MINIPROYECTO 2
EL PROBLEMA DE LA TEORÍA DE LOS SEIS GRADOS DE SEPARACIÓN

Integrantes:
Edgar Hernán Mejía Castro
Código: 0926419 - Plan: 2711
Edwin Ricardo Gamboa
Código: 0926631 – Plan: 2711
Franco Cundar Zambrano
Código: 0810054 – Plan: 3743

Profesor:
Jesús Alexander Aranda Bueno Ph.D.

Universidad del Valle
Escuela de Ingeniería de Sistemas y Computación
Fundamentos de Análisis y Diseño de Algoritmos – Grupo 01
Cali, enero 28 de 2012

TABLA DE CONTENIDO

INTRODUCCIÓN

Con el presente informe se pretende demostrar el problema de la teoría de los seis grados de separación. Para el cumplimiento de tal fin, se desarrollan dos algoritmos que aplican el concepto de programación dinámica y voraz; dichos algoritmos toman como entrada un grafo que representa una red social y genera una salida en términos de porcentaje que representa el grado de satisfactibilidad respecto a los nodos que cumplen con tener caminos con un número de enlaces menor o igual a seis. En este trabajo, se muestra el pseudocódigo de estos algoritmos.

Además, se realiza un análisis de eficiencia teniendo en cuenta los criterios de tiempo y espacio de las operaciones implementadas a partir de la complejidad teórica y practica de los algoritmos. Se describen también los principales detalles de la implementación, la descripción y análisis de los datos de entrada mediante tablas y gráficos para una mejor comprensión, conclusiones y aspectos a mejorar.

El lenguaje utilizado para implementar los algoritmos es Java.

MINIPROYECTO 2
EL PROBLEMA DE LA TEORIA DE LOS SEIS GRADOS DE SEPARACIÓN

1. Detalles de la implementación

Teniendo en cuenta que para probar la teoría de los seis grados de separación, basta con encontrar la longitud de los caminos mínimos entre cada par de vértices y verificar que dicha longitud no sea mayor a 6, entonces se implementaron dos algoritmos que encuentran dichas longitudes, el primero de ellos el Algoritmo de Floyd-Warshall (programación dinámica) y el segundo el Algoritmo de Dijkstra. A continuación se presentan los de implementación de cada uno.

1.1. Algoritmo con programación dinámica

1.1.1. Caracterización de la solución óptima:

El algoritmo de Floyd-Warshall considera los vértices intermedios $<v_2, v_3, v_4, \ldots, v_{i-1}>$ del camino $p<v_1, v_2, v_3, v_4, \ldots, v_{i-1}, v_i>$ más corto entre dos vértices v_1 y v_i. Se basa en la siguiente observación: Sean los nodos de G, V= {1, 2,…, n} y considere el subconjunto {1, 2,…, k} para algún k. Para cualquier par de vértices i,j ϵ V considere todos los caminos de i a j cuyos vértices son todos obtenidos de {1, 2,…, k} y sea p el camino de mínimo peso entre ellos. El algoritmo explota la relación entre el camino p y los caminos cortos de i a j con vértices intermedios en el conjunto {1,2,…, k-1}. Esta relación depende de si k es o no un vértice intermedio de p, de la siguiente manera:

- Su k no es un vértice intermedio de p, entonces todos los vértices intermedios de p están en {1,2,…, k-1}. Por lo tanto, un camino más corto desde i hasta j con todos los vértices intermedios de {1,2,…, k-1}, es también un camino más corto de i hasta j con
- Si k es un vértice intermedio de p, entonces se descompone p en: i~p1~>k~p2~>j. donde p1 y p2 son caminos más cortos con vértices intermedios en {1,2,...,k}.

1.1.2. Definir recursivamente el valor de una solución óptima:

Sea $d_{ij}^{(k)}$ la longitud del camino más corto de i a j con todos sus vértices intermedios en {1,2,…,k}. Cuando k=0 el camino de i a j no tiene vértices intermedios. Entonces,

$$d_{ij}^{(k)} = \begin{cases} \text{longitudes}_{ij} & \text{si } k=0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{ij}^{(k-1)}); & \text{si } k>=1 \end{cases}$$

Siendo longitudes la matriz que mantiene la longitud del camino más corto entre dos vértices i y j.

1.1.3. Calcular el valor de una solución óptima

Basados en la recurrencia podemos calcular la longitud del camino más corto entre cada par de vértices, de la siguiente for:

```
iniciarWarshall(int [][] longitudes)
int n = longitudes.length;
for (int k = 0; k < n; k++) {
      for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                  if(longitudes [i][j]> longitudes [i][k]+ longitudes [k][j]){
                     longitudes [i][j] = longitudes [i][k]+ longitudes [k][j];
             }
       }
}
```

1.2. Algoritmo con programación voraz

Teniendo un grafo conexo con n vértices, sea x el vértice inicial, un arreglo cola de tamaño n guardará al final del algoritmo las distancias desde x al resto de los nodos. El algoritmo funciona así:

a) Inicializa todas las longitudes en *cola* con el valor 9999 relativo ya que son desconocidas al principio, exceptuando la de x que se debe colocar en 0 debido a que la longitud de x a x sería 0.
b) Sea *u* el vértice con valor mínimo en *cola*, y que no ha sido marcado (puesto en true) en el arreglo de booleanos *visitadosVG*. Recorremos todos los vértices adyacentes de *u* excepto los nodos marcados, y que llamaremos *vi*.
c) Si la distancia desde x hasta vi guardada en longitudes es mayor que la distancia desde x hasta u, sumada a la distancia desde a hasta vi; esta se sustituye con la segunda nombrada, esto es:

   si ($longitudes_i$ > longitudesu + $longitudes_{(u, vi)}$)
   entonces $longitudes_i$ = $logitudes_u$ + $longitudes_{(u, vi)}$.

   Esto lo hacemos con el método auxiliar relajación.
d) Marcamos como completo el nodo u en el arreglo *visitadosVG*.
e) Volvemos al paso b) mientras existan nodos no marcados.

Una vez terminado al algoritmo, longitudes estará completamente lleno, con la longitud del camino más corto entre x y todos los demás vértices.
Se repite el procedimiento para cada uno de los vértices.

```
ejecutaDijkstra(matrizAdj[][])
int[] cola;
int n = matrizAdj.length;
nodo_cumple = new int[n];
iniciarNodos();


for (int totalV = 0; totalV < n; totalV++) {
     final boolean [] visitadoVG = new boolean [n];
     iniciarOrigenes(cola,totalV);
     for (int i = 0; i <n; i++) {
              final int u = ExtraerMin.getExtraerMin(cola, visitadoVG);
              visitadoVG[u]=true;
              relajacion(u, cola, matrizAdj, visitadoVG);
     }
}
```

Método Auxiliar 1:

```
getExtraerMin(int [] distancia, boolean [] marca) {
int minimo = 999;
int vertice = 0;

for (int i = 1; i < distancia.length; i++) {
     if (minimo > distancia[i] && !marca[i]) {
          minimo=distancia[i];
           vertice =i;
       }
}
return vertice;
```

Método Auxiliar 2:

```
relajacion(int u, int[] pesosVG, int w[][], boolean[] visitadoVG)
for (int i = 0; i < pesosVG.length; i++) {
if(visitadoVG[i]==false){
   if(pesosVG[i] > (pesosVG[u] + w[u][i])){
        pesosVG[i] = pesosVG[u] + w[u][i];
        caminos[i]+= caminos[u]+","+i+",";
        }
    }
}
```

2. Eficiencia en tiempo

2.1. Análisis de Complejidad Teórica

2.1.1. Algoritmo con programación dinámica

El algoritmo construido utilizando programación dinámica descrito en el punto anterior muestra tres sentencias for cuyo límite de recorrido es el número total de nodos del grafo ingresado, sentencias que además se encuentran anidadas. Las únicas operaciones que realiza se encuentran en la sentencia for más interna y su costo es constante. Por tal razón, presenta una complejidad teórica $\mathbf{\Theta(n^3)}$, siendo n el número de nodos del grafo ingresado.

2.1.2. Algoritmo con programación voraz

El algoritmo construido utilizando programación voraz descrito en el punto anterior muestra dos sentencias for cuyo límite de recorrido es el número total de nodos del grafo ingresado, sentencias que además se encuentran anidadas. Basados solo en esto el algoritmo presentaría una complejidad teórica $\Theta(n^2)$**.** Sin embargo los métodos auxiliares de la sentencia for más interna tienen un costo $\Theta(n)$, por tanto el algoritmo presenta una complejidad teórica $\mathbf{\Theta(n^3)}$, siendo n el número de nodos del grafo ingresado.

2.2. Análisis de Complejidad práctica

Para lograr establecer la complejidad práctica de los dos algoritmos, se realizan una serie experimentos de medición de tiempos de ejecución. Naturalmente, el tiempo de ejecución dependerá del tamaño del grafo, así que el estudio a realizar considerará la dependencia del tiempo en función de dicho tamaño.

Aunque es posible llevar a cabo la medición del tiempo de ejecución de los algoritmos utilizando un cronómetro: pulsando el botón de inicio justo en el momento en que ejecutamos nuestra aplicación y pulsando el botón de parada cuando se muestra el grado de satisfactibilidad del grafo respecto a la teoría, estaríamos midiendo el tiempo total de ejecución del programa; entonces, la medición se lleva a cabo únicamente teniendo en cuenta el tiempo en que se ejecuta el algoritmo tanto voraz como dinámico, es decir, el tiempo medido desde que se inicia el método configurarWarhall(grafo) en el caso del algoritmo dinámico y el método setMatrizAdj(int[ ][ ] matrizAdj) en el caso del algoritmo voraz, así hasta que se ha terminado el cálculo del porcentaje de satisfactibilidad de la teoría, gracias al resultado generado por datos de tipo Date mediante el método getTime.

Debido al grado de error que pueden tener los resultados obtenidos, para los dos algoritmos se realizarán dos mediciones por cada ejecución y se sacará la media de los tres datos obtenidos más su respectiva desviación estándar.

A continuación se detallan los resultados obtenidos de las mediciones del tiempo en segundos para los dos algoritmos, dado un grafo cuyo tamaño está dado por el número de nodos que este contiene. Dichas mediciones se harán variando el porcentaje de conocimiento entre nodos (cantidad de relaciones).

2.2.1. Algoritmo con programación dinámica

Datos obtenidos:
Tablas:

Porcentaje de conocimiento entre nodos: 20%				
Tamaño del grafo(número de nodos)	Tiempo de ejecución 1 (seg)	Tiempo de ejecución 2 (seg)	Tiempo medio (seg)	Desviación estándar
50	0.001	0.002	0.0015	0.0007
100	0.006	0.005	0.0055	0.0007
150	0.019	0.019	0.019	0
200	0.044	0.043	0.0435	0.0007
250	0.083	0.086	0.0845	0.0021
300	0.147	0.142	0.1445	0.0035
350	0.224	0.228	0.226	0.0028
400	0.334	0.338	0.336	0.0028
450	0.478	0.500	0.489	0.0156
500	0.657	0.661	0.659	0.0028
550	0.925	0.923	0.924	0.0014
600	1.275	1.275	1.275	0
650	1.523	1.522	1.5225	0.0007
700	1.885	1.991	1.938	0.075
750	2.367	2.360	2.3635	0.0049
800	2.819	2.830	2.8245	0.0078
850	3.838	3.832	3.835	0.0042
900	4.028	4.046	4.037	0.0127
950	4.790	4.792	4.791	0.0014
1000	5.494	5.505	5.4995	0.0078
1050	7.199	7.204	7.2015	0.0035
1100	8.304	8.331	8.3175	0.0191
1150	10.262	10.377	10.3195	0.0813
1200	10.74	10.651	10.6955	0.0629
1250	11.585	11.516	11.5505	0.0488
1300	14.559	14.824	14.6915	0.1874
1350	15.646	15.631	15.6385	0.0106
1400	18.940	18.915	18.9275	0.0177
1450	20.82	20.796	20.808	0.017

1500	23.126	23.386	23.256	0.1838
1550	24.266	24.239	24.2525	0.0191
1600	26.876	27.097	26.9865	0.1563
1650	30.245	30.254	30.2495	0.0064
1700	34.544	33.111	33.8275	1.0133
1750	36.422	35.807	36.1145	0.4349
1800	40.396	40.333	40.3645	0.0445
1850	42.382	41.800	42.091	0.4115
1900	46.527	47.147	46.837	0.4384
1950	54.997	52.295	53.646	1.9106
2000	56.729	56.023	56.376	0.4992
2050	58.238	58.03	58.134	0.1471
2100	62.709	62.709	62.709	0
2150	67.432	65.783	66.6075	1.166
2200	70.396	71.382	70.889	0.6972
2250	79.865	80.403	80.134	0.3804

Tabla 1. Comportamiento en tiempo de ejecución del algoritmo dinámico con un 20% de relaciones posibles entre nodos.

