**Análisis de Algoritmos**

**Grupo 50**

**Profesora**

Ana Lorena Valerio Solís

**Instituto Tecnológico de Costa Rica**

**Proyecto programado # 1**

**Análisis de complejidad algorítmica para el problema de ubicación de restaurantes de una cadena comercial**

**Integrantes**

Nelson Andrey Vega Soto

Yuliana Rojas

**Introducción:**

**Análisis del Problema:**

Ante la situación de un problema en el que una empresa requiera de algoritmos con los cuales pueda decidir en qué puntos estratégicos colocar sus sucursales para obtener un mayor beneficio, surge la necesidad de crear 4 algoritmos para los diferentes análisis que la empresa necesita hacer.

Estos algoritmos deben cubrir requisitos específicos, como lo es el que los lugares en los que colocaran los restaurantes no deban ser vecinos, así como que los algoritmos sean capaces de hacer el análisis de grafos cíclicos y acíclicos.

**Solución del Problema:**

* Para la solución del problema planteado se utilizan grafos almacenados en listas, los cuales están conformados por vértices y arcos. Que contienen el beneficio y el destino final, respectivamente.
* El primer algoritmo tiene una implementación voraz, el segundo algoritmo inicialmente se trató de hacer divide y vencerás, pero para mejorar su eficiencia se optó por la recursividad. Lo cual, incluso, permitió el análisis de grafos cíclicos con este mismo algoritmo

En el caso del algoritmo 3, también se implementa la estrategia voraz. En este caso el grafo debe ser ordenado previamente.

**Análisis de Resultados:**

**Algoritmo A**

**Medición Empírica**

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 10 | 15 | 20 | 25 | 40 | 50 | 100 | - | 200 |
| Asignaciones | 47 | 109 | 184 | 340 | 505 | 1148 | 1520 | 6166 | - | 23763 |
| Comparaciones | 53 | 156 | 271 | 554 | 858 | 2045 | 2749 | 11751 | - | 46365 |
| Cantidad de líneas ejecutadas | 100 | 265 | 455 | 894 | 1363 | 3193 | 4269 | 17917 | - | 70128 |
| Cantidad de líneas del código | 12 | | | | | | | | | |

**Factor de Crecimiento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas |
| De 6 a 10 | 1.6 | 2.25 | 2.94 | 2.56 |
| De 10 a 15 | 1.5 | 1.68 | 1.73 | 1.77 |
| De 15 a 20 | 1.3 | 1.84 | 2.04 | 1.93 |
| De 20 a 25 | 1.2 | 1.48 | 1.54 | 1.52 |
| De 25 a 40 | 1.6 | 2.13 | 2.21 | 2.18 |
| De 40 a 50 | 1.25 | 1.40 | 1.44 | 1.43 |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | Cuadrática |  |
| Clasificación del comportamiento de las **comparaciones** | Cuadrática |

|  |  |  |  |
| --- | --- | --- | --- |
| Clasificación según su entrada de los datos use la notación  Theta, O Grande, y Omega según corresponda | | | |
| **Entrada de los datos** |  |  |  |
| Clasificación |  |  |  |

**Factor de Crecimiento**

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| public void greedy(Grafo grafoM){  ArrayList<Vertice> grafo = grafoM.grafo;  int beneficioTotal = 0;  for(int i=0 ; i< grafo.size() ; i++){  if(!grafo.get(i).visitado){  grafo.get(i).visitado = true;  beneficioTotal += grafo.get(i).getBeneficio();  for(int j=0 ; j<grafo.get(i).listaArcos.size() ; j++){  grafoM.buscarVertice(grafo.get(i).listaArcos.get(j).destino).visitado = true;  }  }  }  }  public Vertice buscarVertice(String lugar){  for (int i=0; i<grafo.size() ; i++){  if(grafo.get(i).nombre == lugar)  return grafo.get(i);  }  return null;  } | 1  1  1 + 1+n + n  n  n  n  n+n+n2+n2  n (4n+3) = 4n2+3n  2n+2  n  n  1 |
| Total (la suma de todos los pasos) | 6n2 + 10n + 4 |
| Clasificación en notación O Grande | O(6n2) cuadrática |

**Nombre del algoritmo #2: máximo Beneficio B**

**Medición Empírica**

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 10 | 15 | 20 | 25 | 40 | 50 | 100 | 200 |  |
| Asignaciones | 360 | 835 | 1117 | 545 | 1042 | 2788 | 1417 | 3379 | 6495 |  |
| Comparaciones | 618 | 1502 | 2687 | 4808 | 7303 | 18600 | 24013 | 105807 | 415828 |  |
| Cantidad de líneas ejecutadas | 909 | 2222 | 3972 | 7134 | 10850 | 27738 | 35852 | 158277 | 622896 |  |
| Cantidad de líneas del código | 40 | | | | | | | | | |

