Análisis de Algoritmos **Luz Gasca Soto & Jorge L. García F**

Algoritmo de Dijsktra, RMC

Mayo 30, 2020

El Problema de la Ruta Más Corta, RMC.

RMC: Sea $G=(V,A)$ una digráfica con pesos en las aristas, n vértices, y m aristas, G también es considerada una **red.** El problema **RMC** consiste en encontrar la Ruta Más Corta (longitud) o más barata (costos) entre dos o más vértices.

La **Longitud** o **Costo** de una ruta es la suma de las longitudes o costos de los arcos que la constituyen.

El Problema RMC es básico para los problemas de Optimización Combinatoria y Flujo en Redes. Algoritmos que resuelven estos problemas lo utilizan como una operación más que los complementa. Este tipo de problemas tiene aplicaciones en modelos de distribución y asignación de recursos, entre otras aplicaciones de la Teoría de Redes. Su aplicación más importante sucede cuando se combinan problemas de flujo máximo en redes a costo mínimo, esto es el problema de Flujo Máximo a Costo Mínimo en una red.

Algoritmos para la Ruta Más Corta

Los algoritmos para RMC se pueden clasificar, según su construcción, básicamente en dos tipos:

* **Algoritmos que construyen Arboles de RMC**
  los cuales encuentran la RMC del origen s a cada uno de los otros vértices.
  (RMC entre dos vértices específicos)
* **Algoritmos auxiliados por Matrices**
  que localizan la RMC entre cada par de vértices.
  (RMC entre todo par de vértices)

Los algoritmos de RMC entre dos vértices específicos basan su solución analítica en las Ecuaciones de Bellman. La multiplicación matricial constituye una herramienta fundamental para los algoritmos que resuelven la RMC entre todo par de vértices.

Notación

En el presente trabajo denotaremos por:

$G = (N, A, \delta)$ a una red con costos en los arcos, donde
N es el conjunto, no vacío y finito, de nodos o vértices, $n = |N|$;
A es el conjunto de arcos o aristas dirigidas, $A \subseteq N \times N$, $m = |A|$; y
$\delta : A \rightarrow R^+$ es la función de costo sobre los arcos.

El arco de i a j se representa como (i , j).
La longitud o costo del arco (i, j) está dada por $a(i,j)$ o por a_{ij}.
Los nodos origen y destino son s y d, respectivamente.
$d(v)$ representa la etiqueta del vértice v.
$P(vw)$ representa una ruta o trayectoria del vértice v a w; para toda $v, w \in N$.
RMC(s, w) representa la ruta más corta entre los vértices s y w.
$A^*(s)$ es una Arborescencia de G para la cual cada ruta $P(vw)$ es única y representa la **RMC**(s, x), para toda $x \in N$.

Análisis de Algoritmos **Luz Gasca Soto & Jorge L. García F**

Algoritmo de Dijsktra, RMC Mayo 30, 2020

Algoritmo de Dijsktra, RMC

El Algoritmo de Dijkstra es uno de los más eficientes para resolver el Problema de RMC, de un nodo fuente s a cada uno de los otros nodos en una Red G para la cual las longitudes de los arcos a_{ij} son valores no negativos. Es considerado el algoritmo básico de etiquetamiento.
En este escrito daremos una panorámica general del algoritmo.

Panorámica General del Algoritmo

- Cada nodo o vértice i posee una etiqueta $d(i)$ donde:
  $d(i)$ es **permanente** si representa la RMC de s a i.
  $d(i)$ es **temporal** si aún no representa la RMC de s a i;
  es decir, está por encima de la mínima longitud posible.
- El algoritmo inicia etiquetando los vértices:
  El origen s , será permanente con $d(s) = 0$
  Los demás tendrán etiqueta temporal, dada por:
  $d(j) = a_{sj}$ si $\exists\ (s, j) \in A;\ \ j \in N \setminus \{s\}$ // costo del arco (s,j) si éste existe.
  $d(j) = \infty$ si $\nexists\ (s, j) \in A.$ // costo infinito.
- Busca entre los nodos con etiqueta temporal al nodo k cuya etiqueta, $d(k)$, resulta ser la mínima.
  Si $d(k) \neq \infty$ Entonces
  Tal $d(k)$ se convierte en permanente.
  Se revisan las etiquetas temporales de los nodos j adyacentes al vértice k,
  actualizando de la siguiente forma sus etiquetas:
  $d(j) \leftarrow min\ \{ d(j), d(k) + a_{kj} \}\ \forall\ j$ adyacente a k
- El proceso termina cuando todos los nodos han sido etiquetados de manera permanente o cuando todas las etiquetas $d(k)$ tienen como valor mínimo a infinito.