Porcentaje de conocimiento entre nodos: 50%				
Tamaño del grafo(número de nodos)	Tiempo de ejecución 1 (seg)	Tiempo de ejecución 2 (seg)	Tiempo medio (seg)	Desviación estándar
50	0.004	0.003	0.0035	0.0007
100	0.005	0.005	0.005	0
150	0.018	0.018	0.018	0
200	0.043	0.042	0.0425	0.0007
250	0.087	0.083	0.085	0.0028
300	0.151	0.152	0.1515	0.0007
350	0.232	0.239	0.2355	0.0049
400	0.348	0.349	0.3485	0.0007
450	0.474	0.473	0.4735	0.0007
500	0.702	0.707	0.7045	0.0035
550	0.967	0.969	0.968	0.0014
600	1.247	1.253	1.25	0.0042
650	1.796	1.806	1.801	0.0071
700	1.993	1.954	1.9735	0.0276
750	2.425	2.463	2.444	0.0269
800	3.578	3.435	3.5065	0.1011
850	3.686	3.625	3.6555	0.0431
900	4.799	4.764	4.7815	0.0247
950	6.087	6.131	6.109	0.0311

1000	6.642	6.714	6.678	0.0509
1050	9.103	8.724	8.9135	0.2680
1100	8.861	8.629	8.745	0.1640
1150	11.193	10.952	11.0725	0.1704
1200	12.991	13.325	13.158	0.2362
1250	13.898	13.857	13.8775	0.0290
1300	15.311	15.097	15.204	0.1513
1350	16.414	16.365	16.3895	0.0346
1400	18.349	18.065	18.207	0.2008
1450	19.976	20.020	19.998	0.0311
1500	25.816	24.642	25.229	0.8301
1550	28.034	28.175	28.1045	0.0997
1600	29.324	29.737	29.5305	0.2920
1650	33.089	32.564	32.8265	0.3712

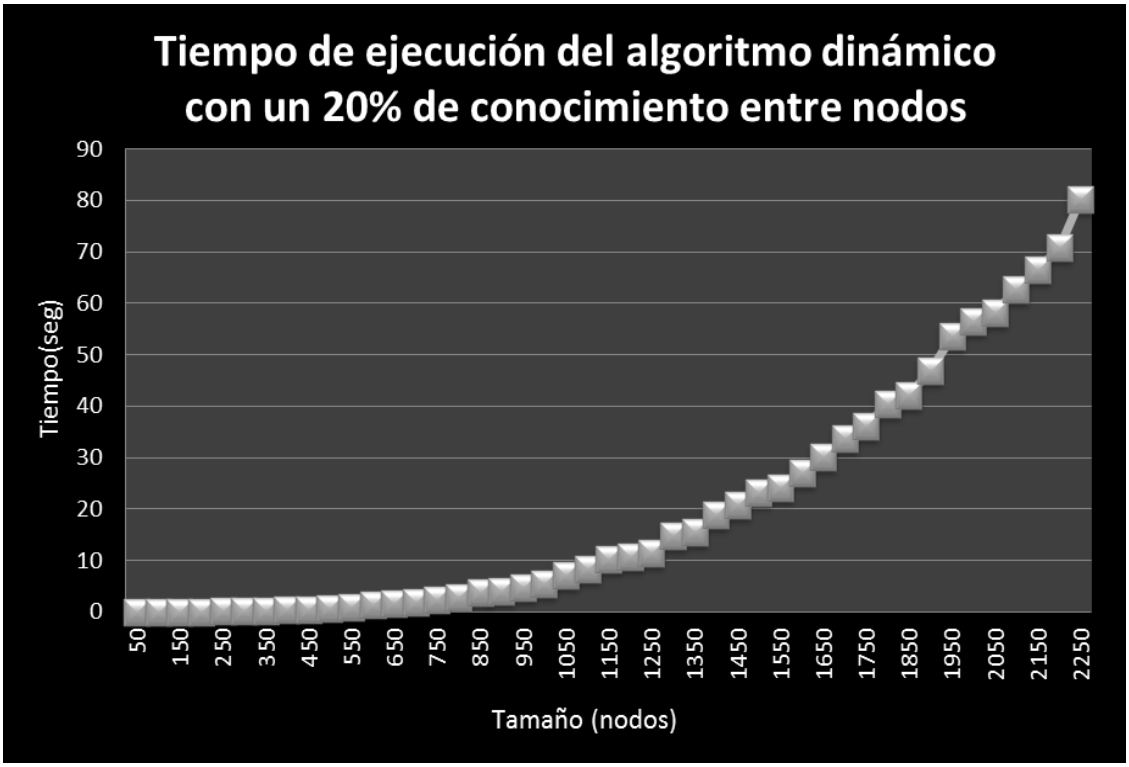
Tabla 2. Comportamiento en tiempo de ejecución del algoritmo dinámico con un 50% de relaciones posibles entre nodos.

Porcentaje de conocimiento entre nodos: 80%				
Tamaño del grafo(número de nodos)	Tiempo de ejecución 1 (seg)	Tiempo de ejecución 2 (seg)	Tiempo medio (seg)	Desviación estándar
50	0.003	0.003	0.003	0
100	0.006	0.006	0.006	0
150	0.019	0.018	0.0185	0.0007
200	0.041	0.047	0.044	0.0042
250	0.085	0.082	0.0835	0.0021
300	0.148	0.148	0.148	0
350	0.258	0.256	0.257	0.0014
400	0.343	0.332	0.3375	0.0078
450	0.542	0.541	0.5415	0.0007
500	0.715	0.763	0.739	0.0339
550	1.088	1.084	1.086	0.0028
600	1.396	1.379	1.3875	0.0120
650	1.736	1.749	1.7425	0.0092
700	2.205	2.215	2.21	0.0071
750	2.667	2.695	2.681	0.0198
800	3.270	3.379	3.3245	0.0771
850	4.393	3.995	4.194	0.2814
900	4.623	4.604	4.6135	0.0134
950	5.503	5.417	5.46	0.0608
1000	6.338	6.284	6.311	0.0382
1050	7.303	7.307	7.305	0.0028

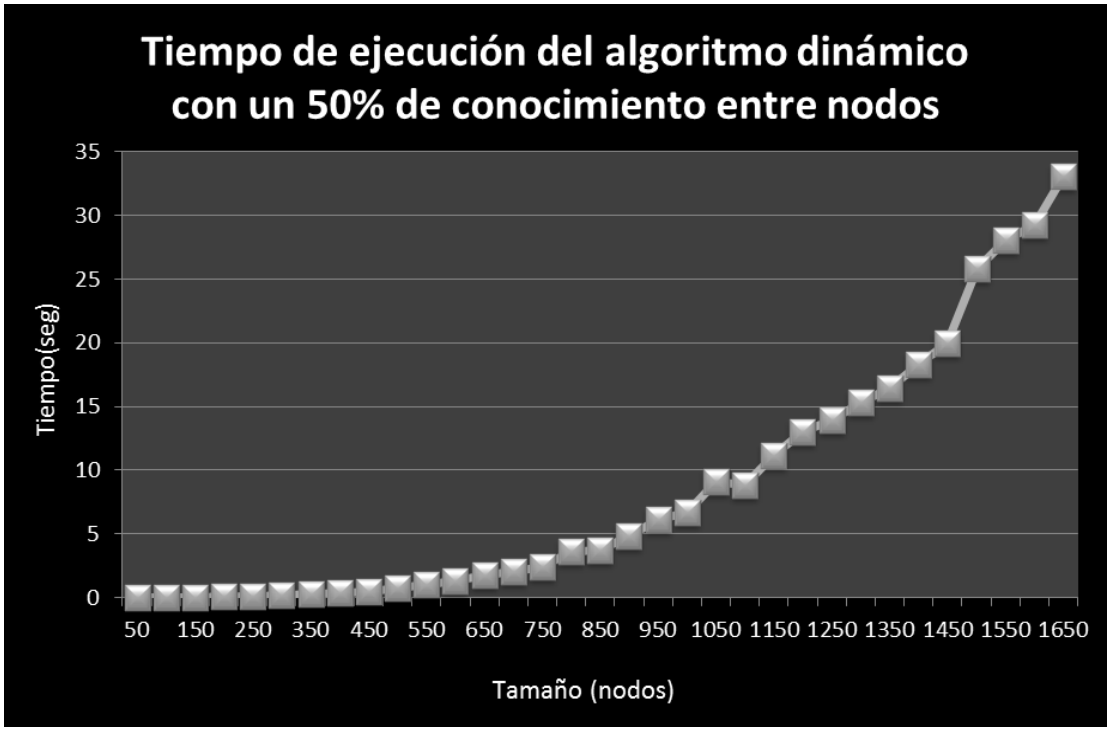
1100	8.413	8.886	8.6495	0.3345
1150	9.827	10.066	9.9465	0.1690
1200	12.304	11.672	11.988	0.4469

Tabla 3. Comportamiento en tiempo de ejecución del algoritmo dinámico con un 80% de relaciones posibles entre nodos.

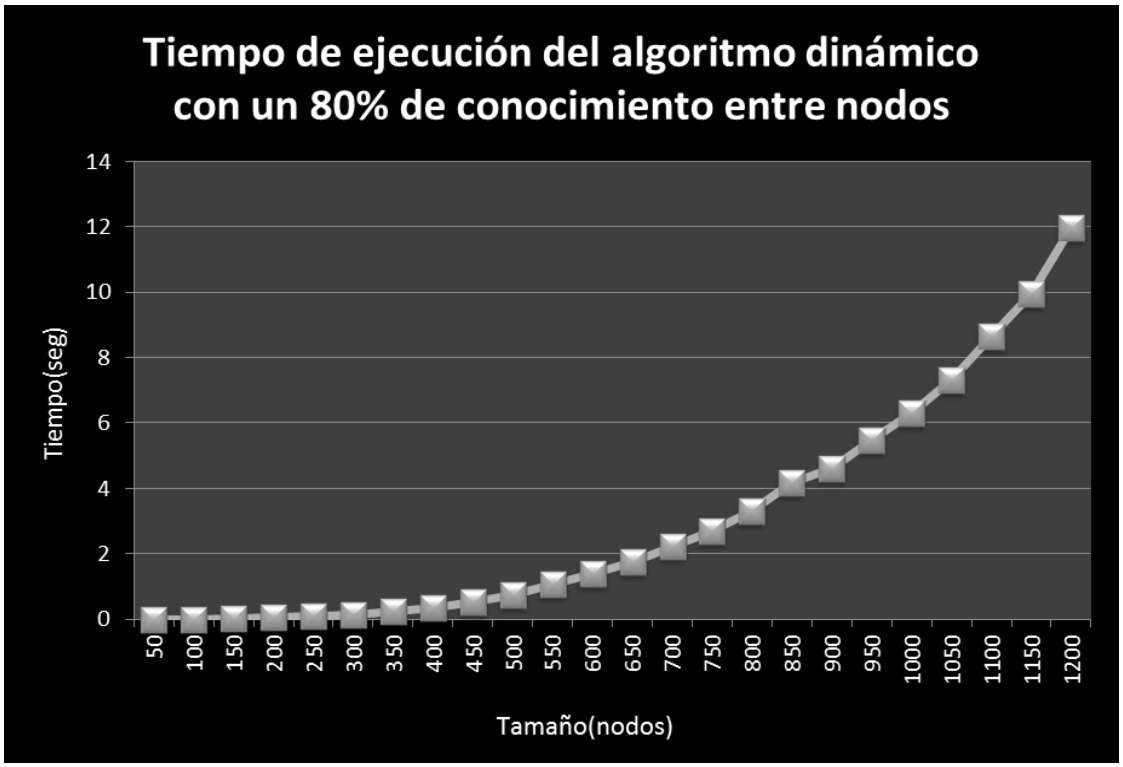
Gráficos:



Gráfica 1. Comportamiento en tiempo de ejecución del algoritmo dinámico con un 20% de relaciones posibles entre nodos (Tabla 1)



Gráfica 2. Comportamiento en tiempo de ejecución del algoritmo dinámico con un 50% de relaciones posibles entre nodos (Tabla 2)



Gráfica 3. Comportamiento en tiempo de ejecución del algoritmo dinámico con un 80% de relaciones posibles entre nodos (Tabla 3)

Análisis de tablas y gráficos

Como podemos observar en los anteriores gráficos (Gráfico 1, Gráfico 2 y Gráfico 3), el algoritmo claramente es cuadrático respecto al tamaño de entrada, es decir, presenta una complejidad temporal $\boldsymbol{\Theta(n^2)}$. Debido a que la complejidad teórica del algoritmo no corresponde con la complejidad práctica, observamos que la forma en que se implementó y las estructuras de control utilizadas, no influyen directamente en el comportamiento real del algoritmo; además, observamos que la complejidad teórica obtenida $\Theta(n^3)$ corresponde al proceso de cálculos realizados por el algoritmo mas no representa el comportamiento respecto al tiempo en ejecución del algoritmo dinámico.

