Tarea 4 de Diseño y análisis de algoritmos

Leandro Rabindranath León

Fecha de entrega: hasta el martes 6 de diciembre de 2016

1. Introducción

El fin de este laboratorio es aprehender los principios básicos de optimización combinatoria mediante las redes de flujo. Un fin "colateral" es que los Profesores de Investigación de Operaciones, así como los estudiantes, tengan la posibilidad de aprehender y comprender el vínculo esencial entre los algoritmos y la Investigación de Operaciones.

Para desarrollar exitosamente este laboratorio requieres tener instalado:

1. La biblioteca $\aleph_\omega$ (Aleph-ω); especialmente debes estar familiarizado con el tipo `Net_Cost_Graph` definido en `tpl_netcost.H` y usar una versión igual o mayor a la 1.6-d.

2. La biblioteca TCLAP

   `http://tclap.sourceforge.net/`

   Esta biblioteca se encuentra en la mayoría de los repositorios de las principales distribiciones. No es necesaria su comprensión, pero sí la requerirías para poder compilar `rand-gen` y `solver`

La instalación y uso de estos programas ya se te debe haber presentado y explicado anteriormente.

1.1. Asignación de cupos a estudiantes para cursos intensivos

En una cierta Universidad de un cierto país se ha alcanzado un estado interno de descomposición cultural entre sus miembros (Profesores y Estudiantes) que ha deparado en una severa intermitencia académica. Salvando una porción minoritaria, muchos estudiantes y profesores de la Universidad en cuestión se han acostumbrado a impartir y cursar un semestre por año impedidos por disturbios promovidos por estudiantes. Como resultado, la duración promedio de una carrera de 5 años se ha duplicado a 10 años.

Un fenómeno que se ha observado es un incremento substancial en la demanda de los cursos intensivos. Un curso intensivo es un curso especial, muy acelerado, que

se dicta durante las vacaciones de Agosto. Los cursos intensivos son pagos, no son gratuitos. Los profesores que dictan el intensivo sacrifican sus vacaciones en aras de una compensación económica adicional a su salario regular.

El gobierno y otras ONG ven con cierta preocupación los cursos intensivos debido a las siguientes observaciones:

1. El promedio de aprobación es substancialmente mayor que el de un semestre regular, lo cual a primera vista parece extraño porque se dispone de menos tiempo y recursos. Hay materias muy laboriosas y que requieren tiempo de entrenamiento para dominarlas que se llegan a dictar en intensivos. Por ejemplo, las materias de programación.

2. Prácticamente cualquier Profesor puede dictar una materia en el intensivo. Se han observado Profesores que dictan una materia sin tener experiencia consumada.

3. Durante la realización de los cursos intensivos no hay interrupciones, lo cual también resulta algo extraño.

4. Los estudiantes deben pagar el curso intensivo.

Ante ese panorama general de deterioro, el gobierno, quien costea enteramente el presupuesto de la Universidad, luego de años de mostrar una alta preocupación, está considerando seriamente que la educación devenga privada. El razonamiento de quiénes apoyan esta medida es la idea que al costear los estudiantes su educación éstos colaborarían y exigirían que la Universidad operase regularmente, sin interrupciones. El principal sostén de este razonamiento es la sospecha o creencia de que la ausencia de interrupciones durante el intensivo se debe a que éste es pago.

Así las cosas, el gobierno, con el propósito de sondear la Universidad y obtener información que le permita decidir si la privatiza o no, ha decidido apoyar la realización plena de los cursos intensivos bajo las siguientes premisas:

- La Universidad es libre de proponer el curso, el cupo del curso, el Profesor y los honorarios del Profesor.

- Para asegurar la calidad de los cursos, el gobierno es enteramente responsable de la evaluación. La Universidad no diseñará las evaluaciones. De este modo el gobierno podrá cotejar con estadísticas de cursos regulares y sondear si el intensivo es o no tan riguroso como un curso regular.

- Supeditado a las condiciones curriculares, cada estudiante puede seleccionar tantas materias como él quiera para cursar en el intensivo. Esta selección es una propuesta.

- Con base a la selección de cada estudiante, el gobierno decidirá, según criterios que trataremos más adelante, las materias que él cursará.

- Para garantizar el cumplimiento de la ley, específicamente la gratuidad de la Universidad, el gobierno asumirá enteramente los costes de los cursos intensivos.

Así las cosas, el problema principal del gobierno es doble:

1. Calcular la máxima asignación de cursos, de manera de beneficiar a la mayor cantidad posible de estudiantes, y con ello mitigar el estado de miseria causado por los disturbios.

2. Minimizar el coste de subvención que asume el gobierno.

En otros términos, el gobierno desea asignar cursos a la mayor cantidad posible de estudiantes que opten por curso intensivo y al menor coste posible.

1.2. Coste esperado de manutención por estudiante

El gobierno cuenta con un estudio de aprendizaje automático (machine learning) sobre datos estudiantiles de los últimos 20 años. Dicho estudio arroja tres grandes agrupaciones o grupos (clusters) de tipos de estudiantes:

Tipo A: Estudiantes de alto rendimiento.

Tipo B: Estudiantes regulares de rendimiento normal.

Tipo C: Estudiantes de bajo rendimiento.

