de conjuntos CC.

Vacío(conjunto de conjuntos CC)

Efecto	Devuelve verdadero si conjunto de conjuntos CC está vacío y falso, sino.
Requiere	Conjunto de conjuntos CC inicializado.
Modifica	No aplica.

AgregarConjunto(e, conjunto de conjuntos CC)

Efecto	Agrega un conjunto con etiqueta e, en conjunto de conjuntos CC.
Requiere	Conjunto de conjuntos CC.inicializado.

Modifica	Conjunto de conjuntos CC.

EliminarConjunto(c, conjunto de conjuntos CC)

Efecto	Elimina el conjunto c en conjunto de conjuntos CC.
Requiere	Conjunto de conjuntos CC inicializado y c válido en CC.
Modifica	Conjunto de conjuntos CC.

ExisteConjunto(e, conjunto de conjuntos CC)

Efecto	Devuelve verdadero, si existe un conjunto con etiqueta e en conjunto de conjuntos CC, y falso sino.
Requiere	Conjunto de conjuntos CC inicializado.
Modifica	No aplica.

AgregarElementoConjunto(e,c, conjunto de conjuntos CC)

Efecto	Agrega un elemento con etiqueta e al conjunto c, en conjunto de conjuntos CC.
Requiere	Conjunto de conjuntos CC inicializado y c válido en CC.
Modifica	Conjunto de conjuntos CC.

EliminarElementoConjunto(e,c, conjunto de conjuntos CC)

Efecto	Elimina el elemento con etiqueta e en el conjunto c, en conjunto de conjuntos CC.
Requiere	Conjunto de conjuntos CC inicializado y c válido en CC.
Modifica	Conjunto de conjuntos CC.

PerteneceElementoConjunto(e,c, conjunto de conjuntos CC)

Efecto	Devuelve verdadero si existe elemento con etiqueta e en conjunto c, en conjunto de conjuntos CC.
Requiere	Conjunto de conjuntos CC inicializado y c válido en CC.

Modifica	No aplica.

4.1.2 Modelo diccionario

4.1.2.1 Definición del modelo

El modelo diccionario, es como un conjunto, pero sin operadores fuertes de conjunto como: unión, intersección, …

4.1.2.2 Operadores básicos

Iniciar(diccionario D)

Efecto	Inicia el diccionario D como vacío.
Requiere	No aplica.
Modifica	Diccionario D.

Destruir(diccionario D)

Efecto	Destruye el diccionario D, dejándolo inutilizable.
Requiere	Diccionario D inicializado.
Modifica	Diccionario D.

Vaciar(diccionario D)

Efecto	Vacía el diccionario D, dejándolo con 0 elementos.
Requiere	Diccionario D inicializado.
Modifica	Diccionario D.

Vacío(diccionario D)

Efecto	Devuelve verdadero si diccionario D está vacío y falso, sino.
Requiere	Diccionario D inicializado.
Modifica	No aplica.

AgregarElemento (e, diccionario D)

Efecto	Agrega un elemento con etiqueta e, en diccionario D.
Requiere	Diccionario D inicializado.
Modifica	Diccionario D.

EliminarElemento(e, diccionario D)

Efecto	Elimina el elemento e en diccionario D.
Requiere	Diccionario D inicializado y e válido en D.
Modifica	Diccionario D.

PerteneceElemento (e, diccionario D)

Efecto	Devuelve verdadero, si existe elemento con etiqueta e en diccionario D, y falso sino.
Requiere	Diccionario D inicializado.
Modifica	No aplica.

4.2. Estructuras de datos

4.2.1 Lista de listas

4.2.1.1 Descripción

La lista de listas implementada por punteros, es una estructura de datos que consiste en una lista, que contiene elementos y en donde cada elemento, señala a una lista en donde están otros elementos, con los que comparte algún tipo de relación.

En el caso de que la lista de listas, sea utilizada para implementar un conjunto de conjuntos, consistiría en una lista que contiene a los conjuntos C1, C2, …, Cn, en donde cada conjunto, apuntaría a una lista, que contendrá a los respectivos elementos de cada conjunto.