Además, si tenemos en cuenta la cantidad de relaciones entre nodos, observamos que dado un mismo tamaño de entrada el algoritmo dinámico presenta un comportamiento similar para todos los casos. Basándonos en los datos obtenidos después de realizar las pruebas con dicho algoritmo observamos claramente la anterior situación para la mayoría de los casos, pues si tenemos en cuenta que para un grafo cuyo tamaño es de 1100, el algoritmo dinámico realiza el análisis del cumplimiento de la teoría para un 20% de conocimiento entre nodos en un tiempo medio de 8.3175 seg, para un 50% de conocimiento entre nodos en un tiempo medio de 8.745 y para un 80% de conocimiento entre nodos en un tiempo medio de 8.6495. A partir de la anterior comparación, observamos que el algoritmo voraz funciona de la misma manera para un mismo tamaño de entrada independientemente de la cantidad de relaciones existentes entre los nodos del grafo ingresado.

Al realizar las pruebas, también se observó que el tamaño más grande de entrada posible dependía directamente del porcentaje de conocimiento entre nodos. El límite máximo para un porcentaje de conocimiento entre nodos del 20% fue de 2250, para el 50% de conocimiento el límite máximo fue de 1650 y para un 80% de conocimiento entre nodos fue de 1200. Cuando se trató de calcular el tiempo de ejecución para un número más grande a los anteriormente descritos y que corresponda al porcentaje de conocimiento, el IDE (Netbeans) utilizado para el desarrollo del proyecto genera el siguiente error en tiempo de ejecución:

Exception in thread “AWT-EventQueue-0” java.lang.OutOfMemoryError: Java heap space.

Respecto a los resultados consignados en las tres tablas anteriores (Tabla 1, Tabla 2 y Tabla 3), observamos que la diferencia entre dos tiempos medios secuenciales independientemente del porcentaje de conocimiento entre nodos, tiende a ser más grande cuando el tamaño de entrada es muy grande y así mismo, cuando el tamaño de entrada es muy pequeño los tiempos medios obtenidos tienden a estar en un intervalo fijo, en este caso, los tiempos medios

obtenidos a partir de un tamaño de entrada pequeño tienden a estar en el intervalo 0,001 – 0,999.

Hasta un tamaño de límite de entrada sin importar el porcentaje de conocimiento entre nodos, observamos que no hay variación en la ejecución del algoritmo, es decir, que dados como entrada un grafo y un porcentaje respecto a la cantidad de relaciones entre nodos, el tiempo de ejecución depende exclusivamente del tamaño de entrada, aproximadamente hasta que el tamaño es de 1200 nodos en el caso de los tres porcentajes analizados. Para el caso del 20% y 50% de relaciones entre nodos, la anterior propiedad se cumple hasta 1650 nodos.

2.2.2. Algoritmo con programación voraz

Datos obtenidos:
Tablas:

Porcentaje de conocimiento entre nodos: 20%				
Tamaño del grafo(número de nodos)	Tiempo de ejecución 1 (seg)	Tiempo de ejecución 2 (seg)	Tiempo medio (seg)	Desviación estándar
50	0.004	0.002	0.003	0.0014
100	0.013	0.012	0.0125	0.0007
150	0.035	0.037	0.036	0.0014
200	0.079	0.079	0.079	0
250	0.160	0.163	0.1615	0.0021
300	0.246	0.244	0.245	0.0014
350	0.369	0.381	0.375	0.0085
400	0.610	0.612	0.611	0.0014
450	0.853	0.855	0.854	0.0014
500	1.084	1.090	1.087	0.0042
550	1.358	1.334	1.346	0.0170
600	1.786	1.890	1.838	0.0735
650	2.329	2.457	2.393	0.0905
700	3.220	3.093	3.1565	0.0898
750	3.699	3.759	3.729	0.0424
800	4.489	4.334	4.4115	0.1096
850	4.829	4.857	4.843	0.0198
900	6.359	6.424	6.3915	0.0460
950	7.109	6.992	7.0505	0.0827
1000	8.027	8.028	8.0275	0.0007
1050	9.906	10.984	10.445	0.7623
1100	12.746	12.537	12.6415	0.1478
1150	13.939	13.625	13.782	0.2220

1200	16.506	16.624	16.565	0.0834
1250	17.565	18.002	17.7835	0.3090
1300	19.957	19.300	19.6285	0.4646
1350	21.014	20.895	20.9545	0.0841
1400	22.980	22.521	22.7505	0.3246
1450	24.885	24.169	24.527	0.5063
1500	26.504	27.048	26.776	0.3847
1550	29.518	30.015	29.7665	0.3514
1600	32.344	34.214	33.279	1.3223
1650	40.462	39.627	40.0445	0.5904
1700	46.288	45.785	46.0365	0.3557
1750	48.656	48.211	48.4335	0.3147
1800	50.414	51.224	50.819	0.5728
1850	52.663	54.347	53.505	1.1908
1900	57.058	59.779	58.4185	1.9240
1950	64.319	63.412	63.8655	0.6413
2000	66.032	65.942	65.987	0.0636
2050	71.653	70.151	70.902	1.0621
2100	76.961	77.844	77.4025	0.6244
2150	83.829	85.009	84.419	0.8344
2200	91.291	89.987	90.639	0.9221
2250	97.388	98.348	97.868	0.6788
2300	145.265	137.544	141.4045	5.4596
2350	195.953	199.276	197.6145	2.3497

Tabla 4. Comportamiento en tiempo de ejecución del algoritmo voraz con un 20% de relaciones posibles entre nodos.

Porcentaje de conocimiento entre nodos: 50%				
Tamaño del grafo(número de nodos)	Tiempo de ejecución 1 (seg)	Tiempo de ejecución 2 (seg)	Tiempo medio (seg)	Desviación estándar
50	0.002	0.003	0.0025	0.0007
100	0.010	0.011	0.0105	0.0007
150	0.042	0.037	0.0395	0.0035
200	0.101	0.102	0.1015	0.0007
250	0.168	0.182	0.175	0.0099
300	0.296	0.311	0.3035	0.0106
350	0.488	0.449	0.4685	0.0276
400	0.697	0.679	0.688	0.0127
450	0.946	0.933	0.9395	0.0092
500	1.255	1.267	1.261	0.0085
550	1.798	1.862	1.83	0.0453

600	2.126	2.102	2.114	0.0170
650	2.703	2.695	2.699	0.0057
700	3.304	3.364	3.334	0.0424
750	3.921	4.106	4.0135	0.1308
800	4.931	4.820	4.8755	0.0785
850	6.111	6.005	6.058	0.0750
900	7.456	7.467	7.4615	0.0078
950	8.644	8.474	8.559	0.1202
1000	9.792	9.621	9.7065	0.1209
1050	11.455	10.965	11.21	0.3465
1100	13.536	13.187	13.3615	0.2468
1150	14.397	14.893	14.645	0.3507
1200	17.646	16.657	17.1515	0.6993
1250	20.583	19.544	20.0635	0.7347
1300	22.181	23.017	22.599	0.5911
1350	28.312	27.289	27.8005	0.7234
1400	32.019	32.887	32.453	0.6138
1450	34.266	36.478	35.372	1.5641
1500	40.977	43.564	42.2705	1.8293
1550	57.774	59.887	58.8305	1.4941
1600	66.828	68.381	67.6045	1.0981
1650	117.645	115.652	116.6485	1.4093

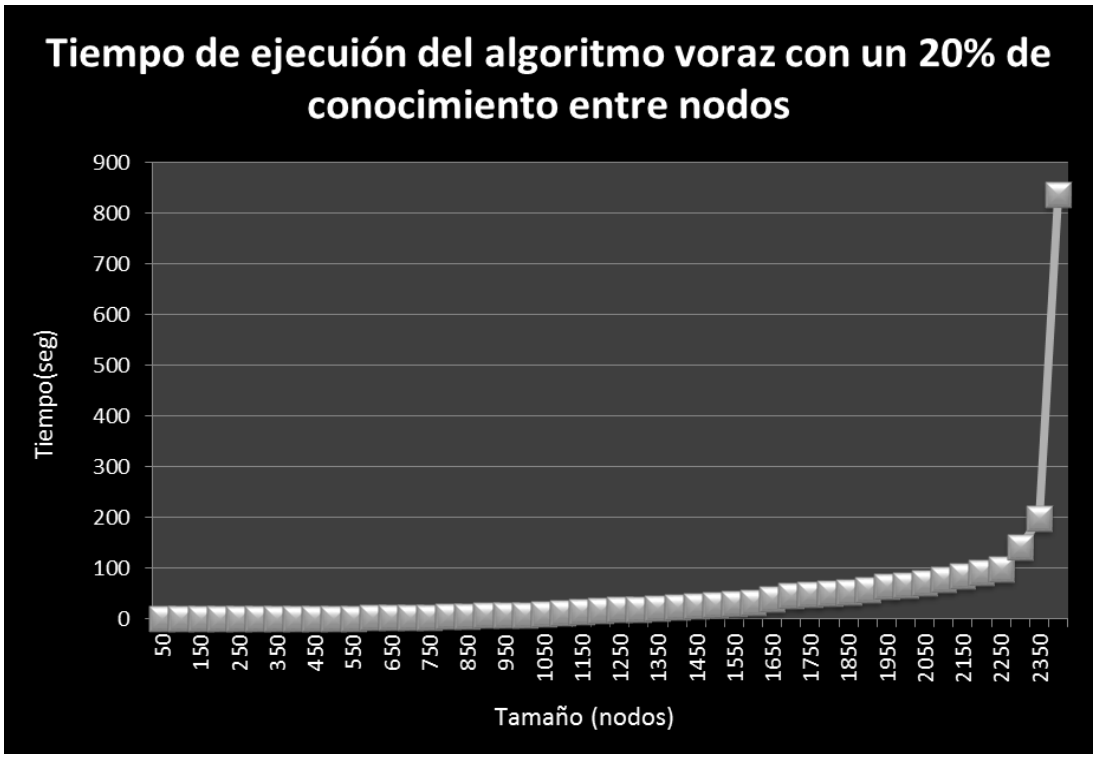
Tabla 5. Comportamiento en tiempo de ejecución del algoritmo voraz con un 50% de relaciones posibles entre nodos.