Por cada tipo de estudiante, el modelo maneja una variable aleatoria, llamada A, que define la cantidad de materias aprobadas en el intensivo. Por ejemplo, la probabilidad de aprobar dos (2) materias dado que el estudiante inscribió tres (3) podría definirse como:

$$P(A = 2|\text{tiene tres materias inscritas}) = P(A = 2|3) = 0{,}1$$

Por cada tipo de estudiante, se define una matriz de probabilidades de aprobación. Un ejemplo de tal matriz se ilustra en el cuadro 1.

Nº materias inscritas	P($A = 0$)	$P(A = 1)$	$P(A = 2)$	$P(A = 3)$	Coste esperado
1	0.4	0.6	0	0	400
2	0.5	0.3	0.2	0	1300
3	0.6	0.2	0.1	0.1	2300

Cuadro 1: Un ejemplo de probabilidades de aprobación y costes esperados para estudiantes de un grupo hipotético. Los costes están calculados para un coste por estudiante de 1000

El gobierno estima un coste constante por cada estudiante, independientemente del grupo al que pertenezca, que incluye sus gastos de manutención. Nota que este coste no es el mismo coste de los honorarios por el curso (que también financiará el gobierno). El coste de manutención es asociado por cada materia inscrita. Se considera que una materia aprobada es una perdida completa de ese coste. Una pretensión del gobierno es minimizar esta pérdida.

Si c es el coste de manutención de un estudiante y éste inscribe n materias, el coste esperado es la esperanza de lo que el gobierno habría de pagar por el estudiante.

Por ejemplo, con base a $c = 1000$ y la matriz del cuadro 1, el coste esperado por un estudiante que inscribiese tres (3) materias sería:

$$\underbrace{3 \times 1000 \times \mathrm{P}(A = 0|3)}_{\text{Aprueba 0 materias}} + \underbrace{2 \times 1000 \times \mathrm{P}(A = 1|3)}_{\text{Aprueba 1 materia}} + \underbrace{1 \times 1000 \times \mathrm{P}(A = 2|3)}_{\text{Aprueba 2 materias}} + \underbrace{0 \times 1000 \times \mathrm{P}(A = 3|3)}_{\text{Aprueba 3 materias}}$$

Los costes esperados para la matriz del cuadro 1 están puestos en la última columna del cuadro 1.

El gobierno proporciona como parámetros las matrices de probabilidades de aprobación para los tres tipos de grupos de estudiantes.

Es esencial que entiendas perfectamente el concepto de coste esperado de manutención. Presta mucha atención porque su cálculo, aunque simple, es truculento de programar.

Nota que requieres manejar tres matrices, una por cada grupo.

Una de las entradas al solucionador (ver § 1.8) serán las tres matrices en el orden A, B y C. Cada fila esta ordenada por la cantidad de materias inscritas y cada columna, que comienza por cero, refleja la probabilidad de que el estudiante apruebe número-columna–materias.

1.3. Generación aleatoria de problemas

Con el fin de que puedas experimentar y verificar tus análisis y programas, te proveemos un generador aleatorio de problemas. Fundamentalmente su sintaxis es la siguiente:

```
./rand-gen -n num-cursos -m num-estudiantes
```

El generador produce al azar un problema de asignación. Por ejemplo, el comando

```
./rand-gen -n 200 -m 10000
```

Genera 200 cursos al azar y 10000 estudiantes al azar con selecciones al azar.

El siguiente ejemplo

```
./rand-gen -n 3 -m 8
```

Produce la siguiente salida:

```
3
0 2 1275
1 5 1000
2 2 1000
8
4100 C 1 3 2 1 0
4101 C 1 3 0 1 2
4102 B 3 3 0 1 2
4103 C 1 2 0 1
4104 B 1 1 1
4105 A 1 3 1 2 0
4106 B 1 3 2 1 0
4107 B 1 3 0 1 2
```

La primera línea indica el número de cursos n (3 en el ejemplo). Las siguientes n líneas especifican los cursos. Cada línea de curso se estructura en tres campos: el código o identificador de curso, el cupo del curso, es decir la máxima cantidad de estudiantes que podrían inscribirse, y el coste de honorarios del curso por estudiante.

La línea siguiente a las n correspondientes a los cursos contiene la cantidad de estudiantes m. Las líneas siguientes contienen a los estudiantes y sus selecciones. Cada línea de estudiante se estructura en: el identificador del estudiante, el tipo de estudiante (A, B o C), la máxima cantidad de materias que el estudiante quiere cursar, el número de materias que el estudiante ofrece cursar y los códigos de las materias que el estudiante seleccionó. Por ejemplo, la línea:

```
4103 C 1 2 0 1
```

Indica que el estudiante 4103 pertenece al grupo C, desea cursar a lo sumo una sola materia y seleccionó 2 materias, cuyos códigos son 0 y 1 respectivamente. Nota que aunque el estudiante haya seleccionado dos materias, el sistema le asignará a lo sumo una sola (que es lo máximo que él está dispuesto).

Ejecuta

```
./rand-gen -h
```

Para ver una explicación más detallada de los distintos parámetros de ejecución.

El fuente de `rand-gen` está distribuido con este laboratorio. Para poder compilarlo y encadenarlo debes tener instalada la biblioteca `TCLAP`.