4.2.2 Lista simplemente enlazada

4.2.2.1 Descripción

La estructura de datos lista simplemente enlazada, es una estructura en memoria dinámica implementada por punteros. Consiste en definir un primer puntero, el cual apuntará a un segundo puntero y así sucesivamente.

En una lista simplemente enlazada, los accesos se hacen a través de punteros, a diferencia del arreglo, en el cual se hacen mediante índices. Además, en la lista simplemente enlazada los elementos son contiguos, es decir, no hay posiciones vacías.

4.2.3 Matriz de adyacencia

4.2.3.1 Descripción

La matriz de adyacencia, usada para implementar un grafo, consiste en un arreglo de una dimensión y un arreglo de dos dimensiones. El arreglo de una dimensión contendrá las etiquetas de los vértices en donde, claramente, los vértices serán enteros. Es decir, la matriz de una dimensión, será la matriz de vértices.

La matriz de dos dimensiones A, vendrá dada por las aristas, en donde la posición A[i][j], contendrá algún tipo de distinción que determine si existe una arista entre el vértice i y el vértice j. Por ejemplo, si las aristas no tuvieran pesos, si existe una arista entre i y j, en la posición A[i][j] estaría contenida un verdadero, es decir, la matriz sería una matriz de booleanos.

Algunas características de la matriz de adyacencia, son:

- El vértice es un entero.
- Se denotará por “M“ el máximo de la matriz.
- Se denotará por “N” el número real de vértices.
- Si no existen lazos, la diagonal no se usa.
- Si el grafo, es un grafo no dirigido, la matriz será una matriz simétrica.

4.3. Problemas

4.3.1 Problema de vendedor

4.3.1.1 Descripción del problema

Según el libro “matematicas discretas” de Kenneth H. Rosen, se dice que un circuito x0,x1,...xn,x0 con n>1 del grafo G es un circuito hamiltoniano si x0,x1,...,xn-1,xn es una camino hamiltoniano.

El problema del vendedor, también llamado “*circuito hamilton de menor costo*” consiste precisamente, en hallar el circuito hamilton en un grafo completo G, tal que el costo requerido por ch_y = xy0,xy1,...,xyn,xy0 con n>1, sea menor ch_y < ch_i, para cualquier otro circuito hamilton ch_i, i != y existente en el grafo G.

Una manera de plantear el problema del vendedor, es la siguiente:

“Un viajante quiere visitar exactamente una vez cada ciudad de un conjunto de n ciudades para finalmente regresar al punto de partida. El problema del vendedor o “problema del viajante” pide determinar un circuito de peso total mínimo de un grafo ponderado, completo y no dirigido que visita cada vértice exactamente una vez y regresa al punto de partida.”

4.3.2 Problema de asignación

4.3.2.1 Descripción del problema

Los problemas de asignación se definen en base a la asignación de los artículos de un conjunto a los otros de otro conjunto, de manera que se realice una optimización de los valores entre estas asignaciones. Como ejemplo del

mundo real, estos conjuntos pueden ser las diferentes tareas que un grupo de trabajadores realizan, en donde la optimización se enfocaría en la asignación de estas tareas de tal forma que se ejecuten con la mejor eficiencia en tiempo, costos, entre otros. De esta manera se tiene la particularidad de emparejar cada artículo de un conjunto con uno solo del otro.

4.3.3 Problema de distribución

4.3.3.1 Descripción del problema

En los problemas de distribución existe una cantidad *t* de algún recurso que se puede dividir en un número *n* de cantidades variables, y en este caso enteras, c ta quel t = i = 1nci. Estas cantidades *c* se distribuyen a un número *w* de regiones, zonas, u otros ítems. Dentro de las diferentes formas de distribución existe un límite máximo y un límite mínimo de la cantidad de *t* que se le puede proporcionar a cada ítem.

La maximización o la minimización de un problema de distribución se refiere a la mejor distribución que se pueda realizar de *t* en *w*, de forma que la suma de los valores asociados a las diferentes particiones de una cantidad cque se le pueda dar a cada ítem de *w* sea la óptima. Es decir, que la suma sea en su total la mínima o la máxima de todas las posibles distribuciones factibles de *t* en *w*.