El algoritmo queda determinado, en forma general, de la siguiente manera:

Algoritmo Dijkstra

Objetivo: Determinar una arborescencia de RMC en una red $G=(V, A, \delta)$ conexa no trivial, con pesos en las aristas. Se tiene que $|V|= n$, $|A|= m$.

I Definición inicial de etiquetas.

II Mientras existan Etiquetas Temporales Finitas

1. Selección del nodo, k, con etiqueta temporal mínima
2. Si $d(k)$ es finita
   a) $d(k)$ se convierte en etiqueta permanente
   b) Actualización de las etiquetas temporales de los nodos adyacentes a k.

Análisis de Algoritmos **Luz Gasca Soto & Jorge L. García F**

Algoritmo de Dijsktra, RMC

Mayo 30, 2020

Colas de Prioridades

Una Cola de Prioridades es un TDA con muchísimas aplicaciones. Su nombre proviene de una aplicación en sistemas operativos: el mantenimiento de una cola de procesos los cuales serán ejecutados según su prioridad; en realidad éstas son usadas diariamente en varias actividades cotidianas, tales como La sala de espera de un hospital, donde cada paciente será atendido según la gravedad de su caso.

Recordemos que en el algoritmo de Dijkstra, nos interesa recuperar la etiqueta temporal mínima, pues requerimos manipular un conjunto mediante operaciones que faciliten cosas como, agregar un elemento, encontrar y borrar el mínimo elemento.

Especificaciones del TDA Cola de Prioridades,PQ

Una cola de prioridades de tipo ***T*** con prioridades reales, es un TDA cuyos objetos son los conjuntos ***Q*** de objetos en ***T.*** Cada uno de los elementos en ***Q*** tiene asociado un número real ***k***, denominado llave del objeto.
Las **operaciones válidas** sobre un conjunto ***Q*** son las siguientes:

Inserta(*x*:objeto, *k* :llave) [***Insert***]:
// Incluye un nuevo objeto ***x*** , con llave ***k*** a la colección de objetos ***Q.***
EncuentraMin: objeto [***FindMin***]:
// Encuentra y regresa el objeto cuya llave es mínima en la colección ***Q.***
BorraMin(*x*:objeto) [***DeleteMin***]:
// Borra un objeto *x*, cuya llave es mínima en la colección de objetos ***Q.***

Inicia [*Inicialize*] // Crea una nueva estructura de tipo Cola de Prioridades.
Vacio[*Empty*] // Indica si la colección de objetos ***Q*** es o no vacía.
Elimina(x :objeto) [***Delete***] // Elimina un objeto **x** de la colección de objetos ***Q.***
MinQ // Apuntador al mínimo elemento en la cola de prioridades
Funde (***Q1, Q2***: PQ)[***Meld***] // Une dos colas de prioridades en una

DecrementaLlave(***x*** :objeto, ***k*** :Llave) [***DecreaseKey***]:
// Toma al objeto ***x*** de ***Q*** y cambia el valor de su llave a ***k***, sólo si ***k*** es menor que la llave anterior.

TDA's en el Algoritmo de Dijkstra

Al intentar implementar un algoritmo, debemos decidir la mejor representación de los datos y surge, entonces, el balance de factores no todos alcanzables simultáneamente. Analicemos un poco al algoritmo de Dijkstra, sus ejemplares y demás estructuras auxiliares, necesarias para llevar a cabo su ejecución. Un ejemplar para el problema de la RMC, es una red con costos positivos en los arcos.

TDA para la RED *G*

Representaremos a la red ***G*** como una Lista de Adyacencias, las operaciones básicas requeridas para identificar el Tipo de Datos Abstracto RED, son:

CreaRed : RED // Genera un objeto de tipo RED.
AccesaAdyacentes (v: Vértice) // Regresa la lista de los vértices adyacentes a v.
GuardaArco (***a***: Arco; ***c:*** TipoCosto) // Añade el arco a con costo c en la red.
PrimerVertice : Vértice // Regresa el primer vertice de la red.
PrimerArco : Arco // Regresa el primer arco en la red.
VerticeFinal (*a*: Arco): Vértice // Regresa el vertice final del arco a.
ImprimeRed // Muestra los datos contenidos en la red.