1.4. Modelo de red de Flujo

El problema de la máxima asignación de cupos a menor coste se resuelve de manera "simple" mediante un modelo de red de flujo con costes con las siguientes premisas.

1. Los nodos que representan los cursos se conectan con el nodo sumidero. Cada arco del curso al sumidero contiene como capacidad el cupo del curso y como coste el coste de los honorarios del curso. Sólo puede haber un arco del curso hacia el sumidero.

2. El nodo fuente está conectado a los estudiantes. Sólo puede haber un arco del fuente hacia un estudiante. Cada arco tiene como capacidad la máxima cantidad de cursos que el estudiante está dispuesto a tomar y como coste el coste esperado del estudiante, el cual, recuerda, es función del grupo al que pertenezca el estudiante y de la cantidad de materias que cursaría (ver § 1.2).

3. Por cada materia que haya seleccionado el estudiante, existe un arco con capacidad unitaria y coste cero desde el estudiante hacia el curso.

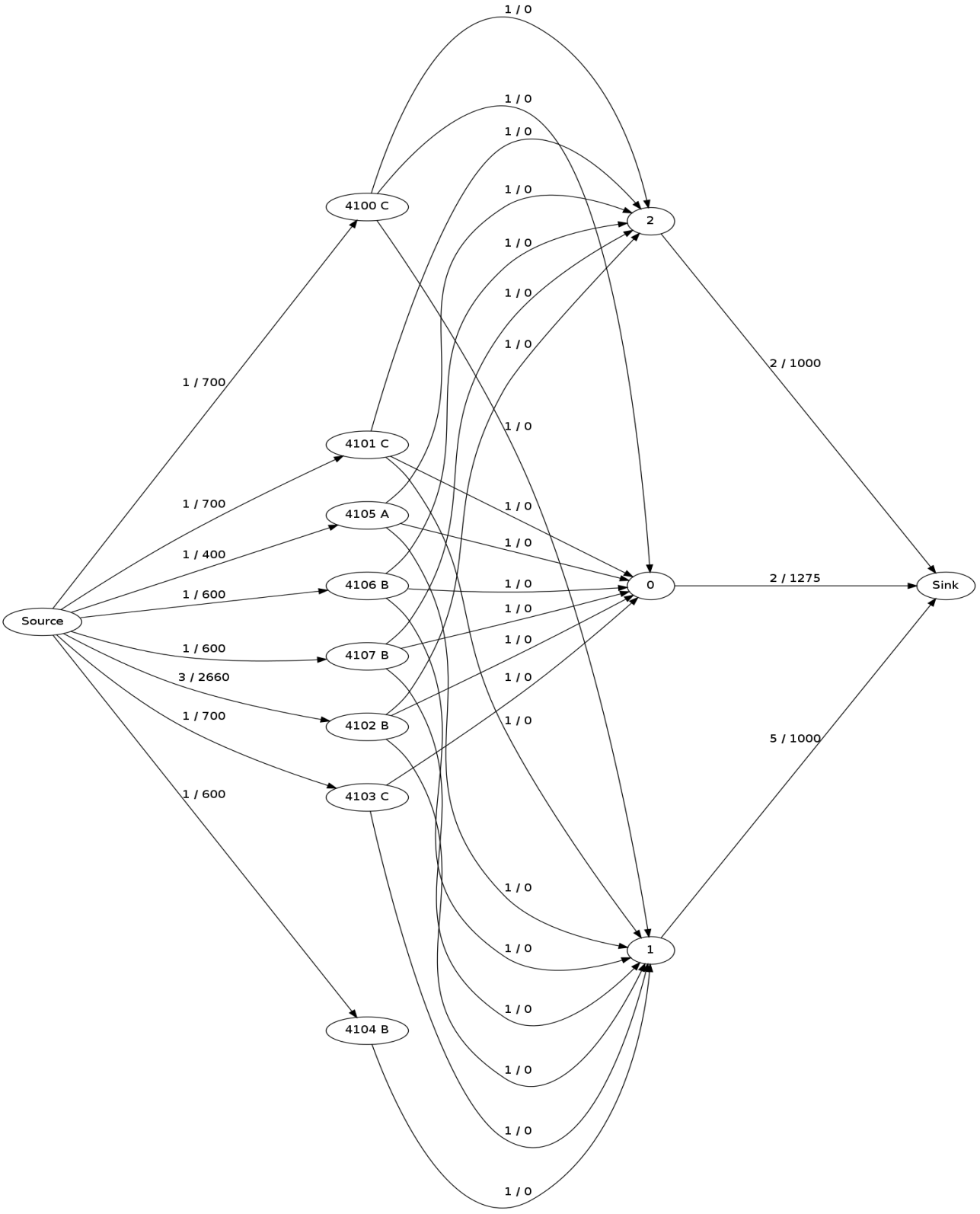

Figura 1: Una simplísima red de ejemplo

Para el siguiente problema[1]:

```
3
0 2 1275
1 5 1000
2 2 1000
8
4100 C 1 3 2 1 0
4101 C 1 3 0 1 2
4102 B 3 3 0 1 2
4103 C 1 2 0 1
4104 B 1 1 1
4105 A 1 3 1 2 0
4106 B 1 3 2 1 0
4107 B 1 3 0 1 2
```

Y la siguiente matriz de probabilidades de grupos;

```
0.4   0.6   0.0   0.0
0.5   0.3   0.2   0.0
0.6   0.2   0.1   0.1
0.6   0.4   0.0   0.0
0.7   0.2   0.1   0.0
0.8   0.1  0.06  0.04
0.7   0.3   0.0   0.0
0.8  0.15  0.05   0.0
0.9  0.05  0.03  0.02
```

La red de flujo con costes resultante es como la de la figura 1.