Porcentaje de conocimiento entre nodos: 80%				
Tamaño del grafo(número de nodos)	Tiempo de ejecución 1 (seg)	Tiempo de ejecución 2 (seg)	Tiempo medio (seg)	Desviación estándar
50	0.004	0.004	0.004	0
100	0.012	0.010	0.011	0.0014
150	0.036	0.042	0.039	0.0042
200	0.099	0.096	0.0975	0.0021
250	0.131	0.147	0.139	0.0113
300	0.287	0.246	0.2665	0.0290
350	0.412	0.413	0.4125	0.0007
400	0.497	0.498	0.4975	0.0007
450	0.809	0.764	0.7865	0.0318
500	1.333	1.282	1.3075	0.0361
550	1.894	1.805	1.8495	0.0629
600	1.988	2.004	1.996	0.0113
650	2.350	2.445	2.3975	0.0672

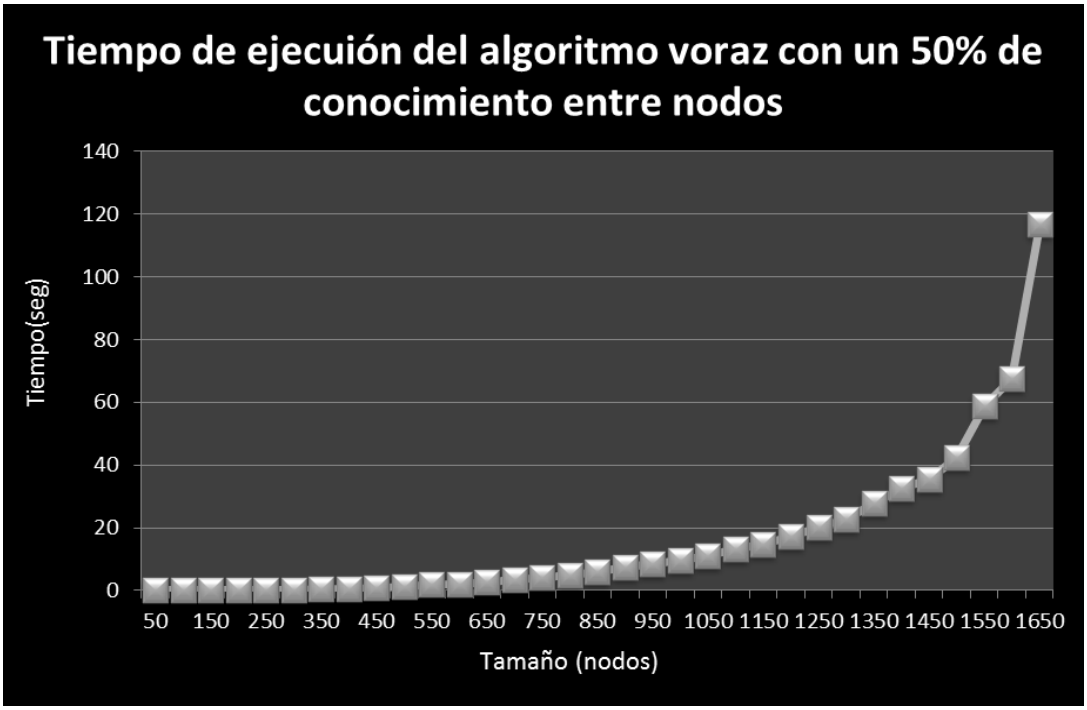
700	3.212	3.365	3.2885	0.1082
750	3.686	3.711	3.6985	0.0177
800	4.577	4.703	4.64	0.0891
850	5.462	5.489	5.4755	0.0191
900	6.861	6.790	6.8255	0.0502
950	7.908	8.142	8.025	0.1655
1000	8.555	8.761	8.658	0.1457
1050	10.924	10.653	10.7885	0.1916
1100	12.440	12.982	12.711	0.3833
1150	15.054	16.357	15.7055	0.9214
1200	19.605	20.351	19.978	0.5275

Tabla 6. Comportamiento en tiempo de ejecución del algoritmo voraz con un 80% de relaciones posibles entre nodos.

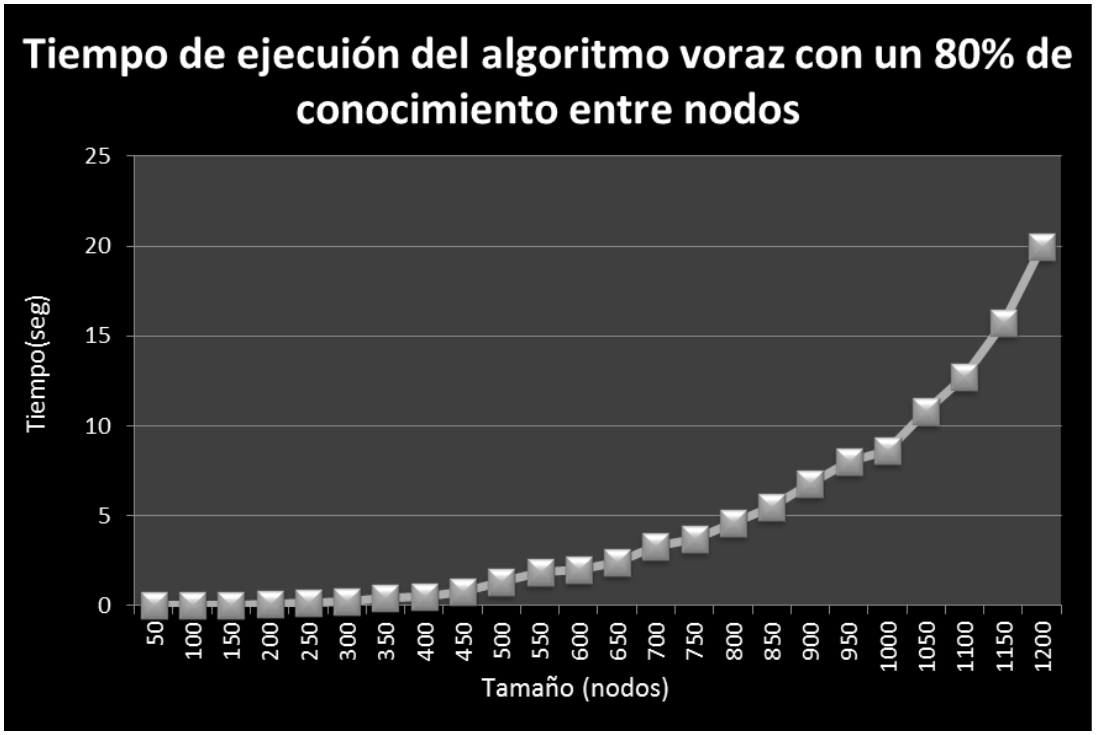
Gráficos:



Gráfica 4. Comportamiento en tiempo de ejecución del algoritmo voraz con un 20% de relaciones posibles entre nodos (Tabla 4)



Gráfica 5. Comportamiento en tiempo de ejecución del algoritmo voraz con un 50% de relaciones posibles entre nodos (Tabla 5)



Gráfica 6. Comportamiento en tiempo de ejecución del algoritmo voraz con un 80% de relaciones posibles entre nodos (Tabla 6)

Análisis de tablas y gráficos

Como podemos observar en los anteriores gráficos (Gráfico 4, Gráfico 5 y Gráfico 6), el algoritmo claramente es cuadrático respecto al tamaño de entrada, es decir, presenta una complejidad temporal $\boldsymbol{\Theta(n^2)}$. Debido a que la complejidad teórica del algoritmo no corresponde con la complejidad práctica, observamos que la forma en que se implementó y las estructuras de control utilizadas, no influyen directamente en el comportamiento real del algoritmo; además, observamos que la complejidad teórica obtenida $\Theta(n^3)$ corresponde al proceso de cálculos realizados por el algoritmo mas no representa el comportamiento respecto al tiempo en ejecución del algoritmo voraz.

Al igual que en el algoritmo dinámico, observamos que dado un mismo tamaño de entrada teniendo en cuenta los tres casos de porcentajes de relaciones posibles entre nodos, el algoritmo voraz presenta un comportamiento similar respecto a tiempo de ejecución. Basándonos en los datos obtenidos después de realizar las pruebas con dicho algoritmo observamos claramente la anterior situación para la mayoría de los casos, pues si tenemos en cuenta que para un grafo cuyo tamaño es de 800, el algoritmo voraz realiza el análisis del cumplimiento de la teoría para un 20% de conocimiento entre nodos en un tiempo medio de 4.4115 seg, para un 50% de conocimiento entre nodos en un tiempo medio de 4.8755 y para un 80% de conocimiento entre nodos en un tiempo medio de 4.64. A partir de la anterior comparación, observamos que el algoritmo voraz funciona de la misma manera para un mismo tamaño de entrada independientemente de la cantidad de relaciones existentes entre los nodos del grafo ingresado.

Al realizar las pruebas, también se observó que el tamaño más grande de entrada posible dependía directamente del porcentaje de conocimiento entre nodos. El límite máximo para un porcentaje de conocimiento entre nodos del 20% fue de 2350, para el 50% de conocimiento el límite máximo fue de 1650 y para un 80% de conocimiento entre nodos fue de 1200. Cuando se trató de calcular el tiempo de ejecución para un número más grande a los anteriormente descritos y que corresponda al porcentaje de conocimiento, el IDE (Netbeans) utilizado para el desarrollo del proyecto genera el siguiente error en tiempo de ejecución:

Exception in thread “AWT-EventQueue-0” java.lang.OutOfMemoryError: Java heap space.

Respecto a los resultados consignados en las tres tablas anteriores (Tabla 1, Tabla 2 y Tabla 3), observamos que la diferencia entre dos tiempos medios secuenciales independientemente del porcentaje de conocimiento entre nodos, tiende a ser más grande cuando el tamaño de entrada es muy grande y así mismo, cuando el tamaño de entrada es muy pequeño los tiempos medios obtenidos tienden a estar en un intervalo fijo, en este caso, los tiempos medios

obtenidos a partir de un tamaño de entrada pequeño tienden a estar en el intervalo 0,001 – 0,999 para los tres casos, hasta que el tamaño de la entrada es de 450 nodos.

Hasta un tamaño de límite de entrada sin importar el porcentaje de conocimiento entre nodos, observamos que no hay variación en la ejecución del algoritmo, es decir, que dados como entrada un grafo y un porcentaje respecto a la cantidad de relaciones entre nodos, el tiempo de ejecución depende exclusivamente del tamaño de entrada, aproximadamente hasta que el tamaño es de 1200 nodos en el caso de los tres porcentajes analizados. Después de este tamaño, los tiempos de ejecución tienden a ser diferentes, así que los resultados ya dependerían exclusivamente del tamaño de entrada del grafo ingresado y de la cantidad de relaciones del mismo.

3. Eficiencia en Espacio

2.1. Análisis de Complejidad Teórica

El espacio requerido por un programa comprende diversos elementos, pero a efectos del análisis del presente algoritmo, se tomará como más relevante el relacionado con la entrada de datos del programa así como el tamaño de las estructuras encargadas de almacenar el resultado de las operaciones a partir de dichos datos.

A continuación, se realiza una breve descripción de las estructuras utilizadas por cada algoritmo teniendo en cuenta su tamaño de almacenamiento, con el fin de verificar el cumplimiento de la teoría de los seis grados. A partir de dicho tamaño se pasa a realizar el análisis sobre la complejidad teórica espacial de los algoritmos que utilizan programación dinámica y voraz respectivamente.

2.1.1. Algoritmo con programación dinámica

Estructuras		
Número	Estructura	Tamaño
1	int[ ][ ] caminosCortos	Cantidad de caminos entre dos nodos
2	int[ ] nodo_cumple	Cantidad de nodos que cumplen con la teoría.
3	int[ ][ ] matrizAdjacente	Cantidad de relaciones entre nodos.

Tabla 7. Estructuras de almacenamiento de datos utilizadas en el algoritmo dinámico

De las estructuras descritas en la anterior tabla (Tabla 7), la estructura número 3 ingresa como parámetro a partir del grafo o red social ingresada. Las estructuras 1 y 2 en un principio no contienen ningún valor pero al final tienen un tamaño que depende del número de nodos que contiene el grafo ingresado.