4.3.4 Problema de las reinas

4.3.4.1 Descripción del problema

El problema de las reinas inicialmente, plantea el colocar a ocho reinas, en un tablero de 8x8, de manera tal, que las reinas no se maten entre sí, sabiendo que las reinas se matan por filas, columnas y diagonales.

Para quitar el problema de que se maten las reinas por fila, simplemente se coloca reina 1 en la fila 1, la reina 2 en la fila 2 ,... y así, hasta la reina 8 en la posición 8. Luego el problema se resume, a colocar ocho reinas, en un tablero de 8x8 de manera, que no se maten entre sí, por columnas ni por diagonales.

Así mismo, el problema se puede generalizar como el colocar n reinas, en un tablero de nxn, de manera tal, que las reinas no se maten entre sí.

4.3.4.2 Problema modificado (optimización)

Para convertir el problema de las reinas en un problema de optimización, básicamente lo que se sigue es asignarle un valor o peso a cada casilla del tablero, de manera tal, que la solución óptima, sea aquella en donde la suma de los valores contenidos en las posiciones finales de cada reina, sea mayor (si se está maximizando) o menor (si se está minimizando), que todas las demás soluciones factibles del problema.

4.3.5 Problema de coloreo de grafos

4.3.5.1 Descripción del problema

Según el libro, “Matemáticas discretas”, de Kenneth H. Rosen, el problema del coloreo de grafos, se puede definir como el problema de

determinar el menor número de colores que deben utilizarse para colorear un mapa, de modo que dos regiones adyacentes, no tengan nunca el mismo color.

Todo mapa plano se puede representar por medio de un grafo. Es por esto, que se puede establecer una relación de correspondencia tal que cada región del mapa se representa mediante un vértice. Una arista conecta dos vértices si las regiones representadas por dichos vértices tienen frontera en común.

Además, se le denomina “número cromático” de un grafo, al número mínimo de colores que se requieren para una coloración del grafo.

4.3.6 Problema de mochila

4.3.6.1 Descripción del problema

El problema de mochila, modela una situación análoga a la de 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 un valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.

4.4. Métodos de búsqueda

4.4.1 Búsqueda Greedy

Para muchos problemas de optimización, usar métodos que encuentren la solución óptima, no siempre es lo que más se desea. La búsqueda exhaustiva greedy, siempre toma la decisión que mejor se ve en el momento. Es decir,

4.4.2 Búsqueda local

La búsqueda local se enfoca en alcanzar una optimización con el uso de la diferentes transformaciones que se le pueden aplicar a una solución factible inicial a un problema. Esta solución factible inicial se genera al azar, aceptándose con el único requisito de ser una solución totalmente factible. Después, dependiendo de cómo se defina las transformaciones a aplicar, se itera un número de estas transformaciones de forma que se explore un conjunto finito de alternativas a la solución factible inicial. Ejecutando esta iteración también se determina la mejor solución factible entre la solución inicial, y el conjunto de las alternativas.

4.4.3 Búsqueda exhaustiva pura

La búsqueda exhaustiva pura de soluciones de un problema alcanza la creación de todas las soluciones factibles. Este tipo de búsqueda es más lenta en completar. Esto se debe a que se explora todas las combinaciones de los factores que se pueden aplicar a la solución de un problema, aún cuando la combinación no termine siendo factible. Es decir, la combinación exhaustiva busca definir un conjunto factible de soluciones, las cuáles se usan para identificar óptima de estas soluciones.

4.4.4 Búsqueda exhaustiva con ramificación y acotamiento

Así como la búsqueda exhaustiva pura busca encontrar el conjunto de soluciones factibles a un problema, la búsqueda exhaustiva con ramificación y acotamiento mantiene este objetivo pero con un abordaje distinto. La diferencia en abordaje se debe a la idea de usar un proceso de acotamiento en las posibles combinaciones de factores que afectan a la solución, de forma que solo se ejecute las combinaciones que tengan la mayor esperanza de alcanzar soluciones factibles. En general, la búsqueda exhaustiva con ramificación y acotamiento considera todas las soluciones factibles pero no necesariamente las construye a todas. En el peor de los casos, si el tipo de entrada es tal que complique el acotamiento, la búsqueda exhaustiva con ramificación y acotamiento puede ser peor la búsqueda exhaustiva pura.

