**Instituto Tecnológico de Costa Rica**

**Análisis de Algoritmos**

**Grupo 50**

**Profesora**

Ana Lorena Valerio Solís

**Proyecto programado # 2**

**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, cite las referencias usadas.**

**Solución del problema**

* Hacer un diagrama con las estructuras utilizadas, listas, pilas, arreglos u otras estructuras, por estrategia.
* Realice un diagrama de flujo o seudocódigo donde explique la lógica que desarrollo, para aplicar cada una de las estrategias
* Describa cuál es la estrategia seleccionada en programación dinámica para avanzar entre las fases.
* Describa cuáles son los tipos de cruce realizados.
* Describa cuál es el tipo de mutación que se aplicó.

**Análisis de Resultados:**

|  |  |  |
| --- | --- | --- |
| **Algoritmo** | **Estado** | **Comentario** |
| Determinístico | Finalizado y completo |  |
| Genético | Finalizado y completo |  |

**Algoritmo #1: Dinámico**

**Medición Empírica**

| Operaciones | Cantidad de datos de entrada por defecto | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 20 | 30 | 60 | 120 | 1200 |
| Asignaciones | 454 | 1540 | 3234 | 12104 | 96863 | 31541789 |
| Comparaciones | 707 | 2661 | 5816 | 22696 | 139627 | 35850716 |
| Memoria |  |  |  |  |  |  |

**Factor de Crecimiento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Factor talla | Factor Asig | Factor Comp | Factor Mem |
| 20 a10 | 2 | 3,392 | 3,763 |  |
| 30 a 20 | 1,5 | 2,1 | 2,185 |  |
| 60 a 30 | 2 | 3,742 | 3,902 |  |
| 120 a 60 | 2 | 8,002 | 6,152 |  |
| 1200 a 120 | 10 | 325,6329 | 256,7906 |  |

|  |  |  |
| --- | --- | --- |
| 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 |  |  |  |

**Medición Analítica**

|  |  |
| --- | --- |
| 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 lineaEnsamblaje() {  buscarPosibles();  for (int e = 0; e < ensamblaje.getEstaciones().size(); e++) {  Estacion estacionT = ensamblaje.getEstaciones().get(e);  int tiempo = ensamblaje.getTiempoCiclo();  int i = 0;  while (i < listaPosiblesTareas.size()) {  if (tiempo >= listaPosiblesTareas.get(i).getTiempo()) {  estacionT.setTarea(listaPosiblesTareas.get(i).getNombre());  tiempo -= listaPosiblesTareas.get(i).getTiempo();  listaPosiblesTareas.get(i).setIsEstacion(true);  listaPosiblesTareas.get(i).setEstacion(estacionT.getNombre());  listaPosiblesTareas.remove(i);  buscarPosibles();  i = 0;  } else {  i++;  }  }  estacionT.setTiempoSobrante(tiempo);  }  }  public boolean **precedentesUsados**(Tarea tarea) {  for (int i = 0; i < tarea.getPrecedentes().size(); i++) {  if (!ensamblaje.getTarea(tarea.getPrecedentes().get(i)).isIsEstacion()) {  return false;  }  }  return true;  }    public void buscarPosibles() {  for (int i = 0; i < listaTareas.size(); i++) {  if ((precedentesUsados(listaTareas.get(i)))  && (!listaTareas.get(i).isIsEstacion())  && (!listaPosiblesTareas.contains(listaTareas.get(i)))) {  asig++;  listaPosiblesTareas.add(listaTareas.get(i));  }  }  listaPosiblesTareas.sort(Comparator.comparing(Tarea::getTiempo));  Collections.reverse(listaPosiblesTareas);  } | 4n2 + 8n + 4  2n + 2  N  N  N  N \* (n+2)  N\*n  N\*n  N\*n  N\*n  N\*n  N\*n  N\*n  N\*n \* [4n2 + 8n + 4] = 4n4+8n3+4n2  N  N  17n+14n2+8n3+4n4 + 6  2n + 2  N  N  4n + 2  2n+2  N \* [ 4n + 2] = 4n2 + 2n  2N  N  1  1  4n2 + 8n + 4 |
| Total (la suma de todos los pasos) | 4n4 +8n3+ 14n2 +17n +6 |
| Clasificación en notación O Grande | O(n4) |