Los resultados del análisis del cumplimiento de la teoría se guardan en la matriz que en principio había ingresado con información de las relaciones entre nodos. Dicho análisis, obliga a recorrer la matriz tanto a la hora de consignar información como a la hora de sacar la información para llevar a cabo las comparaciones necesarias y teniendo en cuenta todo este proceso, la matriz requiere un tamaño que es proporcional a n^2, por lo tanto el coste espacial teórico es **$\Theta(n^2)$.**

2.1.2. Algoritmo con programación voraz

Estructuras		
Número	Estructura	Tamaño
1	int[ ] nodo_cumple	Cantidad de nodos que cumplen con la teoría.
2	int[ ] cola	Cantidad de caminos más cortos entre dos nodos.
3	int[ ][ ] matrizAdjacente	Cantidad de relaciones entre todos nodos.

Tabla 8. Estructuras de almacenamiento de datos utilizadas en el algoritmo dinámico

De las estructuras descritas en la anterior tabla (Tabla 8), la estructura número 3 ingresa como parámetro a partir del grafo o red social ingresada. Las estructuras 1 y 2 en un principio no contienen ningún valor pero al final tienen un tamaño que depende del número de nodos que contiene el grafo ingresado, es decir, van creciendo linealmente hasta llegar a un punto límite. Entonces, las anteriores estructuras necesitan un tamaño que es proporcional a n, por lo tanto el coste espacial teórico es **$\Theta(n)$.**

2.2. Análisis de Complejidad práctica

Para lograr establecer la complejidad práctica espacial de los algoritmos, se realizan una serie de experimentos de medición de la memoria heap consumida al ejecutarse dichos algoritmos. El estudio a realizar considerará la dependencia de la memoria en función del tamaño de entrada, realizando variaciones respecto a la cantidad de relaciones entre los nodos del grafo.

Es muy importante tener en cuenta que la cantidad de memoria consumida cuando se ejecuta el algoritmo es independiente de la cantidad de memoria que se consume al iniciarse la aplicación y como nuestro interés se enfoca únicamente en la memoria requerida por el algoritmo, cada medición tomada inicia únicamente desde que se hace el llamado al método configurarWarhall(grafo) en el caso del algoritmo dinámico y el método setMatrizAdj(int[ ][ ] matrizAdj) en el caso del algoritmo voraz, así hasta que se ha terminado el cálculo del porcentaje de satisfactibilidad de la teoría. Para generar dichos resultados y para saber si existe concordancia entre los mismos, se hace uso de una herramienta llamada JRockit que muestra la cantidad de memoria heap consumida a medida que se ejecuta el algoritmo. Los resultados que genera la herramienta se muestran en términos de porcentaje y muestran el intervalo de consumo de la memoria heap en un intervalo, así que para efectos de medición, se toma el valor más alto cuando el tiempo de ejecución es grande.

A continuación se detallan los resultados obtenidos de las mediciones de la memoria heap consumida al ejecutarse los algoritmos con programación dinámica y voraz, dado un grafo cuyo tamaño está dado por el número de nodos que este contiene. Dichas mediciones se harán variando el porcentaje de conocimiento entre nodos (cantidad de relaciones). Además, Debido al grado de error que pueden tener los resultados obtenidos, para los dos algoritmos se realizarán dos mediciones por cada ejecución y se sacará la media de los tres datos obtenidos más su respectiva desviación estándar.

2.2.1. Algoritmo con programación dinámica

Datos obtenidos:
Tablas:

Porcentaje de conocimiento entre nodos: 20%				
Tamaño del grafo(número de nodos)	Memoria heap consumida 1(%)	Memoria heap consumida 2(%)	Consumo medio memoria heap (%)	Desviación estándar
50	72.072	74.235	73.1535	1.5295
100	75.305	76.556	75.9305	0.8846
150	80.238	84.271	82.2545	2.8518
200	81.255	78.657	79.956	1.8371
250	79.264	80.244	79.754	0.6930
300	81.363	78.658	80.0105	1.9127
350	83.491	78.545	81.018	3.4974
400	77.834	79.402	78.618	1.1087
450	74.254	79.870	77.062	3.9711

500	76.761	83.241	80.001	4.5821
550	83.922	81.802	82.862	1.4991
600	83.047	81.837	82.442	0.8556
650	81.125	84.625	82.875	2.4749
700	84.541	86.254	85.3975	1.2113
750	84.123	87.946	86.0345	2.7033
800	86.728	86.059	86.3935	0.4731
850	80.719	83.377	82.048	1.8795
900	81.741	79.684	80.7125	1.4545
950	84.750	82.241	83.4955	1.7741
1000	82.205	79.684	80.9445	1.7826
1050	83.541	84.747	84.144	0.8528
1100	80.324	83.659	81.9915	2.3582
1150	84.648	82.647	83.6475	1.4149
1200	80.346	80.647	80.4965	0.2128
1250	83.882	86.321	85.1015	1.7246
1300	81.644	83.553	82.5985	1.3499
1350	82.751	83.254	83.0025	0.3557
1400	85.951	82.460	84.2055	2.4685
1450	80.946	82.652	81.799	1.2063
1500	82.170	84.971	83.5705	1.9806
1550	86.325	84.655	85.49	1.1809
1600	84.821	86.024	85.4225	0.8506
1650	87.694	83.271	85.4825	3.1275
1700	83.251	84.264	83.7575	0.7163
1750	81.554	86.247	83.9005	3.3185
1800	85.766	86.519	86.1425	0.5325
1850	86.974	87.878	87.426	0.6392
1900	84.654	88.212	86.433	2.5159
1950	86.053	84.640	85.3465	0.9991
2000	87.628	85.431	86.5295	1.5535
2050	87.383	85.753	86.568	1.1526
2100	85.014	86.324	85.669	0.9263
2150	86.624	85.544	86.084	0.7637
2200	87.203	86.324	86.7635	0.6215
2250	86.784	85.722	86.253	0.7509

Tabla 9. Comportamiento respecto a la memoria heap consumida por el algoritmo dinámico con un 20% de relaciones posibles entre nodos.

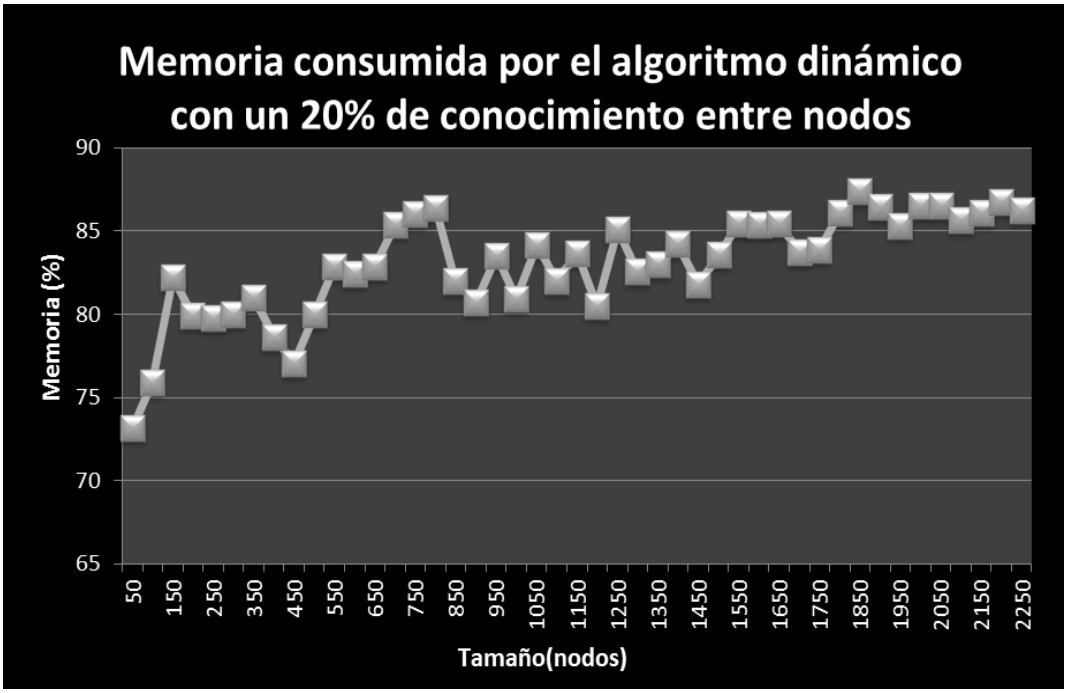
Porcentaje de conocimiento entre nodos: 50%				
Tamaño del grafo(número de nodos)	Memoria heap consumida 1(%)	Memoria heap consumida 2(%)	Consumo medio memoria heap (%)	Desviación estándar
50	72.697	73.645	73.171	0.6703
100	75.621	74.377	74.999	0.8796
150	76.569	72.129	74.349	3.1396
200	70.540	75.694	73.117	3.6444
250	69.365	75.021	72.193	3.9994
300	71.695	73.654	72.6745	1.3852
350	69.621	74.516	72.0685	3.4613
400	71.615	77.628	74.6215	4.2518
450	69.154	74.527	71.8405	3.7993
500	75.621	69.978	72.7995	3.9902
550	70.505	74.651	72.578	2.9317
600	68.954	73.624	71.289	3.3022
650	78.620	74.621	76.6205	2.8277
700	79.627	78.621	79.124	0.7113
750	76.616	79.602	78.109	2.1114
800	77.320	80.648	78.984	2.3533
850	83.654	81.361	82.5075	1.6214
900	80.621	85.989	83.305	3.7957
950	79.692	84.312	82.002	3.2668
1000	85.365	83.676	84.5205	1.1943
1050	84.528	86.952	85.74	1.7140
1100	81.003	81.627	81.315	0.4412
1150	84.158	85.245	84.7015	0.7686
1200	82.828	86.325	84.5765	2.4728
1250	83.902	85.947	84.9245	1.4460
1300	82.934	84.692	83.813	1.2431
1350	86.576	83.624	85.1	2.0874
1400	85.214	86.861	86.0375	1.1646
1450	82.954	84.545	83.7495	1.1250
1500	83.922	86.975	85.4485	2.1588
1550	87.197	87.364	87.2805	0.1181
1600	85.044	86.571	85.8075	1.0798
1650	87.041	86.689	86.865	0.2489

Tabla 10. Comportamiento respecto a la memoria heap consumida por el algoritmo dinámico con un 50% de relaciones posibles entre nodos.