La meta es pues calcular una asignación máxima al menor coste. Nota que el coste está dividido en dos partes:

1. El producto de la cantidad de estudiantes por el coste de honorarios del curso.

2. El coste esperado del estudiante según su grupo en función de la máxima cantidad de materias que él puede tomar.

   Este es el mecanismo por el cual el algoritmo de optimización pondera en el proceso de selección el grupo al que pertenezca el estudiante, así como también la cantidad de materias que él cursaría.

   Nota que una vez efectuado el cálculo este coste puede cambiar si al estudiante se le asignan menos materias que las que él pidió. Distinguiremos este coste como "coste esperado asignado", el cual sólo es posible conocer una vez que se ha hecho la asignación.

[1]Generado mediante `./rand-gen -n 3 -m 8 -c 2 -S 800`

1.5. Modelo $\aleph_\omega$ (Aleph-ω) de red de flujo con costes

El modelo $\aleph_\omega$ (Aleph-ω) de la red que acabamos de definir está especificado en el archivo `net.H`, cuyo contenido resumido es el siguiente:

```
struct Course : public Uid
{
  size_t cap; // cupo en cantidad de estudiantes
  Ulong cost; // coste por estudiante
  DynList<Ulong> student_list; // Lista asignada de estudiantes
};

struct Student : public Uid
{
  char type;          // A: excelente, B: regular, C: no muy bueno
  short num_courses; // Número de cursos que este estudiante desea tomar
  DynList<Ulong> selected_courses; // cursos seleccionados
  DynList<Ulong> assigned_courses; // cursos asignados
};

using Node = Net_Cost_Node<Uid*>;
using Arc = Net_Cost_Arc<Empty_Class, long>;
using Net = Net_Cost_Graph<Node, Arc>;
```

La clase `Uid` representa un identificador único, el cual es heredado por los cursos y los estudiantes. El identificador es accesible a través del miembro dato `id`.

Nota que los nodos de la red contienen punteros de tipo `Uid*`. Puesto que tanto los estudiantes como los cursos heredan `Uid`, es posible obtener un puntero a la clase derivada a partir de un puntero `Uid*`. En este sentido, puede serte de interés emplear la conversión dinámica `dynamic_cast` para averiguar que el de puntero del nodo y eventualmente recuperarlo al curso o estudiantes según sea el caso.

En este laboratorio deberás calcular, entre otras cosas, la lista de estudiantes asignados (`student_list`) a cada curso y la lista de cursos asignados a cada estudiante. Esta información la puedes obtener luego de maximizar la red al mínimo coste.

Para poder calcular las asignaciones efectiva y, dadas las circunstancias, lo más eficientemente posible, debes estudiar y entender bien dos algoritmos impartidos en el curso: el algoritmo de Bellman-Ford para detección y búsqueda de ciclos negativos y el algoritmo de minimización de flujo a coste mínimo por cancelación de ciclos negativos.

1.6. Detección y búsqueda de ciclos negativos mediante el algoritmo de Bellman-Ford

Grosso modo el algoritmo general de Bellman-Ford es como sigue:

1) Si s es el origen, iniciar $\mathrm{Acc}(s) = 0$ y $\forall v \in V - \{s\}, \mathrm{Acc}(v) = \infty$
2) `for int i = 0; i < V; ++i`
   2.1) $\forall e = u \overset{w}{\to} v \in E \Longrightarrow$ // relajar todos los arcos
   `if` $\mathrm{Acc}(u) + w < \mathrm{Acc}(v) \Longrightarrow$
   `pred[v] = u` // actualiza árbol abarcador
   $\mathrm{Acc}(v) = \mathrm{Acc}(u) + w$ // relaja arco

3) $\forall e = u \overset{w}{\to} v \in E \Longrightarrow$ // verifica si hay ciclo negativo
   `if` $\mathrm{Acc}(u) + w < \mathrm{Acc}(v) \Longrightarrow$
   Hay ciclo negativo $\Rightarrow$ terminar el lazo y luego recuperar el ciclo del `pred`

Como debes recordar, el algoritmo mantiene en el arreglo `pred` un árbol abarcador de todos los caminos mínimos que parten desde `start`. Si existe un ciclo negativo, este puede detectarse de dos formas:

1. Si la última pasada 3) se logra relajar algún arco, entonces con certitud existe un ciclo negativo.

   De existir un ciclo negativo éste puede recuperarse con certitud mediante el arreglo `pred`. Para ello, se construye un grafo y se invoca al algoritmo de Tarjan para encontrar un ciclo.

   Nota que en este caso se pagan completamente $\mathcal{O}(V\ E)$ relajaciones antes de proceder a buscar el ciclo.