5. Manual de Usuario

5.1 Requerimientos de Hardware

- Procesador Inter(R) Core(TM) i5-4200U CPU @ 1.60GHz
- Ram 6,00 GB

5.2 Requerimientos de Software

- Sistema operativo Windows 10
- Arquitectura 64 bits
- Ambiente Code::Blocks

5.3 Arquitectura de programa

- 64 bits

5.4 Compilación

- GNU GCC Compiler

5.5 Especificación de las funciones de programa

- Ingresar sólo datos válidos.
- Tome en cuenta todas las indicaciones que se le dicen.
- Todos los problemas tienen un tamaño de problema máximo que de superarse, podría hacer que el programa no funcione o funcione de manera incorrecta.

6. Análisis de algoritmos

6.1 Casos de estudio, tipos de entrada, tamaños de entrada, diseño de experimentos

Para los diferentes algoritmos, se utilizaron distintos tamaños para resolver los problemas. Todos los algoritmos funcionan con enteros, tanto el peso de las aristas en los vértices, como los vértices en sí. De igual manera, para los algoritmos que no resuelven problemas de grafos, se utilizaron enteros.

Además de los probar con distintos n's se analiza las soluciones factibles obtenidas en los algoritmos que son capaces de calcular varias soluciones factibles.

6.2 Datos encontrados, tiempos y espacio presentados en tablas y gráficos

Coloreo de grafos: es n^n.

n	Coloreo								
7	Tiempo	0.312	0.218	0.39	0.562	0.125	0.234	0.14	0.187
	Sol.Fact	182322	112560	220500	381024	49560	123060	49560	81480
8	Tiempo	8.069	8.188	8.557	1.387	8.56	3.384	10.24	1.703
	Sol.Fact	5059208	4440296	5073656	54600	5073656	1534176	6588344	801360
9	Tiempo	34.64	310.867	79.111	34.659	152.426	46.596	169.091	269.815
	Sol.Fact	16783200	75729024	46062072	16677360	94794336	25728696	104799744	188116992

Tiempos promedios	
7	0.292
8	6.209
9	112.03

Reinas:

n	Reinas									
8	Tiempo	0 s	0.016 s	0.015	0	0	0	0	0	0
	Sol.Fact	92	184	276	368	460	552	644	736	828
9	Tiempo	0.046	0.047	0.047	0.047	0.063	0.062	0.047	0.047	0.047
	Sol.Fact	352	704	1056	1408	1760	2112	2464	2816	3168
10	Tiempo	0.25	0.25	0.235	0.25	0.25	0.25	0.265	0.25	0.25
	Sol.Fact	724	1448	2172	2896	3620	4344	5068	5792	6516
11	Tiempo	1.34	1.313	1.297	1.488	1.391	1.391	1.406	1.383	1.35
	Sol.Fact	2680	5360	8040	10720	13400	16080	18760	21440	24120
12	Tiempo	7.764	7.798	8.24	8.388	8.139	8.184	8.437	8.153	8.356
	Sol.Fact	14200	28400	42600	56800	71000	85200	99400	113600	127800

Mochila: es n^2.

N	Mochila									
20	Tiempo	0,014	0,019	0,02	0,01	0,04	0,010	0,019	0,03	0,02
23	Tiempo	0,71	0,71	0,71	0,9	0,72	0,77	0,81	0,72	0,8
28	Tiempo	2,02	2,2	2,5	2,01	1,98	2,3	2,3	2,3	2,32
30	Tiempo	9,37	8,8	9,28	9,37	9,48	9,42	9,38	10,1	9,34

Distribución: es n!.