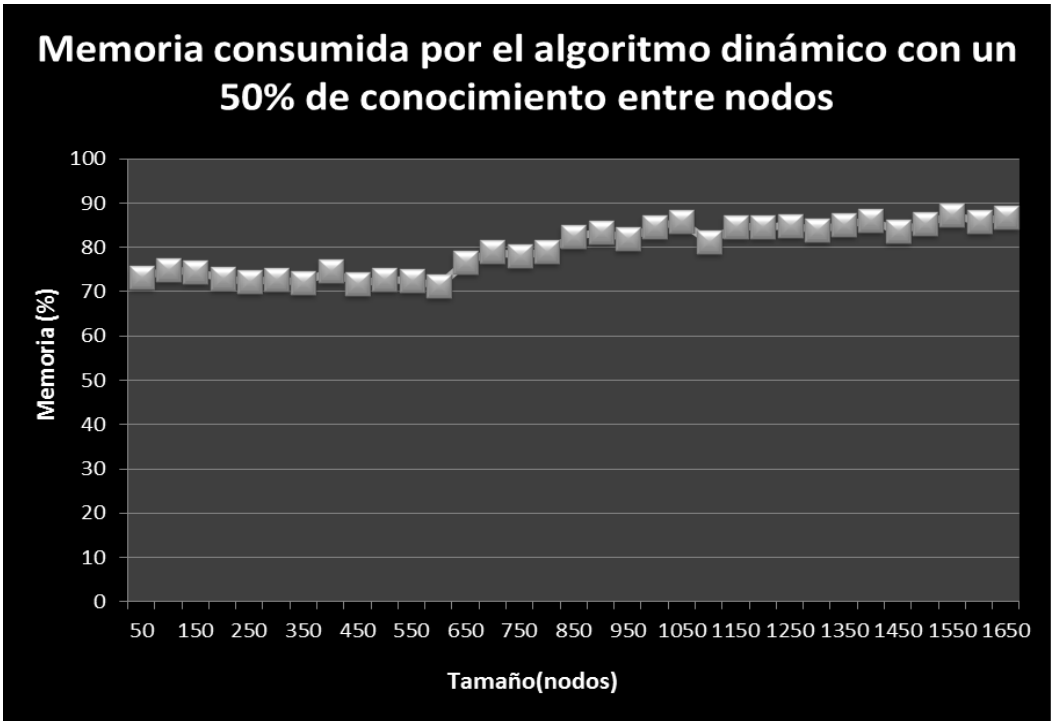
Porcentaje de conocimiento entre nodos: 80%				
Tamaño del grafo(número de nodos)	Memoria heap consumida 1(%)	Memoria heap consumida 2(%)	Consumo medio memoria heap (%)	Desviación estándar
50	80.681	76.624	78.6525	2.8687
100	83.750	81.011	82.3805	1.9368
150	78.545	80.587	79.566	1.4439
200	84.738	80.647	82.6925	2.8928
250	85.643	82.059	83.851	2.5343
300	84.147	83.652	83.8995	0.3500
350	83.984	84.244	84.114	0.1838
400	82.267	86.321	84.294	2.8666
450	83.131	85.612	84.3715	1.7543
500	80.941	88.250	84.5955	5.1682
550	80.273	86.517	83.395	4.4152
600	78.958	85.873	82.4155	4.8896
650	84.026	86.366	85.196	1.6546
700	87.324	88.612	87.968	0.9108
750	85.862	79.221	82.5415	4.6959
800	88.684	83.909	86.2965	3.3764
850	85.218	82.824	84.021	1.6928
900	86.542	79.980	83.261	4.6400
950	85.047	81.013	83.03	2.8525
1000	78.724	86.518	82.621	5.5112
1050	83.014	85.648	84.331	1.8625
1100	84.920	86.740	85.83	1.2869
1150	81.025	84.933	82.979	2.7634
1200	83.989	85.624	84.8065	1.1561

Tabla 11. Comportamiento respecto a la memoria heap consumida por el algoritmo dinámico con un 80% de relaciones posibles entre nodos.

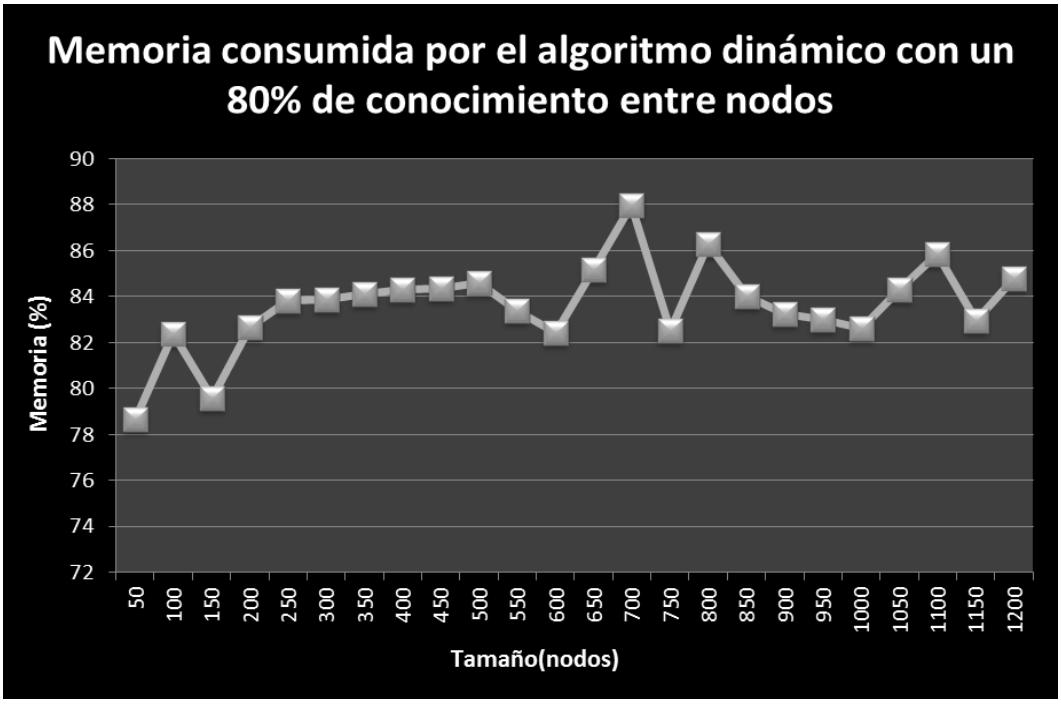
Gráficos:



Gráfica 7. Comportamiento respecto a la memoria consumida por el algoritmo dinámico con un 20% de relaciones posibles entre nodos (Tabla 9)



Gráfica 8. Comportamiento respecto a la memoria consumida por el algoritmo dinámico con un 50% de relaciones posibles entre nodos (Tabla 10)



Gráfica 9. Comportamiento respecto a la memoria consumida por el algoritmo dinámico con un 80% de relaciones posibles entre nodos (Tabla 11)

2.2.2. Algoritmo con programación voraz

Datos obtenidos:
Tablas:

Porcentaje de conocimiento entre nodos: 20%				
Tamaño del grafo(número de nodos)	Memoria heap consumida 1(%)	Memoria heap consumida 2(%)	Consumo medio memoria heap (%)	Desviación estándar
50	77.022	83.273	80.1475	4.4201
100	81.379	80.978	81.1785	0.2835
150	79.696	84.468	82.082	3.3743
200	79.743	84.320	82.0315	3.2364
250	79.848	81.698	80.773	1.3081
300	83.568	79.215	81.3915	3.0780
350	83.891	84.775	84.333	0.6251
400	87.672	79.518	83.595	5.7657
450	83.441	82.165	82.803	0.9023
500	89.979	80.703	85.341	6.5591
550	88.480	85.077	86.7785	2.4063
600	84.245	79.203	81.724	3.5652

650	80.497	88.053	84.275	5.3429
700	83.480	89.618	86.549	4.3402
750	82.858	84.361	83.6095	1.0628
800	85.692	84.184	84.938	1.0663
850	84.862	88.745	86.8035	2.7457
900	88.116	83.432	85.774	3.3121
950	84.928	88.438	86.683	2.4819
1000	79.497	85.563	82.53	4.2893
1050	85.136	80.112	82.624	3.5525
1100	83.022	89.242	86.132	4.3982
1150	81.418	85.066	83.242	2.5795
1200	84.000	88.634	86.317	3.2767
1250	85.789	82.880	84.3345	2.0570
1300	79.634	88.044	83.839	5.9468
1350	87.394	87.057	87.2255	0.2383
1400	79.721	81.461	80.591	1.2304
1450	88.621	81.444	85.0325	5.0749
1500	86.331	79.291	82.811	4.9780
1550	89.552	79.786	84.669	6.9056
1600	84.557	85.340	84.9485	0.5537
1650	89.202	85.089	87.1455	2.9083
1700	88.524	87.323	87.9235	0.8492
1750	81.494	89.591	85.5425	5.7254
1800	84.611	84.337	84.474	0.1937
1850	81.639	85.483	83.561	2.7181
1900	79.494	83.011	81.2525	2.4869
1950	86.892	84.380	85.636	1.7763
2000	85.535	83.817	84.676	1.2148
2050	85.112	84.013	84.5625	0.7771
2100	80.574	84.567	82.5705	2.8235
2150	88.642	83.048	85.845	3.9556
2200	79.050	81.747	80.3985	1.9071
2250	80.761	87.119	83.94	4.4958

Tabla 12. Comportamiento respecto a la memoria heap consumida por el algoritmo voraz con un 20% de relaciones posibles entre nodos

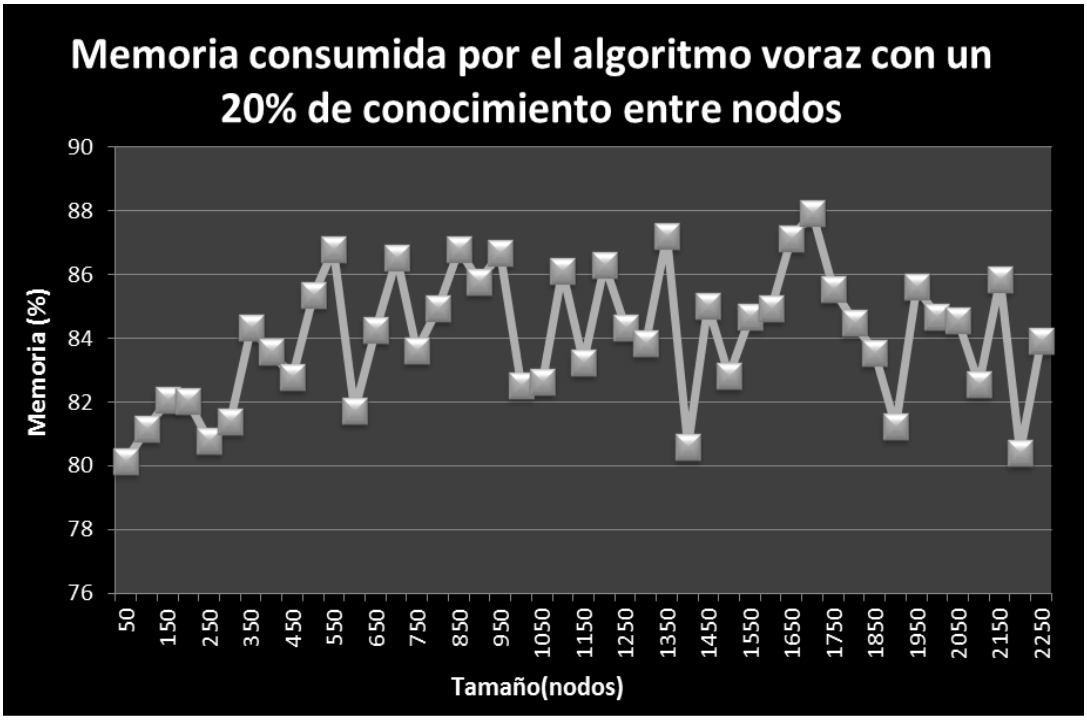
Porcentaje de conocimiento entre nodos: 50%				
Tamaño del grafo(número de nodos)	Memoria heap consumida 1(%)	Memoria heap consumida 2(%)	Consumo medio memoria heap (%)	Desviación estándar
50	88.247	80.432	84.3395	5.5260
100	80.714	85.587	83.1505	3.4457
150	89.163	82.064	85.6135	5.0198
200	86.995	79.911	83.453	5.0091
250	82.546	78.747	80.6465	2.6863
300	88.097	83.928	86.0125	2.9479
350	86.759	80.714	83.7365	4.2745
400	85.220	86.317	85.7685	0.7757
450	80.324	78.335	79.3295	1.4064
500	85.428	84.690	85.059	0.5218
550	84.824	86.198	85.511	0.9716
600	85.254	78.262	81.758	4.9441
650	80.714	84.888	82.801	2.9515
700	81.942	87.332	84.637	3.8113
750	80.489	82.120	81.3045	1.1533
800	81.539	79.152	80.3455	1.6879
850	85.230	83.378	84.304	1.3096
900	86.253	78.441	82.347	5.5239
950	86.722	81.617	84.1695	3.6098
1000	82.869	81.668	82.2685	0.8492
1050	83.678	81.942	82.81	1.2275
1100	81.092	81.623	81.3575	0.3755
1150	89.486	83.860	86.673	3.9782
1200	87.857	82.476	85.1665	3.8049
1250	81.559	85.754	83.6565	2.9663
1300	85.563	79.996	82.7795	3.9365
1350	88.172	81.555	84.8635	4.6789
1400	86.680	81.791	84.2355	3.4570
1450	86.192	88.593	87.3925	1.6978
1500	80.808	80.422	80.615	0.2729
1550	80.520	85.072	82.796	3.2188
1600	79.669	84.242	81.9555	3.2336
1650	89.205	86.519	87.862	1.8993

Tabla 13. Comportamiento respecto a la memoria heap consumida por el algoritmo voraz con un 50% de relaciones posibles entre nodos