2. Puesto que de existir un ciclo negativo éste queda "grabado" en `pred`, cada cierto tiempo, luego de ejecutado 2.1), podríamos examinar el arreglo y ver si en él se encuentra un ciclo.

   La ventaja de este enfoque es que si existe un ciclo negativo existen grandes probabilidades de encontrarlo en `pred` mucho antes de que culminen todas las relajaciones. La desventaja es que si no existe tal ciclo, entonces perderemos el trabajo todas las veces que lo busquemos en `pred`.

Como ves, es algo dilemático decidir cual esquema o combinación utilizar.

En $\aleph_\omega$ (Aleph-ω), todo lo pertinente al algoritmo de Bellman-Ford está contenido en la clase `Bellman_Ford`, contenida en `Bellman_Ford.H`.

En virtud de las consideraciones hechas, la búsqueda de ciclos negativos la realiza el siguiente método:

```
    tuple<Path<GT>, size_t>
search_negative_cycle(Node * start, double it_factor, const size_t step)
```

`start` es el nodo a partir del cual se calculan los caminos mínimos. `it_factor` es un factor de iteraciones. Después de que el algoritmo efectúe la pasada `it_factor` $\times V$, se sondea si en `pred` se encuentra un ciclo. De encontrarse éste, entonces el algoritmo

se detiene y se retorna una dupla conteniendo el ciclo y el número de iteraciones respecto a V en la cual se halló el ciclo. De lo contrario, el algoritmo espera hasta `it_factor` $\times V +$ `step` iteraciones antes de volver a buscar un ciclo. Si al final de las V pasadas no se encuentra un ciclo, entonces se efectúa una última pasada que determinará si existe o no el ciclo. De detectarse, entonces este se extraerá de `pred`, Si no existe un ciclo negativo, entonces la dupla contiene un camino vacío.

1.7. Minimización de costes por cancelación de ciclos

El algoritmo empleado por $\aleph_\omega$ (Aleph-ω) para minimizar una red de flujo máxima es el de cancelación sucesiva de ciclos. El enfoque de $\aleph_\omega$ (Aleph-ω) puede reflejarse en el siguiente esquema que toma en cuenta el factor de iteraciones `it_factor` y el paso `step`:

1) Maximice el flujo de la red N
2) `source =` N`.get_source()`

3) `search:` // Aquí comienza la búsqueda iterativa de ciclos negativos
`while (true)`
    $c =$ `search_negative _cycle(source, it_factor, step)`
    `if` c no contiene un ciclo $\Longrightarrow$
        `break`
    Cancele el ciclo c de N
    Ajuste `it_factor` a la cantidad de iteraciones que se tomó encontrar el ciclo

4) $c =$ `search_negative_cycle(it_factor, step)`
5) `if` c contiene un ciclo $\Longrightarrow$
    Cancele el ciclo c de N
    `goto search;`

Si a ti u otro les enrolla que este programa contenga la instrucción `goto`, entonces se les recomienda que lean:

- *"Structured Programming with go to Statements"*, Donald E. Knuth, ACM Computing Surveys, Volume 6 Issue 4, Dec. 1974, Pages 261-301.

  o

- *"Code Complete"*, Steve McConnel, Microsoft Press; 2nd edition (June 19, 2004). Especialmente la sección 17.3

El enfoque puede estructurarse en dos partes:

1. La primera parte, cual comienza en el `while`, busca ciclos negativos mediante el algoritmo de Bellman-Ford iniciado desde el nodo fuente.

   Si un ciclo negativo es encontrado, entonces el proceso se repite, pero el factor iterativo es ajustado a que la búsqueda de ciclos comience a partir del nuevo factor. La heurística es que conforme se vayan reduciendo la cantidad de ciclos negativos de la red, más iteraciones harían falta para encontrarlo. De este modo, si se ajusta `it_factor`, se podrían ahorrar búsquedas vanas.

2. La segunda parte se ejecuta cuando no se encuentran ciclos negativos tomando al nodo fuente como inicio de la búsqueda. En este caso, aún pueden existir ciclos negativos inalcanzables desde el nodo fuente.

   Para buscar ciclos negativos inalcanzables desde el nodo fuente, se crea un nodo ficticio x y se conecta al resto de los nodos. Luego se inicia el algoritmo de Bellman-Ford desde x. Si existe un ciclo negativo, entonces éste con toda seguridad será alcanzable desde x, pues desde este último se alcanzan a todos los nodos. Esto es lo que hace la llamada `search_negative_cycle(it_factor, step)` (que no recibe nodo origen como parámetro).

   La idea de la separación es ahorrar tiempo, pues la creación de x y de V arcos y luego su destrucción lleva tiempo. Por otra parte, como x es un nodo arbitrario, la búsqueda de ciclos se hace de manera arbitraria. En cambio, cuando se buscan desde el fuente hay bastante probabilidad de que sean rápidamente alcanzables desde él.

La interfaz $\aleph_\omega$ (Aleph-ω) del algoritmo es la siguiente:

```
template <class Net,
  template <class> class Max_Flow_Algo = Ford_Fulkerson_Maximum_Flow>
struct Max_Flow_Min_Cost_By_Cycle_Canceling
{
  tuple<size_t, double> operator () (Net & net,
                                     double it_factor = 0.5,
                                     size_t step = 1);
};
```

El parámetro plantilla `Max_Flow_Algo` es el algoritmo de maximización a utilizar. La rutina retorna una dupla contentiva del número de ciclos negativos que fueron encontrados y el valor final del factor de iteración.