**Factor de Crecimiento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas |
| De 6 a 10 | 1.67 | **2.32** | **2.43** | **2.44** |
| De 10 a 15 | 1.50 | **1.34** | **1.79** | **1.79** |
| De 15 a 20 | 1.33 | 0.49 | 1.79 | 1.80 |
| De 20 a 25 | 1.25 | 1.91 | 1.52 | 1.52 |
| De 25 a 40 |  |  |  |  |
| De 40 a 50 |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** |  |  |
| Clasificación del comportamiento de las **comparaciones** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Clasificación según su entrada de los datos use la notación  Theta, O Grande, y Omega según corresponda | | | |
| **Entrada de los datos** |  |  |  |
| Clasificación |  |  |  |

**Factor de Crecimiento**

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | líneas ejecutadas |
| public void maximoBeneficio(Grafo grafo, Vertice vertice){  if((vertice.visitado)&&(esMarcado(grafo,vertice)==false)){  if(!vertice.marcar){  benTotal += vertice.beneficio;  vertice.marcar = true;  marcar(grafo, vertice);  }  }  if(vertice.visitado){  return;  }  vertice.visitado = true;  if(maxBeneficioAux(grafo,vertice)){  benTotal += vertice.beneficio;  vertice.marcar = true;  marcar(grafo, vertice);  }  for(int i=0 ; i<vertice.listaArcos.size() ; i++){  maximoBeneficio(grafo,grafo.buscarVertice(vertice.listaArcos.get(i).destino));  }  }  public boolean maxBeneficioAux(Grafo grafo, Vertice vertice){  int cont = 0;  boolean siCon = false;  for(int i=0 ; i<vertice.listaArcos.size() ; i++){  if(!grafo.buscarVertice(vertice.listaArcos.get(i).destino).visitado){  grafo.buscarVertice(vertice.listaArcos.get(i).destino).beneficio;  }  if(grafo.buscarVertice(vertice.listaArcos.get(i).destino).marcar){  return false;  }  if(grafo.buscarVertice(vertice.listaArcos.get(i).destino).notM){  siCon = true;  }  }  if(vertice.beneficio >= cont){  return true;  }else{  if(vertice.listaArcos.size() == 1){  if(siCon)  return true;  }  return false;  }  }  public void marcar(Grafo grafo, Vertice vertice){  for(int i=0 ; i<vertice.listaArcos.size() ; i++){  grafo.buscarVertice(vertice.listaArcos.get(i).destino).notM = true;  }  }  public boolean esMarcado(Grafo grafo, Vertice vertice){  for(int i=0 ; i<vertice.listaArcos.size() ; i++){  if(grafo.buscarVertice(vertice.listaArcos.get(i).destino).marcar){  return true;  }  }  return false;  } | n\*(1+7n+4n2+2)  n  n  n  n\*(4n2+2n+5)  n  n  n  n\*(16n2+16n+10)  n  n  n\*(4n2+2n+5)  n\*(2n+2)  n2\*(4n+3)  ``````````````````  1  1  2n+2  n\*(4n+3)  n\*(4n+3)  n\*(4n+3)  n  n\*(4n+3)  n  1  1  1  1  1  1  16n2+16n+10  `````````````````  2n+2  n\*(4n+3)  4n2+2n+5  ```````````````````  2n+2  n\*(4n+3)  n  n  7n+4n2+2 |
| Total (la suma de todos los pasos) | 28n3+34n2+33n |
| Clasificación en notación O Grande | O(28n3) |

**Nombre del algoritmo #3**

**Medición Empírica**

| Operaciones | Cantidad de datos de entrada por defecto | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 10 | 15 | 20 | 25 | 40 | 50 |
| Asignaciones | 47 | 109 | 185 | 340 | 505 | 1080 | 1520 |
| Comparaciones | 53 | 156 | 282 | 554 | 858 | 1903 | 2749 |
| Cantidad de líneas ejecutadas | 100 | 265 | 467 | 894 | 1363 | 2983 | 4269 |
| Cantidad de líneas del código | | 27 | | | | | |

**Factor de Crecimiento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas |
| De 6 a 10 | 1.67 | 2.32 | 2.94 | 2.65 |
| De 10 a 15 | 1.50 | **1.70** | **1.81** | **1.76** |
| De 15 a 20 | 1.33 | 1.84 | 1.96 | 1.91 |
| De 20 a 25 | 1.25 | 1.49 | 1.55 | 1.52 |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** | Cuadrática |  |
| Clasificación del comportamiento de las **comparaciones** | Cuadrática |

|  |  |  |  |
| --- | --- | --- | --- |
| Clasificación según su entrada de los datos use la notación  Theta, O Grande, y Omega según corresponda | | | |
| **Entrada de los datos** |  |  |  |
| Clasificación |  |  |  |

