Instituto Tecnológico de Costa Rica
Unidad de Computación

Análisis de complejidad algorítmica

Kevin Walsh Muñoz
Jonathan Rojas Vargas

Sede San Carlos
18/11/2014

Tabla de contenidos

Introducción

Este proyecto se efectuará con el propósito de evaluar el rendimiento de dos algoritmos diferentes solucionando un mismo problema. En este caso el tienen que ser capaces de buscar la ruta más corta que hay entre el primer y el último vértice de un grafo.

Este tipo de conocimiento es muy valioso actualmente en el mundo de la programación debido a que existe una gran necesidad de contar con los algoritmos más eficientes y así gozar de todas sus ventajas como por ejemplo:

- Reducción de tiempos.
- Adecuado aprovechamiento de recursos.
- Disminución de costos.

Por estos motivos es necesario que siempre se tenga la capacidad de poseer los mejores códigos y mediante las mediciones analíticas y empericas son buenas formas de identificar buenos algoritmos.

Análisis del problema

Para realizar de forma exitosa este proyecto será necesario la creación de dos algoritmos que sean capaces de encontrar la ruta más corta o un aproximado entre un vértice inicial y uno destino del grafo, a cada uno de estos códigos se les deberá calcular su complejidad temporal, además del número de comparaciones, asignaciones y líneas de código ejecutadas cuando el programa finaliza con una cantidad dada de vértices, también se tiene que llevar el control de la cantidad de líneas que poseen dichos algoritmos.

Pero todos estos algoritmos necesitaran de un grafo sobre el cual puedan realizar sus cálculos. Por este motivo se requiere la implementación de un grafo que cumpla con las siguientes restricciones:

- Cada vértice contará con conexiones a todos los vértices (Fuertemente conexo). .
- Cada arco contará con su propia distancia y serán dirigidos.

Todos los algoritmos anteriormente mencionados deberán ser implementados en el lenguaje de programación C# con sus respectivas mediciones en un documento de texto.

Solución del problema

Para solucionar de forma exitosa los problemas anteriormente citados se empleará C# como lenguaje de programación.

Como primer objetivo se creará el grafo el cual contará con una cantidad número de vértices indicado por el usuario. El nombre y la distancia entre cada vértice se agregará de forma automática. Cuando el grafo esté finalizado será necesario interpretarlo y representarlo en una matriz para que pueda ser manipulado por el algoritmo genético.

El primer algoritmo que busque la ruta más eficiente debido a que es más sencilla la manipulación de los datos usando una matriz con las distancias entre los vértices será necesario la creación de la misma. Luego para crear el algoritmo genético se deberán seguir varios pasos:

- Generar las rutas iniciales.
- Crear una función para analizar las rutas.
- Analizar las rutas iniciales
- Mediante un ciclo crear las generaciones
- Luego seleccionar los individuos para realizar el cruce.
- Cruzar los que tengan similitudes.

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generacion */
        FOR Tamaño poblacion/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generacion,
            para el cruce (probabilidad de seleccion proporcional
            a la funcion de evaluacion del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la funcion de evaluacion de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generacion.
        END
        IF  la poblacion ha convergido THEN
            Terminado := TRUE
    END
END
```

Ilustración 1 Seudocódigo algoritmo genético

Para la realización del método probabilístico se trabajó directamente con el grafo, este fue recorrido de un punto origen y un punto destino con el fin de obtener un aproximado a la ruta más corta. Este iniciaba en el punto origen, luego tomando una probabilidad entre el rango de peso escogemos un arco del vértice origen y así sucesivamente hasta llegar al punto destino. Tomando en cuenta que son varios intentos para obtener la ruta más optimizada, ya que se van comparando entre las rutas generadas.

Cuando se tengan todos los algoritmos implementados se continuará realizando los correspondientes cálculos de comparaciones, asignaciones y tiempo de ejecución a cada código.

Análisis de resultados

Probabilístico

`public void algoritmoProbabilistico(Vertice origen, Vertice destino, int cantHormigas)`	
`{`	
`Vertice temO = buscarV(origen.numero);`	1
`Vertice temD = buscarV(destino.numero);`	1
`if ((temO == null) || (temD == null))`	1
`{`	
`return;`	
`}`	
`ruta.AddFirst(temO);`	
`for (int i = 0; i < cantHormigas; i++)`	n+1
`{`	
`while (true)`	n*n
`{`	
`arcosVertice.Clear();`	1
`verificarProbabilidad(temO);`	
`lock (syncLock)`	
`{`	
`numRandom = r.Next(0, arcosVertice.Count);`	1
`}`	
`Arco temA = arcosVertice.ElementAt<Arco>(numRandom);`	1
`Vertice temV = temA.destino;`	1
`if ((temA.visitado == false))`	1
`{`	
`ruta.AddLast(temV);`	1
`distanciaTotal += temA.distancia;`	1
`temO = temV;`	1
`temA.visitado = true;`	1
`}`	
`if (temV.numero == destino.numero)`	1
`{`	
`break;`	
`}`	
`}`	
`verificarDistancia(ruta, distanciaTotal);`	
`ruta.Clear();`	
`distanciaTotal = 0;`	
`limpiar();`	
`}`	
`}`	