Con este laboratorio se te distribuye una versión del algoritmo, que se encuentra en `cancel.C`. configurada para imprimir trazas que te permitan observar y estudiar su comportamiento. Puedes usarla con la seguridad de que será el mismo algoritmo conque se te evaluará.

Nota que `it_factor` y `step` tienen valores por omisión. Estos valores no necesariamente ni probablemente serán los adecuados. En este laboratorio deberás investigar cuáles son los mejores valores de `it_factor` y `step`. También deberás seleccionar cuál algoritmo de maximización de flujo es el más adecuado. Del resultado de tu investigación depende que tengas éxito o no minimizando el coste de la red. El evaluador probará tu selección mediante una llamada a:

```
max_flow_min_cost_by_cycle_canceling(net);
```

1.8. Solucionador

En este laboratorio se te provee un programa "solucionador", cuya mínima sintaxis básica y obligatoria es:

```
./solver -f matriz-probs
```

`matriz-probs` es una matriz 9×4 que contiene las matrices de probabilidades A, B y C, en ese orden. `solver` queda a la espera de la red.

Ejecuta `./solver -h` para conocer los distintos parámetros del solucionador.

En este laboratorio deberás ejecutar `solver` muy a menudo, pues es tu "mejor" herramienta disponible para estudiar los valores de `it_factor` y `step`.

Es posible y bastante práctico usar `solver` en combinación con `rand-gen`. Por ejemplo

```
./rand-test -n 100 -m 5000 -C 2000 -S 1000 -s 18 -c 30 | ./solve \
-f probs.txt -a ford-fulkerson -c 800 -i 0.1 -s 100
```

Aquí se genera un problema de 100 cursos, 5000 estudiantes, con un coste gaussiano de honorarios por curso de 2000, un cupo poissoniano por curso de 30 y semilla aleatoria de 18. Luego, la salida de `rand-gen` se le pasa como entrada a `solver`, el cual usa al algoritmo de Ford-Fulkerson para maximizar la red, un factor de iteración de 0.1 y un paso de 100.

Se te proporcionan redes ya generadas en los archivos con nombre `in-n.txt`. Ten cuidado con modificarlos, pues `solver` no se cuida para nada de malas entradas y bien podría agotar la memoria del computador si algún valor de entrada es incorrecto. Para usar alguna entrada ejecuta:

```
./solver -p probs.txt < in-n.txt
```

2. Laboratorio

Tu trabajo en este laboratorio será implementar la clase `Solver` incluida en `solver.H`.

Lee y comprende bien todos los métodos de la clase `Solver` antes de diseñar su estado interno. Ten bien claro cómo vas a implementar cada método.

Puedes modificar programa `solver.C` para modificar el solucionador y probar las distintas rutinas. Ten sumo cuidado con no modificar la lectura de la red.

En este laboratorio tendrás una cantidad máxima de evaluaciones. Por consiguiente, debes ser muy cuidadoso y verificar muy bien tu solucionador antes de someter. **No debes usar al evaluador como verificador**. Construye casos de prueba sólidos de cada uno de tus métodos.

2.1. Constructor

Programa

```
Solver(Prob_Mat && pA, Prob_Mat && pB, Prob_Mat && pC,
       DynList<Course> && course_list, DynList<Student> && st_list,
       const long cost_by_student)
```

`pA`, `pB` y `pC` son las matrices de probabilidad para los grupos A, B y C respectivamente. `course_list` es la lista de cursos con cupos y honorarios, pero sin estudiantes asignados. `st_list` es la lista de estudiantes con su tipo, cantidad máxima

de materias y las materias seleccionadas, pero sin materias asignadas. Finalmente, `cost_by_student` es el costo de manutención por estudiante.

Quizá una de las principales dificultades de este laboratorio es que para poder realizar los siguientes cálculos requieres mantener un conjunto de estructuras de datos. Nuevamente la insistencia en que estudies y te asegures de entender cada método antes de sentarte a programarlo. Diseña cuidadosamente tus estructuras de datos y prepara el solucionador para entradas grandes, pero realistas (aproximadamente 300 cursos y 10000 estudiantes).

2.2. Lista de estudiantes

Programa

```
const DynList<Student> & get_students() const;
```

La cual retorna la lista de estudiantes ordenada por su identificador de todos los estudiantes entrados al constructor.

2.3. Lista de cursos

Programa

```
const DynList<Course> & get_courses() const;
```

La cual retorna la lista de cursos ordenada por su identificador de todos los estudiantes entrados al constructor.

2.4. Consulta de estudiante

Programa

```
const Student & get_student(Ulong id) const;
```

La cual retorna una referencia constante al estudiante `id`. El método debe generar la excepción `domain_error` si el `id` no es encontrado.

2.5. Consulta de curso

Programa

```
const Course & get_course(Ulong id) const;
```

La cual retorna una referencia constante al estudiante `id`. El método debe generar la excepción `domain_error` si el `id` no es encontrado.