**Factor de Crecimiento**

|  |  |
| --- | --- |
| Código fuente | Líneas ejecutadas |
| public void cantMaxRest(Grafo grafoM) {  int cant = 0;  ArrayList<Vertice> grafo = grafoM.grafo;  for(int i=0 ; i<grafo.size() ; i++){  if(!grafo.get(i).visitado){  grafo.get(i).visitado = true;  cant++;  for(int j=0 ; j < grafo.get(i).listaArcos.size() ; j++){  grafoM.buscarVertice(grafo.get(i).listaArcos.get(j).destino).visitado = true;  }}}}  public void ordenarGrafo(){  ArrayList<Vertice> grafoOrdenado = new ArrayList<>();  for(int j = 0; j<grafo.size();j++){  if(grafoOrdenado.isEmpty()){  grafoOrdenado.add(grafo.get(j));  }  else{  for(int i = 0; i<grafoOrdenado.size(); i++){  if(grafo.get(j).listaArcos.size()<= grafoOrdenado.get(i).listaArcos.size()){  grafoOrdenado.add(i, grafo.get(j));  }  else{  grafoOrdenado.add(grafo.get(j));  }  break;  }  }  }  this.grafo = grafoOrdenado;  } | 1  1  2n+2  n  n.  n  n (2n+2)  n2 \* (4n+3)  1  2n+2  n  n  2n+2  n  n  n  n  1 |
| Total (la suma de todos los pasos) | 4n3+5n2+17n+10 |
| Clasificación en notación O Grande | O(4n3) |

**Nombre del algoritmo #4 (Algoritmo 2 con datos cíclicos)**

**Medición Empírica**

| Operaciones | Cantidad de datos de entrada por defecto | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 6 | 10 | 15 | 20 | 25 | - |
| Asignaciones | 286 | 267 | 543 | 202 | 2026 | - |
| Comparaciones | 857 | 1803 | 3196 | 5445 | 7705 | - |
| Cantidad de líneas ejecutadas | 1244 | 2653 | 4706 | 8071 | 11446 | - |

**Factor de Crecimiento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Factor talla | Factor Asig | Factor Comp | Factor Cantidad de líneas ejecutadas |
| De 6 a 10 | 1.67 | 0.93 | 2.10 | 2.13 |
| De 10 a 15 | 1.50 | **2.03** | **1.77** | **1.77** |
| De 15 a 20 | 1.33 | 0.37 | 1.70 | 1.72 |
| De 20 a 25 | 1.25 | 10.03 | 1.42 | 1.42 |
| De 15 a 25 | 1.67 | 3.73 | 2.41 | 2.43 |
| De 10 a 20 | 2 | 0.76 | 3.02 | 3.04 |

|  |  |  |
| --- | --- | --- |
| Clasificación del comportamiento de las **asignaciones** |  |  |
| Clasificación del comportamiento de las **comparaciones** |  |

**NOTA: la medición analítica se omite, ya que es la misma del algoritmo B**

**Ya que este algoritmo fue más eficiente que la implementación del algoritmo D.**

**Conclusiones:**

* Según la medición realizada indique cuál de los algoritmos es más eficiente analítica y empíricamente; tanto en comparaciones y asignaciones y según la entrada de los datos.

**Recomendaciones:**

* Aspectos o ámbitos de interés para ampliar el alcance del proyecto.
* Áreas o campos de estudio que pueden complementar las temáticas analizadas.
* Ideas que sean de utilidad para interesados en el área en que se desenvuelve el proyecto.

**Literatura citada**

* Design and Analysis of Algorithms, Massachusetts Institute of Technology

Profs. Erik Demaine, Srini Devadas, and Nancy Lynch

Algoritmos Genéticos. (s.f.). Recuperado 28 septiembre, 2019, de http://www.nce.ufrj.br/GINAPE/VIDA/alggenet.htm

Díaz, Javier F, D. J. F. (2012, 5 marzo). Algoritmo de cifrado simétrico AES. Recuperado 28 septiembre, 2019, de http://sedici.unlp.edu.ar/handle/10915/4210

José Leandro González González, J. L. G. G. (2013, 11 junio). Propuesta de algoritmo de clasificación genética | González González | Revista Cubana de Ingeniería. Recuperado 28 septiembre, 2019, de http://rci.cujae.edu.cu/index.php/rci/article/view/158

Pérez Pérez, Isaías, P. P. I. (2010, 1 enero). Análisis del desempeño de los estudiantes en el diseño y construcción de algoritmos secuenciales, selectivos e iterativos. Recuperado 28 septiembre, 2019, de https://www.uaeh.edu.mx/investigacion/producto.php?producto=4791