Proyeto de Matematica Discreta II - DINIC

Granier, Pierre-Yves - flammeus8@gmail.com
Leberle, Maico C. - maico.leberle@gmail.com
Drazile, Eduardo - edudraz@gmail.com
Schmidt, Mariano - nanoschmidt22@gmail.com

May 27, 2014

1 Introducción

En el laboratorio se pretende implementar el algoritmo de DINIC para obtener el flujo maximal, (y su respectivo corte minimal) en un Network dado.

2 Decisiones de Diseño

2.1 Estructuras de Datos

Para la implementación usamos las siguientes estructuras de datos:

1. Vertice
2. Lista
3. Lado
4. DovahkiinSt

2.2 Descripción del uso de las estructuras para implementar el Newtork

En primera instancia veamos los vertices del Network. Cada uno de estos vertices tiene una lista de vecinos forward, y una lista de vecinos backward. Ademas contiene una lista de vertices a los que va a poder llegar en la "Iteración actual" (Según cumpla o no las condiciones de Dinic para poder ser agregado en la corrida del DFS)

Los lados son dos vértices, los cuales tienen tambien capacidad, y flujo.

Las listas enlazadas no solo tienen puntero al primer elemento, sino que también último elemento de la lista. Las listas son con elementos *void, lo que significa que las podemos usar para alojar distintos tipos de estructuras. (Usa una estructura auxuliar member, que son los nodos de la lista. Tambien cada member (nodo) de la lista, tiene un puntero al nodo anteorior, digamos que son listas, doblemente enlazadas.

Por último, la estructura DovahkiinSt, es la estructura principal la cual esta descripta en el enunciado, y mostramos exactamente acá como esta definida.

```
struct DovahkiinSt {
VerticeP fuente, resumidero;
/* En flujo se guarda el valor del flujo hasta el momento. */
u64 flujo;

/* data es la lista de vértices, junto con toda su información asociada.
temp se utiliza para establecer los niveles de BFS en ActualizarDistancias. */
list_t data, temp;

/* iteracion es la manera que tuvimos de evitar tener que actualizar las distancias
de todos los vértices en cada ActualizarDistancias: sólo se acepta como válida a
la distancia v->distancia, de un VerticeP v, si v->iteracion es igual al valor
actual de la variable iteracion que se define aquí. Si v->iteracion es distinto
(a saber, menor), entonces la distancia BFS de v no está determinada (con lo cual
implementamos la idea de que v no puede formar parte de un camino aumentante
de la fuente al resumidero. */

unsigned int iteracion;
/* flujo_maximal == true sii ya se ha hallado un flujo maximal. */
bool flujo_maximal;

}
```

2.3 Descripcipón del uso dado a las estructuras para modelar el problema

Diremos que un network, esta compuesto por una Lista de Vértices, donde cada vértice tiene una lista de vecinos forward y una lista de vecinos backward.

El tad vértice tiene las funciones

`add_neighboor_to_list() y buscar_vecinos_posibles()`

las cuales sirven para realizar las busqueda BFS, en

`actualizar_distancias()`

Esas dos funciones son de las que se vale esta última funcion para actualizar las iteraciones a los que pertenecen los vértices (su distancia en una determinada iteración)

Para realizar las busquedas DFS dentro de Dinic es que se utilizan la funciones de BusquedaCaminoAumentante

Explicaremos mas en detalle este algoritmo que es el mas complicado de la implementación: Empiezo en la fuente, y despues sigo esta idea: si el vértice en el que estoy parado tiene mas vecinos posibles que agregar, avanzo en el network (llamo a la funcion "avanzar()"), que agrega a sus vecinos posibles. En caso de esta manera no llegar al resumidero, vuelvo a través de los ancestros (la función avanzar, marca los ancestros de cada vértice que se va agregando) a través de la función retroceder. Si llego al resumidero, no me queda mas que volver por los ancestros para encontrar el camino y aumentar el flujo.

2.4 Otras decisiones

Usamos listas doblemente enlazadas ya que nos permiten manejar de mejor manera el hecho de cantidad de vertices, lados, y nos permiten ser mas eficientes a la hora de trabajar con listas como la de "vecinos_posibles" que son de tamaño dinámico.

Por otro lado, en las listas, tuvimos que implementar algunas funciones especiales, que destruyen el nodo de la listas, pero no su contenido. Esto lo que nos permitia era usar las estructuras ya allocadas de vértices y lados, y cuando tengamos que borrar esa lista, que las estructuras de vértices y lados no se dañen.

3 División de Tareas

Granier, Pierre-Yves: Implementación de Algunas Estrucuras de Datos. Colaboración en la implementación de Actualizar Distancias y Búsqueda de Camino Aumentante.

Leberle, Maico C.: Implementación de Búsqueda de Camino Aumentante y Actualizar Distancias, Debuggeo y Control de Memory Leaks

Drazile, Eduardo: Funciones de AumentarFlujo, ImprimirFlujo, Cortes y etc(las últimas pedidas) y colaboración con las estructuras de datos. Además de construir el Main.

Schmidt, Mariano: Implementación de Parseo de la Entrada, colaboración en la funcion de Búsqueda de caminos aumentantes y Actualizar Distancias.