**Algoritmo #2: Genético**

**Medición Empírica**

| Operaciones | Cantidad de datos de entrada por defecto | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 20 | 30 | 60 | 120 | 1200 |
| Asignaciones | 3460 | 9622 | 18540 | 114226 | 377692 | 113386023 |
| Comparaciones | 4489 | 14986 | 31279 | 216542 | 750716 | 228154108 |
| Memoria |  |  |  |  |  |  |

**Factor de Crecimiento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Factor talla | Factor Asig | Factor Comp | Factor mem |
| 20 a10 | 2 | 2,7809 | 3,3383 |
| 30 a 20 | 1,5 | 1,9268 | 2,0872 |
| 60 a 30 | 2 | 6,161 | 0,069 |
| 120 a 60 | 2 | 3,3065 | 3,47 |
| 1200 a 120 | 10 | 300,207 | 303,915 |

|  |  |  |
| --- | --- | --- |
| 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 |  |  |  |

**Medición Analítica**

|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código del método de resolver el rompecabezas. | líneas ejecutadas |
| public void lineaEnsamblaje() {  buscarPosibles();  int estacionesS = listaEstaciones.size();  for (Estacion estacion : listaEstaciones) {  int ciclo = ensamblaje.getTiempoCiclo();  if (estacionesS == listaTareas.size() + 1) {  int x = (int) ((Math.random()) \* (listaPosiblesTareas.size()));  estacion.getTareas().add(listaPosiblesTareas.get(x).getNombre());  estacion.setTiempoSobrante(ciclo - listaPosiblesTareas.get(x).getTiempo());  listaPosiblesTareas.get(x).setIsEstacion(true);  listaTareas.remove(listaPosiblesTareas.get(x));  listaPosiblesTareas.remove(x);  buscarPosibles();  } else {  int i = 0;  while (i < listaPosiblesTareas.size()) {  int x = (int) ((Math.random()) \* (listaPosiblesTareas.size()));  if (ciclo >= listaPosiblesTareas.get(x).getTiempo()) {  if (estacionesS == listaTareas.size() + 1) {  break;  }  estacion.setTarea(listaPosiblesTareas.get(x).getNombre());  ciclo -= listaPosiblesTareas.get(x).getTiempo();  listaPosiblesTareas.get(x).setIsEstacion(true);  listaPosiblesTareas.get(x).setEstacion(estacion.getNombre());  listaTareas.remove(listaPosiblesTareas.get(x));  listaPosiblesTareas.remove(x);  buscarPosibles();  i = 0;  } else {  asig++;  }  }  estacion.setTiempoSobrante(ciclo);  }  estacionesS--;  }  }  public boolean **precedentesUsados**(Tarea tarea) {  for (int i = 0; i < tarea.getPrecedentes().size(); i++) {  if (!ensamblaje.getTarea(tarea.getPrecedentes().get(i)).isIsEstacion()) {  return false;  }  }  return true;  }  public **void buscarPosibles**() {  for (int i = 0; i < listaTareas.size(); i++) {  if ((precedentesUsados(listaTareas.get(i)))  && (!listaTareas.get(i).isIsEstacion())  && (!listaPosiblesTareas.contains(listaTareas.get(i)))) {  listaPosiblesTareas.add(listaTareas.get(i));  }  }  }  public void **crearEstaciones**(Ensamblaje ensamblaje) {  int estac = (int) (ensamblaje.getLineaEnsamblaje().size() \* 0.7);  for (int i = 0; i < estac; i++) {  