```
public void verificarDistancia(LinkedList<Vertice> ruta, int
distancia)
{
       int seguVert = 1;                                          1
       int i = 0;                                                 1
       int j = 0;                                                 1
       string rutaTEXT = "";                                      1

        if (distancia < distanciaRuta)                            1
        {
             Vertice temV =                                       1
buscarV(ruta.ElementAt<Vertice>(0).numero);
        rutaTEXT += temV.numero.ToString() + ", ";

        while (i < temV.listArco.Count)                           1
        {
              Arco temA = temV.listArco.ElementAt<Arco>(i);       n+1
              if (seguVert >= ruta.Count)                         n+1
              {
                   break;
              }
           else if (ruta.ElementAt<Vertice>(seguVert).numero ==   n+1
temA.destino.numero)
              {
                   temA.feromona += 5;                            n+1
                   seguVert++;                                    n+1
                   temV = temA.destino;                           n+1
                  rutaTEXT += temA.destino.numero.ToString() +    n+1
", ";
                  i = 0;                                          n+1
              }
              else
              {
                   i++;                                           n+1
              }
              }                                                   n+1
                     rutaOptimaText = rutaTEXT;                   n+1
                     ruta.Clear();                                n+1
                     distanciaRuta = distancia;
       }
}
```

static void limpiar()
{
for (int i = 0; i < grafo.Count; i++)
{
Vertice temV = grafo.ElementAt<Vertice>(i);
 	
for (int j = 0; j < temV.listArco.Count; j++)
{
 	
temV.listArco.ElementAt<Arco>(j).visitado = false;
}
}
}
 	
public void verificarProbabilidad(Vertice temV)
{
for (int i = 0; i < temV.listArco.Count; i++)
{
Arco temA = temV.listArco.ElementAt<Arco>(i);
int numProbabilidad = (rango2 % temA.distancia);
for (int j = 0; j < numProbabilidad; j++)
{
arcosVertice.AddFirst(temA);
}
}
}
	
	n+1
	
	n+1
	
	n^2
	
	n^2
	
	n+1
	
	n+1
	n+1
	n^2+1
	
	n^2+1

Total	$4n^2$+17n+24
Clasificación en notación O	Cuadrática

Genético

```
public void algoritmoGenetico(int origen, int destino, int
cantPoblacionInicial, int totalVertices, int generaciones)
{
       LinkedList<int[]> primeraGeneracion = new LinkedList<int[]>();          1
       LinkedList<int[]> poblacionEvaluada = new LinkedList<int[]>();          1
       LinkedList<int[]> nuevaGeneracion = new LinkedList<int[]>();            1
       for (int i = 0; i < cantPoblacionInicial; i++)                          n+1
       {
             primeraGeneracion.AddFirst(generarGenRandom(origen,               n+1
destino,  totalVertices - 2));
       }
       foreach (var item in primeraGeneracion)                                 n+1
       {
              for (int i = 0; i < item.Length; i++)                            n²+1
              {
                  Console.Write(item[i] + " ");                                n²+1
              }
              Console.WriteLine();
       }
       primeraGeneracion = evaluarPoblacion(primeraGeneracion);                1
       for (int i = 0; i < generaciones; i++)                                  n+1
       {
              cruzar(primeraGeneracion)                                        n+1
       }
}