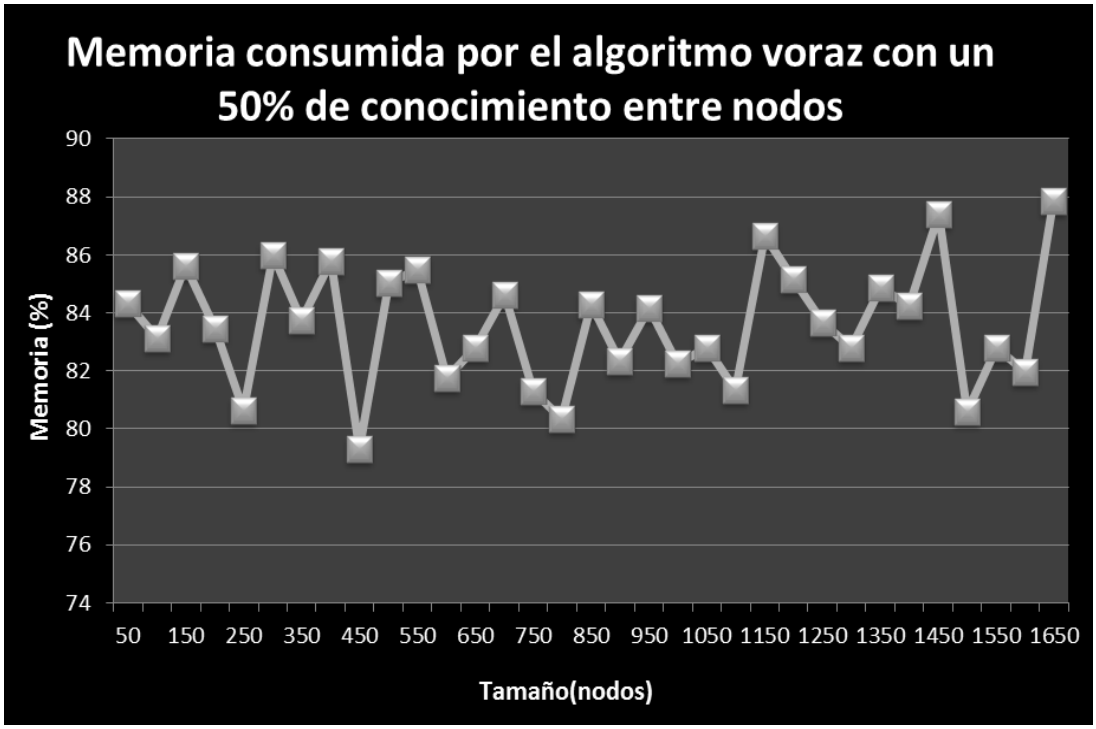
Porcentaje de conocimiento entre nodos: 80%				
Tamaño del grafo(número de nodos)	Memoria heap consumida 1(%)	Memoria heap consumida 2(%)	Consumo medio memoria heap (%)	Desviación estándar
50	82.065	78.868	80.466462	2.2603
100	82.062	86.035	84.048799	2.8094
150	79.019	80.606	79.812494	1.1215
200	88.456	78.141	83.298323	7.2934
250	84.809	81.341	83.075009	2.4519
300	81.984	82.708	82.345718	0.5122
350	80.795	82.666	81.730516	1.3233
400	79.533	82.503	81.017961	2.1006
450	79.023	84.152	81.587816	3.6270
500	88.261	86.382	87.321197	1.3287
550	84.607	86.874	85.740903	1.6029
600	79.621	86.622	83.121485	4.9498
650	80.878	86.798	83.838249	4.1863
700	84.487	86.464	85.475275	1.3981
750	83.272	84.486	83.878907	0.8587
800	86.653	84.106	85.379205	1.8011
850	84.039	84.536	84.287675	0.3510
900	82.284	88.623	85.453095	4.4825
950	86.971	88.322	87.646565	0.9559
1000	88.585	83.854	86.219542	3.3456
1050	85.075	80.261	82.667887	3.4035
1100	83.973	87.161	85.567129	2.2542
1150	78.050	83.315	80.682276	3.7231
1200	79.336	82.673	81.004443	2.3591

Tabla 14. Comportamiento respecto a la memoria heap consumida por el algoritmo voraz con un 80% de relaciones posibles entre nodos

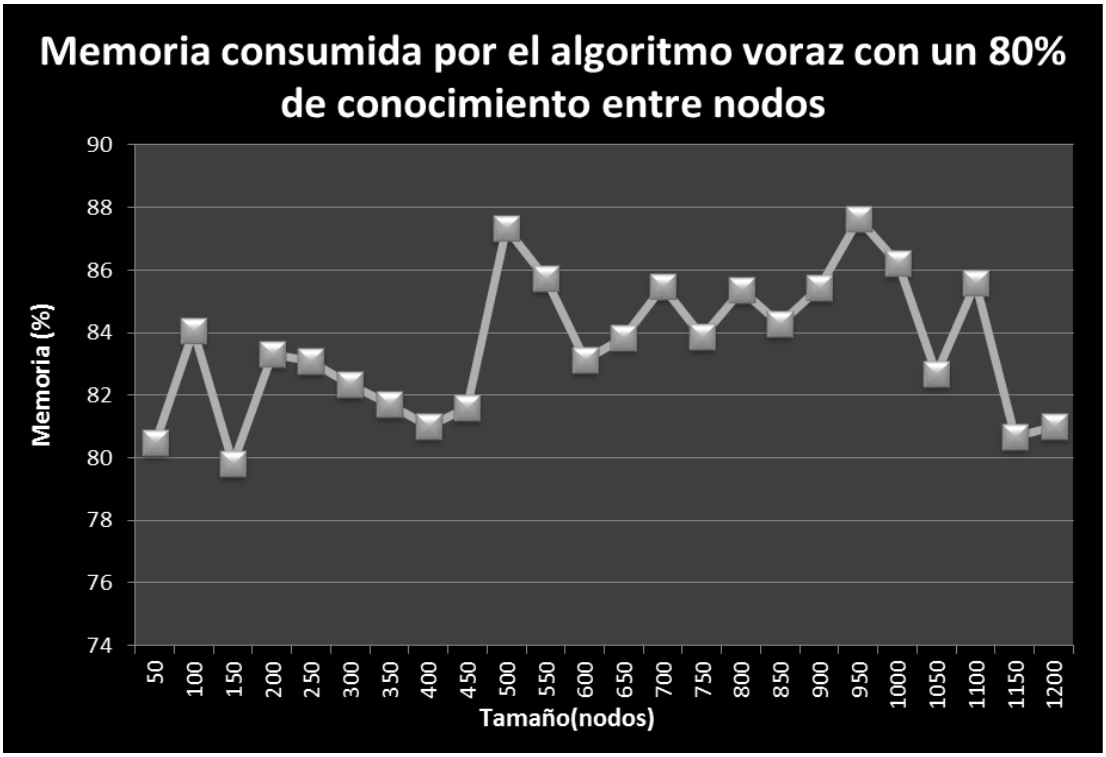
Gráficos:



Gráfica 10. Comportamiento respecto a la memoria consumida por el algoritmo voraz con un 20% de relaciones posibles entre nodos (Tabla 12)



Gráfica 11. Comportamiento respecto a la memoria consumida por el algoritmo voraz con un 50% de relaciones posibles entre nodos (Tabla 13)



Gráfica 12. Comportamiento respecto a la memoria consumida por el algoritmo voraz con un 80% de relaciones posibles entre nodos (Tabla 14)

Análisis de tablas y gráficos:

Como podemos observar en los gráficos correspondientes al consumo de memoria heap por los algoritmos de programación dinámica y voraz (Gráfica 7 a Gráfica 12), los algoritmos presentan un comportamiento constante respecto a la cantidad de memoria heap consumida, es decir, el consumo de memoria heap es independiente del tamaño de entrada y de la cantidad de relaciones entre nodos.

Cuando se realizó la toma de datos respecto a la memoria heap consumida por los algoritmos, observamos que la memoria heap se movía en un rango de consumo de 60 - 89 aproximadamente en términos de porcentaje, con o sin que se estuvieran ejecutando los algoritmos. Para un tamaño de datos grandes resultó más fácil realizar la toma de mediciones, ya que la gráfica generada del consumo de mejoría heap en el tiempo ofrecía un gran número de datos, del cual solo quedaba tomar el número más grande para efectos de medición y análisis.

A continuación se muestra una imagen de la gráfica generada por la herramienta y de la manera en que se obtenían los datos:

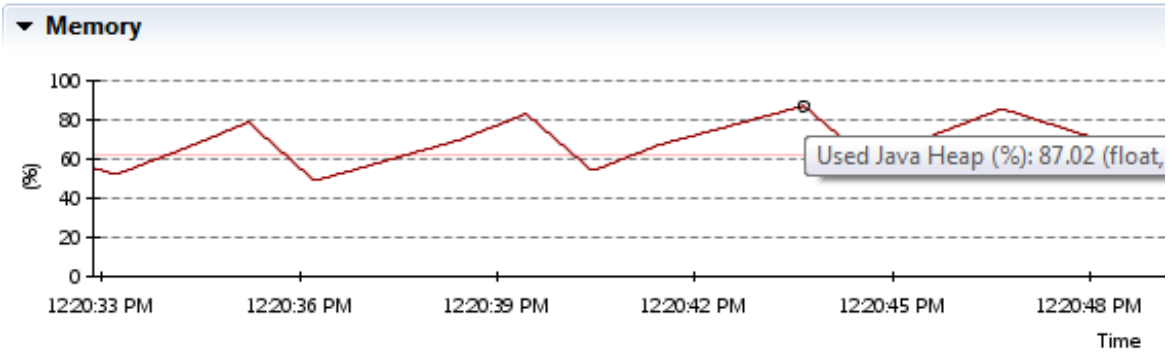


Imagen 1. Porcentaje del consumo de memora heap consumida.

En la anterior imagen corresponde a la manera en que se realizó la toma de datos. La imagen muestra a la cantidad en términos de porcentaje del consumo de memoria heap mientras se ejecuta el algoritmo diámico (87.02%) en el intervalo de tiempo 12:23:33 – 12:20:48.

Después de obtener los datos, observamos que el análisis del cumplimiento de la teoría de los seis grados hecha por los algoritmos no influye en la cantidad de memoria heap consumida, es decir, la memoria heap que se consume a causa de la ejecución de los algoritmos, no afecta de manera crítica el rango en el que se mueve el consumo normal de dicha memoria.

Entonces, observamos que no hay variación respecto a la memoria heap consumida al ejecutarse los algoritmos, es decir, que dados como entrada un grafo y un porcentaje respecto a la cantidad de relaciones entre nodos, la cantidad de memoria heap no depende exclusivamente de estas variables.

Por tal razón, concluimos que la complejidad respecto a la memoria heap consumida por los algoritmos con programación dinámica y voraz presenta una complejidad temporal $\Theta(1)$.

Respecto a la cantidad de espacio demandado al ejecutarse los algoritmos, también se pedía analizar el comportamiento de estos sobre la cantidad de disco requerido. Al realizar la toma de datos, observamos que la cantidad de disco duro requerido al ejecutar los algoritmos es siempre cero e independientemente del tamaño del grafo ingresado o de la cantidad de relaciones entre los nodos ingresados, nunca se requiere la utilización de este tipo de recursos. La explicación de este comportamiento radica en que al ejecutar el algoritmo nunca se guarda ningún tipo de archivo sobre el disco duro ni ocurre otro fenómeno que se relacione con guardar o eliminar algo que ya está fijo en el disco. Entonces, respecto al espacio únicamente se utilizan recursos de memoria.

Respecto a los resultados obtenidos en las complejidades espacial teórica y práctica para los dos algoritmos, vemos que los datos no tienen concordancia, concluyendo que las estructuras de almacenamiento utilizadas y la memoria heap tienen formas diferentes de influir sobre los algoritmos.

4. Otras Cuestiones

1. ¿El desempeño real se ajusta a la complejidad teórica establecida por Ud.?