listaEstaciones.add(new Estacion("Estacion" + String.valueOf(i + 1), ensamblaje.getTiempoCiclo()));  }  }  public void **crearPadres**() {  limpiarListas();  for (int i = 0; i < 5; i++) {  limpiarListas();  lineaEnsamblaje();  Padre padre = new Padre(listaEstaciones, "Padre " + (i + 1));  listaPadres.add(padre);  }  crearHijos();  }  public void **limpiarListas**() {  listaTareas.clear();  for (Tarea x : ensamblaje.getLineaEnsamblaje()) {  x.setIsEstacion(false);  }  for (Tarea x : ensamblaje.getLineaEnsamblaje()) {  listaTareas.add(x);  }  listaEstaciones.clear();  listaPosiblesTareas.clear();  crearEstaciones(ensamblaje);  }  public void **crearHijos**() {  ArrayList<Padre> padres = fitnessPadre(this.listaPadres);  this.listaPadres = padres;  Hijo hijo1 = crearHijo(padres.get(0), padres.get(1));  Hijo hijo2 = crearHijo(padres.get(1), padres.get(2));  Hijo hijo3 = crearHijo(padres.get(0), padres.get(2));  hijo1.setNombre("Hijo1");  hijo2.setNombre("Hijo2");  hijo3.setNombre("Hijo3");  this.listaHijos.add(hijo1);  this.listaHijos.add(hijo2);  this.listaHijos.add(hijo3);  Hijo mejorHijo = fitnessHijo(listaHijos);  mutacion(mejorHijo);  }  public Hijo **crearHijo**(Padre padre1n, Padre padre2n) {  ArrayList<Estacion> esta1 = new ArrayList<>();  ArrayList<Estacion> esta2 = new ArrayList<>();  for (Estacion estacion : padre1n.getPadre()) {  ArrayList<String> tareas = new ArrayList<>();  tareas.addAll(estacion.getTareas());  Estacion nueva = new Estacion(estacion.getNombre(), estacion.getTiempoSobrante());  nueva.setTareas(tareas);  esta1.add(nueva);  }  for (Estacion estacion : padre2n.getPadre()) {  ArrayList<String> tareas = new ArrayList<>();  tareas.addAll(estacion.getTareas());  Estacion nueva = new Estacion(estacion.getNombre(), estacion.getTiempoSobrante());  nueva.setTareas(tareas);  esta2.add(nueva);  }  Padre padre1 = new Padre(esta1, padre1n.getNombre());  Padre padre2 = new Padre(esta2, padre2n.getNombre());  ArrayList<String> noUsadosTemp = new ArrayList<>();  ArrayList<String> usados = new ArrayList<>();  ArrayList<Estacion> estacionesH = new ArrayList<>();  for (Estacion estacion : padre1.getPadre()) {  Estacion estacionN = new Estacion(estacion.getNombre(), ensamblaje.getTiempoCiclo());  estacionesH.add(estacionN);  }  Hijo hijo = new Hijo("", padre1.getNombre(), padre2.getNombre(), estacionesH);  for (int i = 0; i < hijo.getHijo().size(); i++) {  for (String tarea : noUsadosTemp) {  if  (hijo.getHijo().get(i).getTiempoSobrante() >= ensamblaje.getTarea(tarea).getTiempo()) {  hijo.getHijo().get(i).getTareas().add(tarea);  hijo.getHijo().get(i).setTiempoSobrante(hijo.getHijo().get(i).getTiempoSobrante() - ensamblaje.getTarea(tarea).getTiempo());  usados.add(tarea);  }  }  for (String tarea : usados) {  if (noUsadosTemp.contains(tarea)) {  noUsadosTemp.remove(tarea);  }  }  for (int t = 0; t < padre1.getPadre().get(i).getTareas().size(); t++) {  if (usados.contains(padre1.getPadre().get(i).getTareas().get(t))) {  int tiempoSobrante = padre1.getPadre().get(i).getTiempoSobrante() + ensamblaje.getTarea(padre1.getPadre().get(i).getTareas().get(t)).getTiempo();  padre1.getPadre().get(i).setTiempoSobrante(tiempoSobrante);  padre1.getPadre().get(i).getTareas().remove(padre1.getPadre().get(i).getTareas().get(t));  }  }  for (int t = 0; t < padre2.getPadre().get(i).getTareas().size(); t++) {  if (usados.contains(padre2.getPadre().get(i).getTareas().get(t))) {  int tiempoSobrante = padre2.getPadre().get(i).getTiempoSobrante() + ensamblaje.getTarea(padre2.getPadre().get(i).getTareas().get(t)).getTiempo();  padre2.getPadre().get(i).setTiempoSobrante(tiempoSobrante);  padre2.getPadre().get(i).getTareas().remove(padre2.getPadre().get(i).getTareas().get(t));  }  }  if (padre1.getPadre().get(i).getTiempoSobrante() >= padre2.getPadre().get(i).getTiempoSobrante()) {  for (String tarea : padre1.getPadre().get(i).getTareas()) {  if (!usados.contains(ensamblaje.getTarea(tarea))) {  hijo.getHijo().get(i).getTareas().add(tarea);  hijo.getHijo().get(i).setTiempoSobrante(hijo.getHijo().get(i).getTiempoSobrante() - ensamblaje.getTarea(tarea).getTiempo());  usados.add(tarea);  }  }  for (String tarea : padre2.getPadre().get(i).getTareas()) {  if (!usados.contains(ensamblaje.getTarea(tarea))) {  noUsadosTemp.add(tarea);  }  }  } else {  for (String tarea : padre2.getPadre().get(i).getTareas()) {  if (!usados.contains(ensamblaje.getTarea(tarea))) {  hijo.getHijo().get(i).getTareas().add(tarea);  hijo.getHijo().get(i).setTiempoSobrante(hijo.getHijo().get(i).getTiempoSobrante() - ensamblaje.getTarea(tarea).getTiempo());  usados.add(tarea);  }  }  for (String tarea : padre1.getPadre().get(i).getTareas()) {  if (!usados.contains(ensamblaje.getTarea(tarea))) {    noUsadosTemp.add(tarea);  }  }  }  }  return hijo;  }  public void **mutacion**(Hijo hijo) {  listaTareas.clear();  listaPosiblesTareas.clear();  for (Tarea x : ensamblaje.getLineaEnsamblaje()) {  listaTareas.add(x);  }  ArrayList<Estacion> estacionesH = new ArrayList<>();  for (Estacion estacion : hijo.getHijo()) {  Estacion estacionN = new Estacion(estacion.getNombre(), ensamblaje.getTiempoCiclo());  estacionesH.add(estacionN);  }  buscarPosiblesMutantes();  Hijo hijoMutado = new Hijo(hijo.getNombre(), hijo.getNombreP1(), hijo.getNombreP2(), estacionesH);  ArrayList<String> usados = new ArrayList<>();  for (Estacion estacionM : hijoMutado.getHijo()) {  for (Estacion estacionN : hijo.getHijo()) {  for (String tarea : estacionN.getTareas()) {  if ((!usados.contains(tarea))  && (estacionM.getTiempoSobrante() >= ensamblaje.getTarea(tarea).getTiempo())  && (listaPosiblesTareas.contains(ensamblaje.getTarea(tarea)))) {  