2.6. Coste esperado de manutención

Programa

```
double expected_cost(const Student & st) const;
```

La cual, en función de las probabilidades entradas al constructor, el grupo al que pertenezca el estudiante y el número de cursos que él dispuesto a tomar, retorna el coste esperado de manutención.

Este método es crítico y, a pesar de que no lo parezca, algo enredado. Debe reusarse para construir la red de flujo. Asegúrate de probarlo minuciosamente, pues de estar incorrecto, casi con toda seguridad la red y el resto de los cálculos estén errados.

2.7. Construcción de la red de flujo

Ahora que ya tienes la maquinaria de consulta y de cálculo del coste esperado de manutención, ya estás en capacidad de construir la red de flujo. Programa el siguiente método:

```
const Net & build_net();
```

El cual construye la red de flujo con base a los requerimientos explicados en § 1.5. La rutina retorna una referencia constante a la red con flujo cero para todos los arcos.

El orden de construcción de la red debe ser el mismo que el especificado en § 1.5: primero el sumidero, luego los cursos, luego conectar los cursos al sumidero, luego el fuente, luego los estudiantes, luego conectar el fuente a los estudiantes y finalmente conectar los estudiantes a los cursos.

Trata de reutilizar la rutina anterior `expected_cost()` para asignar el coste de manutención esperado en los arcos que van del fuente hacia cada estudiante. No te preocupes por el hecho de que los costes están especificados como `long` en la red pero que el resultado es `double`. La razón de manejar `long` para los costes es evitar errores de redondeo durante la evaluación. Sin embargo, puesto que el coste de manutención suele ser tres ordenes de magnitud mayor que la $\mathrm{P}(A = n|m)$ (que está entre cero y uno), el coste esperado de manutención suele ser entero.

Demás está decirte que el destino de tu trabajo depende en gran parte de que esta red esté construida correctamente.

2.8. Asignación

Una vez que estés completamente seguro de que tu red está correcta programa:

```
template
  <template <class, template <class> class> class Max_Flow_Min_Cost_Algo,
   template <class> class Max_Flow_Algo = Tu-Algoritmo-de-Flujo-Máximo>
  const Net & assign(double it_factor = tu-valor, size_t step = tu-valor)
```

El método calcula la asignación óptima. Para ello, usa el algoritmo de maximización de flujo `Max_Flow_Algo` y el de minimización de coste `Max_Flow_Min_Cost_Algo`; para este último debes usar `Max_Flow_Min_Cost_By_Cycle_Canceling`, especificado en `tpl_netcost.H`, pues ese será el algoritmo que invocará el evaluador. Pero recuerda que puede probar confiadamente con la versión contenida en `cancel.C`.

Después de invocado este método las asignaciones de cursos y estudiantes deben estar actualizadas. En el caso de los cursos, el atributo `student_list`; en el de los estudiantes `assigned_courses`.

El evaluador efectuará y evaluará la siguiente llamada:

```
assign<Max_Flow_Min_Cost_By_Cycle_Canceling>();
```

Lo que significa que tú debes escoger el algoritmo de maximización de flujo, el factor de iteración y el paso de incremento. De tu selección depende que pases o no el test.

Para maximización de flujo puedes emplear cualquiera de los cinco algoritmos:

```
Ford_Fulkerson_Maximum_Flow

Edmonds_Karp_Maximum_Flow

Fifo_Preflow_Maximum_Flow

Heap_Preflow_Maximum_Flow

Random_Preflow_Maximum_Flow
```

Los algoritmos basados en empuje de preflujo (preflow) no fueron estudiados en clase, pero pueden ser perfectamente usados. Ellos suelen ser mucho más rápidos que los algoritmos basados en caminos de aumento, pero éstos últimos crean menos ciclos negativos.

Es indispensable que investigues meticulosamente la mejor selección. La índole bipartita de la red no cambiará. Prepárate para escalas aproximadas de 10000 estudiantes y 500 cursos de aproximadamente 30 de cupo, pero aumenta la escala progresivamente, conforme vayas descubriendo. Comienza con escalas pequeñas y verifica minuciosamente tus resultados.

Está permitido compartir datos, redes y resultados. De modo que entre todos se pueda verificar si la red se está construyendo y/o ésta se está optimizando correctamente. En pos de la transparencia y el honor, **haz estos compartimientos públicos en el foro**.

Está absolutamente prohibido compartir la combinación de parámetros que hayas descubierto. En primer lugar, plagiar esta combinación te impide el aprendizaje que conlleva la búsqueda de estos parámetros. En segundo lugar, parafraseando un principio de investigación que dice *cuando una cosa es descubierta otra es oculta*, plagiar estos parámetros puede impedir que se descubran combinaciones mejores.

Para evaluar esta rutina se te dará el doble del tiempo que le tome al evaluador.

2.9. Coste de manutención esperado asignado

Programa:

```
double assigned_expected_cost(const Student & st) const;
```

La cual retorna el coste de manutención esperado de la asignación, no de la selección. Es decir, en función del coste esperado la cantidad de materias asignadas.

2.10. Costo de los honorarios

Programa:

```
double cost_gain(const Course & c) const;
```

El cual retorna el costo de honorarios del curso; es decir, el producto de la cantidad de estudiantes asignados por el coste de honorarios del curso.

