

Auxiliar tarea computacional

*Profesor: Fernando Ordóñez**Auxiliar: Renaud Chicoisne*

1 Estructuras de datos

Se le recomienda ocupar la siguiente estructura de datos para guardar la información contenida dentro de los archivos `nodos.csv` y `arcos.csv`.

Primero, se define una clase/estructura para los nodos:

```
nodo
{
    int id;                //Indice del nodo: primera columna de nodos.csv
    int b_i;               //Demanda o oferta del nodo: segunda columna de nodos.csv
    vector<int> arcos_salientes; //Arreglo de los indices de los arcos (i,k)
}
```

Secundo, se define una clase/estructura para los arcos:

```
arco
{
    int id;    //Indice del arco: primera columna de arcos.csv
    int i;     //Indice del nodo cola: segunda columna de arcos.csv
    int j;     //Indice del nodo cabeza: tercera columna de arcos.csv
    int u_ij;  //Capacidad del arco: cuarta columna de arcos.csv
    int c_ij;  //Costo del arco: quinta columna de arcos.csv
}
```

Y finalmente, guardamos todo en una unica super estructura:

```
datos
{
    int n;                //Numero de nodos
    int m;                //Numero de arcos
    vector<nodo> nodos;    //Arreglo de los nodos
    vector<arco> arcos;    //Arreglo de los arcos
}
```

Por supuesto se tendrá que agregar varios atributos necesarios durante la ejecución de los algoritmos.

2 Dijkstra

Se puede notar que los largos de los arcos son todos positivos, por lo tanto el método lo más eficiente para calcular las rutas mínimas es al algoritmo de Dijkstra.

Para implementar Dijkstra, si el lenguaje de programación le permite se le recomienda ocupar la clase de tipo `priority_queue` que implementa un *heap* (Java, MatLab, C++). Esta estructura de datos se reordena automáticamente durante la inserción o la supresión de un elemento y permite conocer de manera inmediata el elemento con menor label (la raíz del heap). Los estándares pueden variar según los lenguajes de programación así que verifican bien según que criterio están ordenados los heaps, para no sacar el elemento con etiqueta más grande a cada iteración por ejemplo :).

3 Flujo máximo

Después de la lectura de los datos, tendrán que agregar dos nodos s y t al grafo, con los arcos necesarios para unirlos a la red ya existente.

También hay que pensar en como implementar el grafo residual. Una de las maneras de hacerlo es la siguiente: para cada arco $(i, j) \in E$ se agrega al grafo un arco (j, i) que tiene capacidad x_{ij} , un flujo virtual pasando por el de $u_{ij} - x_{ij}$. Hay que tener cuidado cuando se necesita sacar el flujo óptimo y el corte mínimo, hay que considerar SOLO los arcos originales y no los que se agregaron después.

Durante la búsqueda de un camino aumentante, se puede mostrar que si se elige el camino aumentante que contiene el menor número de arcos, el algoritmo tendrá un comportamiento mejor (ie: más rápido).

4 Corte mínimo

Calcular el corte mínimo desde el flujo óptimo que encontraron les permitiera de comprobar si el flujo encontrado es realmente óptimo. Para encontrar uno, se puede encontrar los nodos alcanzables desde s vía arcos del grafo residual, o bien los nodos que pueden alcanzar t vía los arcos del grafo residual.

5 Flujo a costo mínimo

Para encontrar el flujo a costo mínimo, si ocupan un algoritmo de tipo *shortest augmenting path*, se hacen los mismo comentarios que en la parte Flujo máximo, agregando a cada arco (j, i) del grafo residual un costo $-t_{ij}$. Se puede comprobar que el flujo encontrado es óptimo con las condiciones de holgura complementaria.

6 Checker

Se creó un verificador de resultados para los tres problemas. Con este documento vienen dos archivos `checker_linux` y `checker_mac` que permiten verificar la factibilidad de sus resultados (bajo el formato impuesto en el enunciado de la tarea). Para verificar sus resultados, tienen que poner el ejecutable en la misma carpeta que los archivos de datos `nodos.csv` y `arcos.csv` y el archivo de resultados que quieren verificar.

Al marcar `./checker_linux` aparecerán las opciones disponibles:

```
./checker_linux -M1 verifica el archivo min_path.csv
./checker_linux -F1 verifica los archivos max_flow.csv y min_cut.csv
./checker_linux -C1 verifica el archivo min_cost_flow.csv
```

(Si se ocupa un Mac, hay que reemplazar `checker_linux` por `checker_mac...`)

Cada una de las rutinas de verificación cuenta si hay factibilidad o no de la solución, y si no lo es, por que motivo. Por ejemplo, la opción `-F1` verifica el balanceo de flujo para cada arco, y si el conjunto de arcos entregado es realmente un corte. Si hay factibilidad, entregará el valor del flujo y la capacidad del corte.