usados.add(tarea);  estacionM.getTareas().add(tarea);  estacionM.setTiempoSobrante(estacionM.getTiempoSobrante() - ensamblaje.getTarea(tarea).getTiempo());  ensamblaje.getTarea(tarea).setIsMutado(true);  buscarPosiblesMutantes();  }  }  }  }  this.prodigio = hijoMutado;  }  public ArrayList<Padre**> fitnessPadre**(ArrayList<Padre> padres) {  for (Padre padre : padres) {  padre.sobranteT();  }  padres.sort(Comparator.comparing(Padre::getSobrante));  ArrayList<Padre> nuevosP = new ArrayList<>();  nuevosP.add(padres.get(0));  nuevosP.add(padres.get(1));  nuevosP.add(padres.get(2));  return nuevosP;  }  public Hijo **fitnessHijo**(ArrayList<Hijo> hijos) {  for (Hijo hijo : hijos) {  hijo.sobranteT();  }  hijos.sort(Comparator.comparing(Hijo::getSobrante));  return hijos.get(0);  }  public boolean **precedentesUsadosM**(Tarea tarea) {  for (int i = 0; i < tarea.getPrecedentes().size(); i++) {  if (!ensamblaje.getTarea(tarea.getPrecedentes().get(i)).isIsMutado()) {  return false;  }  }  return true;  }  public void **buscarPosiblesMutantes**() {  for (int i = 0; i < listaTareas.size(); i++) {  if ((precedentesUsadosM(listaTareas.get(i)))  && (!listaTareas.get(i).isIsMutado())  && (!listaPosiblesTareas.contains(listaTareas.get(i)))) {  listaPosiblesTareas.add(listaTareas.get(i));  }  }  } | 2n2+7n+2  1  2n+2  n  n  n  n  n  n  n  n  n  2n3+7n2+2n  n  n2 + 2n  n \* n  n\*n  n\*n  n\*n  n\*n  n\*n  n\*n  n\*n  n\*n  2n4+7n3+2n2  n \* n  n\*n  n  n  2n4+2n3+23n2  +25n+5  -----------------------------  2n+n  n  2n+2  2n+2  n (2n+2)  3n  2n2+7n+2  ----------------------------  1  2n+2  3n  5n+3  11n+10  2n+2  11n2+10n  2n5+2n4+23n3  +25n2+5n)  1  1  16n4+24n3  +112n2  +135n+86  2n5+18n4+47n3  148n2+163n+100  1  2n + 2  n  2n+2  n  1  1  5n+3  **11n+10**  3n+7  1  4n3+36n2+37n+16  4n3+36n2+37n+16  4n3+36n2+37n+16  1  1  1  1  1  1  3n+3  4n4+10n3+4n2  +15n+21  16n4+24n3  +112n2  +135n+86  1  1  2n+2  n  n  n  n  n  2n+2  n  n  n  n  n  1  1  1  1  1  2n+2  n  n  1  2n+2  2n2+2n)  n2  n2  n2  n2  2n2+2n  N2  N2  2n2+2n  n2  n2  n2  n2  2n3+2n  n2  n2  n2  n2  n  n  2n2+2n  n2  n2  n2  n2  2n2+2n  n2  n2  2n2+2n  n2  n2  n2  n2  2n3+2n)  n2  n2  4n3+36n2+37n+16  1  1  2n+2  n  1  2n+2  n  n  4n+2  1  1  2n+2  (2n2+2n)  (2n3+2n2)  N3  N3  N3  N3  N3  N3  (4n4+2n3)  1  4n4+10n3+4n2  +15n+14  2n+2  n  1  1  1  1  1  3n+7  2n+2  n  1  3n+3  2n+2  n  3n+2  2n+2  n  n  4n+2 |
| Total (la suma de todos los pasos) | 2n5+18n4+47n3  148n2+163n+100 |
| Clasificación en notación O Grande |  |

**Graficas**

**Estado final**

**Conclusiones:**

* Según la medición realizada indique cuál estrategia de diseño es más eficiente, justifique su aseveración.
* Responda la siguiente pregunta ¿Conforme crece la talla cual algoritmo se va haciendo más eficiente?

**Recomendaciones:**

**Literatura citada**