LinkedList<int[]> SelecionarCruzar(LinkedList<int[]> poblacion)
{
       LinkedList<int[]> parejas = new LinkedList<int[]>();                    1
       int[] ruta1 = new int[(poblacion.ElementAt<int[]>(0).Length)];          1
       int[] ruta2 = new int[(poblacion.ElementAt<int[]>(0).Length)];          1
       parejas = seleccionarPareja(poblacion);                                 1
       for (int i = 0; i < parejas.Count - 1; i += 2)                          n+1
       {
              ruta1 = parejas.ElementAt<int[]>(i);                             n+1
              ruta2 = parejas.ElementAt<int[]>(i + 1);                         n+1
              cruzar(ruta1, ruta2)
       }
       return parejas;
       seleccionarPareja
}
```

```
LinkedList<int[]> cruzar(int[] ruta1, int[] ruta2)
{
        int[] hijo1;                                                    1
        int[] hijo2;                                                    1
        LinkedList<int> rutaa = new LinkedList<int>();                  1
        LinkedList<int> rutab = new LinkedList<int>();                  1
        LinkedList<int[]> hijos = new LinkedList<int[]>();              1
        int posicion1 = 0;                                              1
        int posicion2 = 0;                                              1
        for (int i = 0; i < ruta1.Length - 2; i++)                      n+1
        {
              rutaa.AddLast(ruta1[i]);                                  n+1
              for (int b = 0; b < ruta2.Length - 2; b++)                n^2+1
              {
                    rutab.AddLast(ruta2[b]);                            n^2+1
                    if (ruta1[i] == ruta2[b])                           n^2+1
                    {
                        posicion1 = i;                                  n^2+1
                        posicion2 = b;                                  n^2+1
                        b = ruta2.Length - 1;                           n^2+1
                        i = ruta1.Length - 1;                           n^2+1
                    }
                }
            }
            for (int i = posicion1; i < ruta1.Length - 2; i++)          n+1
            {
                rutab.AddLast(ruta1[i]);                                n+1
            }
            for (int i = posicion2; i < ruta1.Length - 2; i++)          n+1
            {
                rutaa.AddLast(ruta2[i]);                                n+1
            }
            hijo1 = new int[rutaa.Count - 1];                           1
            hijo2 = new int[rutab.Count - 1];                           1

            for (int i = 0; i < rutaa.Count - 1; i++)                   n+1
            {
                //hijo1[i] = rutaa.ElementAt<int>(0)[i];                n+1
            }
            return hijos;
        }
}
```

```
LinkedList<int[]> seleccionarPareja(LinkedList<int[]> poblacion)
{
      LinkedList<int[]> parejasFinal = new LinkedList<int[]>();          1
      int[] ruta1 = new int[(poblacion.ElementAt<int[]>(0).Length)];     1
      int[] ruta2 = new int[(poblacion.ElementAt<int[]>(0).Length)];     1
      int posicion;                                                      1

      for (int i = 0; i < poblacion.Count - 1; i++)                      n+1
      {
            for (int b = 0; b < poblacion.ElementAt<int[]>(i).Length     n²+1
- 2; b++)
            {
      posicion =
selecionarParejaAux(poblacion.ElementAt<int[]>(i)[b], i, poblacion);     n²+1
                  if (posicion != -1)
                  {
                      ruta1 = poblacion.ElementAt<int[]>(i);           n²+1
                      ruta2 = poblacion.ElementAt<int[]>(posicion);    n²+1
                      parejasFinal.AddFirst(ruta1);                    n²+1
                      parejasFinal.AddFirst(ruta2);                    n²+1

poblacion.Remove(poblacion.ElementAt<int[]>(posicion));                n²+1

poblacion.Remove(poblacion.ElementAt<int[]>(i));                       n²+1
                      i = -1;
                      b = poblacion.ElementAt<int[]>(i).Length - 2;
                  }
               }
         }
         return parejasFinal;
       }