N	Distribución. Pura									
7	Tiempo	0,01	0,01	0,01	0,011	0,009	0,01	0,01	0,01	0,01

8	Tiempo	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02
9	Tiempo	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08
12	Tiempo	0,164	0,298	0,0264	0,0239	0,276	0,283	0,187	0,158	0,164

Asignación: es n!.

n	Asignación. Acotamiento									
6	Tiempo	0,024	0,028	0,065	0,014	0,018	0,021	0,049	0,031	0,018
	Sol.Fact	2	2	2	3	2	2	2	3	1
7	Tiempo	0,211	0,114	0,022	0,05	0,37	0,59	0,78	0,017	0,081
	Sol.Fact	1	3	1	2	1	2	2	3	2
8	Tiempo	0,149	0,059	0,024	0,67	0,037	0,02	0,15	0,048	0,57
	Sol.Fact	3	2	3	4	4	2	3	4	2
9	Tiempo	1,07	0,67	0,98	0,21	0,29	0,76	0,89	1,26	0,92
	Sol.Fact	6	3	2	2	2	3	4	7	5

n	Asignación. Pura								
6	Tiempo	0	0	0,009	0,0009	0	0	0	0,009	0
7	Tiempo	0,0009	0,00091	0,0011	0,001	0	0,0009	0,009	0,001	0
8	Tiempo	0,008	0,009	0,01	0,0087	0,008	0,009	0,008	0,0093	0,0078
9	Tiempo	2,3	2,38	2,303	2,3	2,3	2,34	2,29	2,31	2,39

El vendedor: siempre es (n-1)! para un grafo completo.

n	Vendedor. Pura									Media
8	Tiempo	0,0016	0	0	0	0,0001	0,0002	0,00018	0,00019	0,00015	
9	Tiempo	0,09	0,094	0,094	0,098	0,1	0,087	0,093	0,099	0,096	
10	Tiempo	0,34	0,35	0,32	0,34	0,32	0,34	0,32	0,359	0,39	
11	Tiempo	3,655	3,42	3,695	4,011	3,69	3,31	3,41	3,679	3,58	

6.3 Análisis de los datos

Primero que todo, se debe mencionar que no tuvimos tiempo para realizar un análisis completo de los datos. Si bien las pruebas que realizamos para tamaños de problemas y datos del problema muy variados dieron como se esperaba, la mayoría de estas no fueron documentadas. Primero que todo, la cantidad de soluciones factibles encontradas por las búsquedas exhaustivas fue consistente con las calculadas matemáticamente.

Probablemente, lo más interesante de notar es la forma en que los tiempo fueron incrementando. Para cada problema en particular se intentó escoger tamaños de problemas tales que este comportamiento fuera observable pero muchas veces no era fácil ya que en ocasiones se podían observar incrementos de tiempo del orden de 10^2.

En específico se puede observar este tiempo de comportamiento en el problema del coloreo. Se observa que para 7 vértices el tiempo no superaba las decenas de segundo pero al incrementar la cantidad de vértices en dos tenemos tiempos cercanos a los minutos, lo impresionante es que esto tiene sentido, ya que promedio el coloreo de grafos tiene un tiempo de duración de O(n^n).

Otra cosa interesante de notar es el comportamiento que tienen la asignación por BERA y por búsqueda exhaustiva, por ejemplo, vemos que para los n bajos, búsqueda exhaustiva dura menos, ya que BERA toma mucho tiempo calculando todas las cotas, acomodándolas y decidiendo cuales desarrollar. Mientras que para n= 9, BERA empieza a ser bastante más efectivo que búsqueda exhaustiva, porque el tiempo gastado en el cálculo de cotas se ahorra en el sentido de que cada rama que no deba desarrollar son miles de cálculos que no se están efectuando, conforme crezca n, este comportamiento debería verse mucho más acentuado ya que las ramificaciones cada vez tienen más subdivisiones.

Finalmente, cabe mencionar que en bastantes circunstancias el resultado de la búsqueda greedy estuvo cerca de un 10% del mejor resultado, sin embargo, conforme crece la n la búsqueda greedy se aleja mucho más de este valor. Sin embargo para greedy se pudo mantener tiempos cercanos a 0 inclusive para valores de n tan grandes como n=50.

7. Bibliografía

[1] Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft. *Estructura de datos y algoritmos.*1988.

[2] Kenneth H. Rosen. *Matemática discreta*. 2004