2.11. Costo esperado del curso

Programa:

```
double expected_payment(const Course & c) const
```

El cual retorna el costo esperado del curso por los estudiantes inscritos.

2.12. Lista de costes de manutención

Programa

```
DynList<tuple<Ulong, double>> students_expected_payment() const;
```

La cual retorna una lista ordenada por id de duplas conteniendo el id del estudiante y el coste esperado de manutención asignado.

2.13. Lista de costes de cursos

Programa

```
DynList<tuple<Ulong, double, double>> course_payment() const;
```

La cual retorna una lista de tripletas conteniendo el id del curso, es coste de honorarios del curso y el coste esperado de manutención. La lista está ordenada por id.

Nota que el tercer coste puede estar sobrestimado si se contabilizan estudiantes con varias materias asignadas.

3. Evaluación

La fecha de entrega de este laboratorio es hasta el martes 6 de diciembre de 2016.

Puesto que este laboratorio aborda un problema que podría demorar minutos en resolverse, su evaluación será parcialmente manual. Por consiguiente, tienes un máximo de 3 envíos en 3 días diferentes, es decir, no podrás enviar dos veces un mismo día. Ten en cuenta que si llega el último día de entrega y no has hecho el primer envío, solamente tendrás una única oportunidad.

El evaluador tiene una duración máxima determinada por la suma:

- Tiempo t_a en segundos que tome el evaluador en resolver la asignación.
- $2t_a$, es decir el doble de lo que tome el evaluador para optimizar, lo cual se presume que es el máximo tiempo que se te da para resolver tu asignación.
- 20 segundos para el resto de los cálculos.

Dicho lo anterior, es muy importante que tus rutinas sean rápidas, especialmente `get_course()` y `get_student()`, pues éstas son usadas internamente por el evaluador para evaluarte otras rutinas. Si, por ejemplo, `get_student()` es lineal, entonces tienes una alta probabilidad de que el evaluador no culmine (y tengas cero).

Hay dos puntos de bonificación si tu asignación bate o es al menos un 5 % más rápida que la del evaluador. Pero atención, si el evaluador es batido, él aprenderá tus valores, así que toma muy en cuenta este hecho si consideras volver a someter.

Para evaluarte debes enviar el archivo `solver.H` a la dirección:

`alejandro.j.mujic4@gmail.com`

Antes de la fecha y hora de expiración. El "subject" debe **obligatoriamente** contener el texto "**AYDA-LAB-04**" sin las comillas. En el mensaje y en el inicio del fuente del programa debes colocar los nombres, apellidos y números de cédula de la pareja que somete el laboratorio. **Por favor, no uses otros medios para enviar la solución ni escribas otros comentarios adicionales en el email**.

Atención: si tu programa no compila, entonces el evaluador no compila. Por favor, **no asumas que una compilación exitosa de tu parte implica una exitosa del evaluador**. Si una de tus rutinas se cae, entonces el evaluador se cae. Si una de tus rutinas cae en un lazo infinito o demora demasiado, entonces el evaluador cae en un lazo infinito o se demora demasiado. Exactamente sucedería lo mismo si tu programa fuese parte de un sistema de aviónica: se estrella el avión. Por esa razón, en todos estos casos será imposible darte una nota ... o simplemente tienes cero. Si una de tus rutinas dispara una excepción inesperada, el evaluador hará lo posible por capturarla y reportártela; sin embargo, esto no es un compromiso; podría estrellarse el avión. En todo caso, si el evaluador no logra capturar la excepción, no se te dará nota pero se te reportará la excepción.

Demás está decirte que está absolutamente prohibido compartir código. **Por favor, guarda este proyecto y no lo compartas en el futuro**.

4. Recomendaciones

1. Si no estás dispuesto a empezar este laboratorio prontamente, entonces es preferible que no lo hagas.

2. Diseña unas tres redes de prueba con escalas manejables y visibles. El solucionador tiene una opción `-g` que genera el graphviz de la red con costes y otra `G` que genera el graphviz con flujo. Quizá estas visualizaciones puedan ayudarte.

3. Tomate el tiempo para meditar sobre las estructuras de datos que emplearás y bosqueja qué tienes que hacer para implementar cada método. **No te precipites a programar si no tienes esto claro**.

4. Por cada método, diseña tus casos de prueba. Para ello puedes modificar el solucionador. Usa `git` u otro manejador de versiones para mantener un historial de tu desarrollo y poder recuperarte en caso de que algo te resulte mal.

5. Ten sumo cuidado con `expected_cost()`. Si no logras calcular el costes esperado de manutención todo te saldrá mal.

6. Usa la versión del algoritmo de minimización de coste puesta en `cancel.C`. Puedes modificarla para que te imprima mejores trazas según tu circunstancia. Pero no hagas a `assign()` dependiente de ella.

7. El `Makefile` tiene una opción para construir `solver-op`, la cual construye el solucionador optimizado y sin verificaciones de invariantes. Esto lo hace notablemente más rápido pero quedas sin defensas de la biblioteca. Usa esta opción con cuidado, especialmente para escalas grandes y cuando estés estudiando los mejores valores para `assign()`. El evaluador se ejecutará optimizado.

8. Usa el foro de discusión.

9. La opción `-v` del solucionador imprime trazas de procesamiento. Úsala o modifícala si lo consideras necesario.