Análisis de Algoritmos **Luz Gasca Soto & Jorge L. García F**

Algoritmo de Dijsktra, RMC

Mayo 30, 2020

TDA para las ETIQUETAS TEMPORALES

Las etiquetas temporales resultan ser una estructura auxiliar de almacenamiento fundamental en el desarrollo del algoritmo de Dijkstra, por ello la forma cómo se representen será crucial para el cálculo del desempeño computacional. Las etiquetas temporales tienen operaciones específicas que dan prioridad a las etiquetas con el mínimo valor. Entonces, podemos englobarlas en el TDA definido anteriormente como Colas de Prioridades, al cual denominaremos PQ.

TDA para la RUTA

Durante la ejecución del algoritmo de Dijkstra, cada vértice de la red pasa de la situación de no-revisado a ya-revisado; el valor de su etiqueta puede ir disminuyendo. En el cálculo de rutas mínimas el predecesor de un vértice en la arborescencia puede cambiar. Por otra parte, queremos tener acceso directo a la etiqueta del vértice en la cola de prioridades, pues teniendo esta información se puede 'reconstruir' la RMC de un vértice a otro, o, bien, se puede 'reconstruir' la arborescencia de RMC. Por esto, para cada vértice en la red ***G*** declaramos un Estado que almacene tal información, las operaciones básicas para el **TDA ESTADO** son:

Visit : Lógico. // Característica que indica si el vértice ha sido o no visitado.
Padre : vértice. // Característica que indica cuál es el vértice predecesor en la Arborescencia.
Dist : TCosto. // Característica que indica cuál es el valor de la RMC encontrada hasta el momento.
// Dado un vértice ***v*** indica cuál es la Distancia encontrada de la ruta ***P(sv)***.
CreaEstado : ESTADO. // Crea un objeto de tipo ESTADO.
ActualizaVisit (***vi***: Lógico). // Modifica la característica visit con el valor de vi
NvoPadre (***nP*** : vértice). // Cambia el predecesor en la Arborescencia de RMC.
NvaD (**nvaD**: TCosto). // Modifica la Distancia del vértice.
MuestraEdo // Despliega los valores de Visit, Padre y Dist.
AsignaStatus (***vist***: Lógico, ***pre***: vértice, ***nvaD***: T-Costo).
// Actualiza los valores de Visit, Padre y Dist.

Ahora bien, a todos los Estados, uno por vértice, los organizamos en una estructura de tamaño ***n***, que denominaremos RUTA. Finalmente, las operaciones básicas para el **TDA RUTA** son:

CreaRuta : RUTA. // Crea un objeto de tipo RUTA.
Status (***v***: Vértice): Estado. // Muestra el estado actual del vértice ***v.***
ModificaEdo (***v***: Vértice; **visita**: Lógico; ***pa***: Vértice; ***dv***: TipoCosto).
// Cambia los valores del estado para el vértice v en la RUTA.
Distancia (***v***: Vértice): TipoCosto // Dado un vértice v indica cual es su distancia actual.
CambiaDist (***v***: Vértice; ***dv***: TipoCosto). // Modifica la distancia del vértice dado.
SuPadre (***v***: Vértice) : Vértice. // Dado un vértice ***v***, indica cuál vértice es su padre (predecesor).
CambiaPadre (***v, nvoPa***: Vértice) // Dado un vértice, modifica su predecesor.
FueVisitado (***v***: Vértice): Lógico // Indica si el vértice ya ha sido revisado.
CambiaVisitado (***v***: Vértice; ***nvoV*** : Lógico)
// Modifica la situación de visitado, por ***nvoV***, para el vértice dado.
ActivaEnQ (*v*: Vértice): Apuntador.
//Dado un vértice v, mantiene un apuntador a su etiqueta temporal en la cola de prioridades.
DesActivaEnQ (v: Vértice). // Desactiva el apuntador del vértice ***v*** en la cola de prioridades.
ImprimeRuta // Muestra los datos almacenados en RUTA.
ImprimeRMC (***vs, vt***: Vértice). // Despliega la RMC entre los vértices ***vs*** y ***vt.***

Algoritmo de Dijsktra, RMC Mayo 30, 2020

Algoritmo de Dijkstra

A continuación describimos el algoritmo de Dijkstra aplicando las operaciones de los TDAs definidos anteriormente y utilizando la notación de POO. Los parámetros requeridos por el algoritmo son:

- ***G*** red para la cual se buscará la arborescencia de RMC;
- ***s*** nodo raíz de la arborescencia;
- ***n*** número de vértices en la red;
- ***P*** ruta: estructura donde se almacenaran los estados para cada vértice.

```
Dijkstra (G: Red, s: Vertice, n : Entero, P: Ruta)
Variables
Q : PQ;          // Cola de Prioridades
v : Vertice;     // vértice a revisar
e : Arco;        // arco a revisar
nvaD : TipoCosto;  // Nueva Distancia encontrada
Inicio
   P.IniciaRuta;     //  Da valores iniciales el vector de Estados
   P.ModificaEdo(s, −1, 0);   // Valor inicial para s
  Q.Create;        //  Crea Cola de Prioridades
  Q.Inserta( s , P.Distancia( s ));     //  Añade la etiqueta de s en Q
  Mientras  (¬ Q.vacio )  Hacer
        v ← Q.BorraMin;       //  Borra de Q al elemento cuya llave sea la mínima
        Para cada arco e Adyacente a v  Hacer    //  Revisa los vertices Adyacentes a v
            e ←  v .ObtenArco
            w ←  e. VerticeFinal
            nvaD ←  P.Distancia( v ) + e.Costo
            Si  (¬ P.Visitado( w ))   Entonces  // Si no ha sido visitado
               P.ModificaEdo(w, v, nvaD);    // modifica los datos de w  en P
               Q.Inserta(w, nvaD);              //  inserta w en Q
            EnOtroCaso                          //  Si ya fue visitado y ..
              Si  ( nvaD < P.Distancia( w ))  Entonces  //  la distancia encontrada es menor
                 P.ModificaEdo(w, v, nvaD);       // modifica los datos de w en P y
                 Q.DecrementaLlave(w, nvaD); { decrementa la llave de w en Q }
              Fin_Si // nvaD < ...
           Fin_Si  //¬ P.Visitado ...
          v.SiguienteArco
       Fin_Para cada arco e ...
  Fin_Mientras //¬ Q.vacio
Fin  //Dijkstra
```

Cuídate y Quedate en casa, es tu lugar seguro.

Saludos, Lucy Gasca

Análisis de Algoritmos **Luz Gasca Soto & Jorge L. García F**

Algoritmo de Dijsktra, RMC Mayo 30, 2020

Complejidad del Algoritmo

La operación ***P***.IniciaRuta requiere en total ***O(n),*** ya hay ***n*** vértices
El primer ciclo principal (**Mientras** (¬ Q.vacio)) depende del tamaño de **Q** y en el peor de los casos **Q** tendrá a todos los vértices a la vez; es decir, el ciclo iterará a lo más ***n*** veces. La operaciones pesadas en el ciclo son las que involucran a la cola de prioridades. Revisemos tales operaciones.
En total se realizan: ***n*** operaciones Inserta y ***n*** operaciones BorraMin, en un caso extremo se realizan ***m*** operaciones DecrementaLlave.
Si utilizamos un apuntador al vértice en la cola de prioridades, llegar (localizar) al vértice tendrá costo constante, sino… tendríamos que buscarlo y eso podría ser muy costoso, incluso ***O(n)***. Así que supondremos que tenemos tal apuntador.
Supondremos que estamos representando a la gráfica con una lista de adyacencias, eso hará que pedir los vecinos sea tiempo constante, ya que sólo es el acceso al apuntador de la lista. Recordemos que una lista de adyacencias tiene tamaño ***O(n+m)*** y en cambio una matriz de adyacencias es de $O(n^2)$.
Si utilizamos una lista simple como cola de prioridades insertar podría ser de ***O(1)***, pero borrar el mínimo podría ser lineal, así que tal implementación nos costaría: $O(n^2)$.
Si usamos para la cola de prioridades un **Heap Binario**, las operaciones Inserta, BorraMin y DecrementaLlave cuestan ***O(log n),*** entonces como requerimos
n operaciones ***Insert*** : ***O(n log n)***
n operaciones ***DeleteMin***: ***O(n log n)*** y
m operaciones ***Dkey:*** ***O(m log n)***
Así que la suma de todo esto es: ***O(m log n)***

Por lo tanto, el desempeño computacional del Algoritmo de Dijkstra, para determinar una arborescencia de Rutas más Cortas, es en el peor de los casos ***O(m log n)*** utilizando Heap Binarios para manipular las etiquetas de distancias y usando Listas de Adyacencias para representar la gráfica.

Nota: El algoritmo de Dijkstra es muy similar al de Prim, solo que Dijkstra va acumulando las distancias encontradas.

Cuídate
y
Quedate en casa, es tu lugar seguro.

Saludos, Lucy Gasca