   El desempeño real en términos del tiempo de ejecución no se ajusta a la complejidad teórica para ninguno de los dos algoritmos, pues ambos presentan un comportamiento cuadrático **$\Theta(n^2)$**, y para ambos la complejidad teórica era **$\Theta(n^3)$.**

2. Analice el comportamiento de estos dos algoritmos, con diferentes grafos (tamaño y topología de la red). Con base en lo anterior caracterice el mejor y el peor caso para los algoritmos en términos de las redes sociales.

   El comportamiento de ambos algoritmos es directamente proporcional a la cantidad de individuos que posee la red social. Sin embargo ninguno se ve afectado por la cantidad de aristas que pueda conocer cada individuo.

   Entonces, el establecimiento del mejor caso para el algoritmo dinámico se podría basar, más bien, en reducir el número de actualizaciones de la matriz de longitudes del camino más corto entre cada par de vértices. Para esto sería necesario que en cada iteración el primer individuo que se considere como intermedio entre el individuo origen y el individuo destino pertenezca al camino más corto. Lo cual no se puede garantizar para todos los caminos pues el algoritmo considera los nodos en orden de acuerdo al identificador (número entero) de cada individuo y no a la cercanía de los mismos. Con el algoritmo voraz sucede algo similar al momento de tomar la decisión local, por lo que tampoco se puede garantizar un mejor caso.

   Entonces, teniendo en cuenta que los algoritmos se comportan solo de acuerdo al número de individuos, no es posible establecer un mejor o peor caso, pues sin importar el número de aristas el dinámico revisara todas las posiciones de la matriz de adyacencia, es decir todas las posibles relaciones entre dos vértices, con base en esto y otras operaciones constantes actualizará la matriz de longitudes da caminos entre dos vértices. Por otro lado el algoritmo voraz para cada individuo utiliza una lista donde se guarda la longitud del camino más corto entre dicho individuo y los demás, y l decisión local se toma a medida que se va explorando cada posible arista entre el individuo y todos los demás. Es decir también evalúa todas las posibles aristas entre cada par de vértices.

3. ¿La cantidad de relaciones (aristas en el grafo) influyen en el desempeño de los algoritmos, si su respuesta es positiva en qué casos los algoritmos se comportan mejor?

Para el caso del algoritmo con programación dinámica, podríamos afirmar que el número de relaciones no influye en su desempeño pues este depende incamente de la cantidad de individuos (vértices) que posea la red social. Pues el algoritmo llena una matriz cuya es (número de individuos)^2, para esto utiliza tres bucles for que se mueven de 0 a n. Es decir sin importar el número de aristas su desempeño sería similar para redes con igual cantidad de individuos.
Podemos verificar esto con los resultados obtenidos, pues como se esperaba el tiempo de ejecución es directamente proporcional a la cantidad de nodos. Con respecto al incremento en la cantidad de la diferencia entre los tiempos medios de las pruebas con igual número de individuos y con 20%, 50% y 80% de conocimiento (porcentaje de aristas que puede conocer un individuo) no supera 2 segundos. Además al observar cuidadosamente los resultados se puede apreciar que en algunas ocasiones el tiempo es mayor para las pruebas con 50% de conocimiento y en otras para el 80%, a continuación algunos ejemplos de este comportamiento:

Tamaño del grafo(número de nodos)	Tiempo medio (seg), con porcentaje de conocimiento de **20%**	Tiempo medio (seg), con porcentaje de conocimiento de **50%**	Tiempo medio (seg), con porcentaje de conocimiento de **80%**
50	0.0015	**0.0035**	0.003
100	0.0055	0.005	**0.006**
150	**0.019**	0.018	0.0185
400	0.336	**0.3485**	0.3375
800	2.8245	**3.5065**	3.3245
850	3.835	3.6555	**4.194**
900	4.037	**4.7815**	4.6135

Como se observa el desempeño del algoritmo dinámico medido en tiempo de ejecución no obedece a un patrón de comportamiento con respecto a la cantidad de relaciones que pueda tener un individuo.

Para el caso del algoritmo voraz, sucede algo similar pues de acuerdo con el código el desempeño del mismo depende solo del tamaño de los nodos, pues los for utilizados se mueven en el intervalo de 0 al número de individuos.
Esto también se evidencia en los resultados donde el desempeño también obedece a la cantidad de individuos de la red social y no a la cantidad de relaciones.

Tamaño del grafo(número de nodos)	Tiempo medio (seg), con porcentaje de conocimiento de **20%**	Tiempo medio (seg), con porcentaje de conocimiento de **50%**	Tiempo medio (seg), con porcentaje de conocimiento de **80%**
100	**0.0125**	0.0105	0.011
200	0.079	**0.1015**	0.0975
250	0.1615	**0.175**	0.139
500	1.087	1.261	**1.3075**
700	3.1565	**3.334**	3.2885
1100	12.6415	**13.3615**	12.711
1200	16.565	17.1515	**19.978**

4. Si su aproximación voraz usa heurísticas, explique cómo estas permiten acercarse a la solución del problema.

El algoritmo voraz utilizado no utiliza heurísticas.

5. Compare las dos técnicas usadas (programación voraz y dinámica) con respecto a las estructuras usadas, la complejidad teórica y práctica, y los resultados obtenidos (soluciones).

El algoritmo con programación dinámica utiliza las siguientes estructuras:

- Arreglo unidimensional: su tamaño corresponde al número de individuos y cada posición vale 1 si el individuo cumple con la teoría de los seis grados. De lo contrario su valor es 0.
- Arreglo bidimensional: su dimensión es (número de individuos)^2 y corresponde a la matriz de adyacencia de los individuos.
- Arreglo bidimensional: su dimensión es (número de individuos)^2 y guarda la longitud del camino más corto entre cada par de individuos.

El algoritmo voraz usa las siguientes estructuras:

- Arreglo unidimensional: su tamaño corresponde al número de individuos y cada posición vale 1 si el individuo cumple con la teoría de los seis grados. De lo contrario su valor es 0.
- Arreglo unidimensional: su tamaño es número de individuos, mantiene la longitud del camino más corto entre un vértice origen y todos los demás. Este arreglo es actualizado con el análisis de cada vértice.
- Arreglo bidimensional: su dimensión es (número de individuos)^2 y corresponde a la matriz de adyacencia entre los vértices.

Como se ve los algoritmos se diferencian solo en una estructura, el arreglo que guarda la longitud de los caminos más cortos entre cada par de vértices, para el dinámico es bidimensional y para el voraz es unidimensional.

Con respecto a la complejidad teórica ambos algoritmos son **$\Theta(n^3)$,** esto en términos del número de instrucciones que ejecuta cada uno. Sin embargo en términos del tiempo de ejecución, ambos presentan un comportamiento cuadrático y por tanto su complejidad práctica sería **$\Theta(n^2)$**.

Con respecto a los tiempos de ejecución el algoritmo dinámico siempre tarda menos para cada entrada con porcentaje de conocimiento del 20%, 50% y 80%, por ejemplo para el caso de las entradas con 1200 individuos el algoritmo dinámico tarda 10.6955 seg. para el 20% de conocimiento, 13.158 seg. para el 50% y 11.988 seg. para el 80%. Mientras que el algoritmo voraz tarda 16.565 seg. para el 20% de conocimiento, 17.1515 seg. para el 50% y 19.978 seg. para el 80%.
Lo anterior se debe principalmente a los métodos auxiliares que utiliza el voraz en cada iteración, pues estos no presentan un comportamiento constante y más bien son O(número de individuos). Mientras que el trabajo que realiza algoritmo dinámico es constante en cada iteración.

CONCLUSIONES

Después culminar el presente informe y basándonos en los resultados obtenidos podemos obtener las siguientes conclusiones:

- El comportamiento de un algoritmo se puede evaluar desde varios puntos de vista, en este caso hacemos énfasis en el comportamiento temporal y espacial y en que tan eficiente resulta al utilizar los recursos de los cuales dispone. Dicho comportamiento, es evaluado a partir de una entrada representada por un grafo y por el porcentaje de conocimiento entre sus nodos.

- A partir de los resultados de la eficiencia temporal teórica y práctica de los algoritmos desarrollados, obtuvimos que la complejidad temporal calculada teóricamente para ambos casos no corresponde con los resultados prácticos. Tal y como lo habíamos mencionado en el análisis de los resultados, un análisis a partir de las estructuras de control (en este caso ciclos for), no necesariamente representan el comportamiento real respecto al tiempo de ejecución del algoritmo, pues a nuestro criterio la forma de implementación únicamente representa la cantidad de operaciones realizadas o el proceso requerido para generar la respuesta exacta a un problema, pero no muestra la forma general del tiempo de ejecución del algoritmo.

  Respecto a la complejidad espacial, observamos que la complejidad teórica y práctica calculada a partir del uso de recursos de la memoria heap para los dos algoritmos no corresponden. A pesar de que estas dos complejidades no pueden considerarse netamente independientes, para realizar un análisis más exacto quizá sea necesario tener en cuenta otros factores referentes a espacio o memoria. Además, de acuerdo con los resultados obtenidos, también podríamos afirmar que debido a que el tiempo de ejecución de los algoritmos es demasiado corto y las entradas no son de un tamaño considerable, quizá estos aspectos no logren afectar en gran medida el comportamiento normal de la memora heap, aunque si se observa claramente que el análisis solo puede realizarse teniendo en cuenta un límite en el tamaño de entrada, ya que es posible que se generen errores derivados del consumo excesivo de memoria, tal y como sucedió cuando se llevó a cabo la toma de datos.

- De acuerdo con los resultados obtenidos para los dos algoritmos, observamos que el comportamiento que presentan es similar hasta un mismo tamaño de entrada. Con esto deducimos que hay algoritmos que su funcionamiento lo limitan únicamente al tamaño del problema y desechan factores que a pesar de que representan características del problema o hagan de éste de un grado mayor de complejidad, no influyen de manera decisiva en el tiempo de búsqueda de la solución. En nuestro caso, lo

anterior se presenta hasta cierto punto, pues después de un tamaño de entrada los algoritmos adoptan comportamientos diferentes dependiendo de los criterios de entrada.

- Haciendo una comparación entre los dos algoritmos, observamos que el tiempo de ejecución del algoritmo desarrollado aplicando programación dinámica resulta sobresaliente respecto al que utiliza programación voraz. La diferencia se debe a detalles de implementación, por ejemplo el tipo de estructuras de datos utilizas, la definición de los métodos y el excesivo o mínimo llamado a estos, la forma de realizar los cálculos, entre otros. Teniendo en cuenta los tamaños de entrada, a pesar de que para 50% y 80% de conocimiento entre nodos del grafo ingresado, los dos algoritmos solo pueden analizar un grafo cuyo tamaño de entrada es de 1650 y 1200 respectivamente, observamos que el algoritmo voraz logra analizar para el 20% de nodos conocidos un grafo cuyo tamaño es de 2350 mientras que el algoritmo dinámico analiza únicamente un grafo con tamaño de 2250 antes de que se genere un error debido al excesivo consumo de memoria. En este caso, observamos que para esta cantidad de conocidos el algoritmo voraz puede analizar un tamaño de entrada un poco más grande.

- Al analizar los resultados obtenidos de los datos referentes al cálculo de la complejidad práctica espacial sobre la memoria heap, concluimos que es necesario tener en cuenta factores como la forma de implementación con el fin de lograr la mayor cantidad de memoria disponible para realizar los cálculos necesarios. Con esto, se destaca la importancia de utilizar las estructuras de datos adecuadas y definir bien los procesos con el fin de que la memoria no se limite. Además se observa la solución general a un problema esta representada por la complementariedad entre una solución real al problema que se haya planteado, una forma adecuada en utilización de los recursos de espacio y un tiempo lo más efectivo posible.

ASPECTOS A MEJORAR

- La forma de generar los algoritmos aleatorios y así personalizar la salida según la tipología de la red social.

- Disminuir la complejidad total tanto en espacio como en tiempo de los algoritmos que utilizan tanto programación dinámica como programación voraz.

- Mejorar la forma de implementación para lograr una aplicación más robusta, simple y eficiente.

BIBLIOGRAFÍA

Cormen, Thomas. Introduction to Algorithms. Estados Unidos, MIT Press, 2001.