int selecionarParejaAux(int numero, int posicion, LinkedList<int[]>
poblacion)
{
      for (int i = posicion + 1; i < poblacion.Count - 1; i++)           n+1
      {
            for (int b = 0; b < poblacion.ElementAt<int[]>(i).Length     n²+1
- 2; b++)
            {
                 if (poblacion.ElementAt<int[]>(i)[b] == numero)         n²+1
                 {
                     return i;
                 }
            }
       }
       return -1;
}
```

`int[] agregarPeso(int[] ruta)`
`{`
`  int peso = 0;`
`  for (int i = 0; i < ruta.Length - 2; i++)`
`  {`
`  peso += matrizValores[ruta[i], ruta[i + 1]];`
`  }`
`  ruta[ruta.Length - 1] = peso;`
`  return ruta;`
`  }`
 	
`  LinkedList<int[]> evaluarPoblacion(LinkedList<int[]> poblacion)`
`  {`
`  int[] tempValores = new int[(poblacion.ElementAt<int[]>(0).Length)];`
`  int[] valor1 = new int[(poblacion.ElementAt<int[]>(0).Length)];`
`  int[] valor2 = new int[(poblacion.ElementAt<int[]>(0).Length)];`
`  LinkedList<int[]> poblacionEvaluada = new LinkedList<int[]>();`
`  foreach (var item in poblacion)`
`  {`
`  poblacionEvaluada.AddFirst(agregarPeso(item));`
`  }`
 	
`  for (int i = 0; i < poblacionEvaluada.Count; i++)`
`  {`
`  valor1 = poblacionEvaluada.ElementAt<int[]>(i);`
`  for (int b = i; b < poblacionEvaluada.Count; b++)`
`  {`
`  valor2 = poblacionEvaluada.ElementAt<int[]>(b);`
`  if (valor1[valor1.Length - 1] > valor2[valor2.Length - 1])`
`  {`
`  tempValores = valor2;`
 	
`  poblacionEvaluada.Remove(poblacionEvaluada.ElementAt<int[]>(b));`
`  poblacionEvaluada.AddFirst(tempValores);`
`  i = -1;`
`  break;`
`  }`
`  }`
`  }`
 	
`  return poblacionEvaluada;`
`}`
	
	1
	n+1
	
	n
	
	1
	
	1
	1
	1
	1
	1
	
	n+1
	
	n
	
	n+1
	n
	
	n^2+1
	
	n^2
	n^2+1
	
	n^2
	
	n^2

Total	$24n^2+25n+51$
Clasificacion en notación O	Cuadrática

Conclusiones

Estas estrategias de programación pueden ser muy importantes para resolver problemas, que realizados de la manera tradicional son casi imposibles de resolver a diferencia de estos.

Por este motivo es importante saber y tratar con estos tipos de algoritmos ya que en un futuro pueden ser muy útiles.

Con respecto al proyecto no pudimos concluir, sin embargo logramos conocer y comprender estos algoritmos, ya que son bastantes complejos.

Recomendaciones

Actualmente en el mercado de aplicaciones de software es de gran importancia que dichas aplicaciones se ejecuten de la manera más rápida posible por este motivo en algunos casos se pueden utilizar algoritmos genéticos o probabilísticos para generar soluciones eficientes y muy acertadas.

Referencias Bibliográficas

A, G. G. (22 de 10 de 2014). *Universidad de Oviedo.* Obtenido de http://gio.uniovi.es/documentos/nacionales/ArtNac87.pdf

Álvarez, J. L. (22 de 10 de 2014). *Ingenieros en Apuros.* Obtenido de Introducción a los Algortimos Geneticos: http://ingenierosenapuros.files.wordpress.com/2012/06/introduccic3b3n-a-losalgoritmos-genc3a9ticos-p.pdf

Coello, C. C. (23 de 10 de 2014). Obtenido de http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&ved=0CEYQFjAG&url=http%3A%2F%2Fdelta.cs.cinvestav.mx%2F~ccoello%2Frevistas%2Fgenetico.pdf.gz&ei=jBpIVML4Dc2QgwSl_IHYBg&usg=AFQjCNGFkrb4pJHRZqIPnWRcAbmIHO_guQ&bvm=bv.77880786,d.eXY&cad=rja