Laboratorio de Lenguajes de Programación

USB / CI-3661 / Ene-Mar 2016

(Programación Lógica – 25 puntos)

Pares e impares (4 puntos)

Considere la siguiente multiplicación

```
  IPP *
   PP
------
 PIPP +
 PIP
------
 IIPP
```

donde `P` representa algún dígito par e `I` representa algún dígito impar. Note que hay varias ocurrencias de `P` en la expresión, sin embargo podrían ser dígitos diferentes, siempre y cuando sean pares. Lo mismo aplica para las ocurrencias de `I`.

Escriba el predicado `pei/0` que triunfa presentando en pantalla la solución al problema. La solución debe presentarse con el mismo formato, sustituyendo cada dígito por el valor correspondiente.

Reorganización de trenes (8 puntos)

Ud. es el operador de cambia vías del Yalda de Calacras – el casi-metro de una urbe distópica futurística. Las formaciones (o "trenes", para el pueblo) se especifican como una lista Prolog

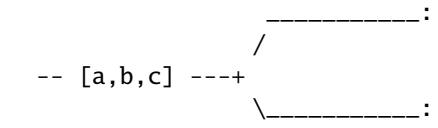
```
[c1,c2,c3,...,cN]
```

donde los `cI` son los vagones. La máquina principal está ubicada del lado de `c1` (como si fuera `c0`), pero no está indicada explícitamente porque sólo interesan los vagones.

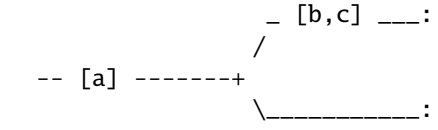
Ud. se encuentra en el patio de operaciones, junto a la "Y" de intercambio. La formación llega por el pie de la "Y" y puede usar los brazos de la "Y" para tomar o dejar vagones. Su trabajo es transformar la formación que llega a una permutación específica.

Por ejemplo, si llega la formación `[a,b,c]` y se necesita reordenarla para tener la formación `[b,c,a]`, podría procederse de la siguiente manera. (Imagine que la "Y" está acostada)

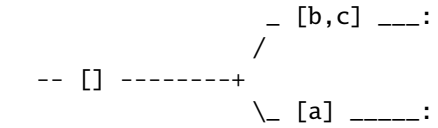
- Llegada del tren, por la base de la "Y".



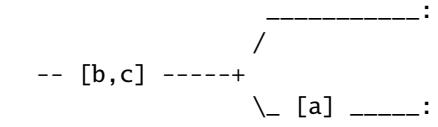
- Retrocede al brazo superior, desengancha los dos últimos vagones, `[b,c]`, y regresa a la base de la "Y".



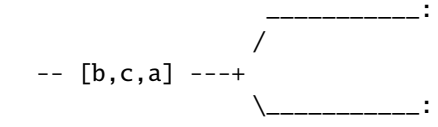
- Retrocede al brazo inferior, desengancha el único vagón, `[a]`, y regresa a la base de la "Y" – recuerde que la locomotora es tácita.



- Retrocede al brazo superior, engancha `[b,c]` y regresa a la base de la "Y".



- Retrocede al brazo inferior, engancha `[a]` y regresa a la base de la "Y".



Esta serie de operaciones son representadas por la siguiente secuencia de instrucciones

```
[push(above,2), push(below,1), pop(above,2), pop(below,1)]
```

Si el significado de la lista no es obvio, piense que `above` y `below` hacen referencia a los brazos superior e inferior de la "Y"; que `push` y `pop` hacen referencia a dejar o tomar vagones; y que el número es la cantidad de vagones involucrados en la operación.

Se desea que Ud. implante el predicado `vagones/3` que sea invocado de la forma

```
?- vagones([a,b,c],[b,c,a],Operaciones).
```

y triunfe unificando `Operaciones` con la lista resultante, que además debe ser la solución *más corta* posible para el problema.

Si después de estudiar el problema necesita preguntar si los trenes pueden estar vacíos, pierde un punto. Si necesita preguntar si los trenes tienen al menos un vagón, pierde otro punto.

Calendario NFL (13 puntos)

La *National Football League* está organizada en dos conferencias, *National* (`nfc` de aquí en más) y *American* (`afc` de aquí en más), cada una de las cuales organizada en cuatro divisiones según su distribución geográfica (`north`, `east`, `south` y `west`).

Cada una de las divisiones tiene cuatro equipos, cuyos nombres representaremos usando átomos Prolog.

Los resultados del 2015

Para comenzar, consulte la página de la NFL y complete los resultados de la temporada 2015-2016. Por ejemplo, la División Norte de la Conferencia Nacional quedó así

```
standings(nfc,north,1,vikings).
standings(nfc,north,2,packers).
standings(nfc,north,3,lions).
standings(nfc,north,4,bears).
```

Complete los resultados que faltan. No, no gana puntos por completar los resultados, pero si no los tiene, nada va a funcionar.

El calendario balanceado

La NFL tiene una estrategia sencilla para intentar que el calendario de cada año sea razonablemente balanceado para todos y cada uno de los equipos, además de atractivo para el público. Cada equipo juega dieciséis (16) partidos en la temporada regular, la mitad como anfitrión y la mitad como visitante.

Las reglas de construcción del calendario son sencillas. Sea el equipo `T` que finalizó en la posición `p` (1 a 4) de la división `D` en la conferencia `C`. Entonces:

- El equipo `T` jugará dos partidos contra cada otro rival de su propia división `D`, uno en casa, y uno visitando. Estos partidos se conocen como "divisionales". En nuestro ejemplo, los `bears` tienen que jugar dos partidos contra `vikings`, dos partidos contra `packers` y dos partidos contra `lions`.

  Esto resulta en seis (6) partidos, tres (3) de local y tres (3) de visitante.

- El equipo `T` jugará un partido contra cada otro rival de *otra* división de su propia conferencia: son los juegos intra-conferencia. Con dos equipos, jugará en casa; y con los otros dos equpos, jugará de visitante.

  Cada año, la liga establece cuál será esa otra división particular. Para el año 2016, encontrará la siguiente información – aplica para *ambas* conferencias.

  ```
  intra(north,east).
  intra(south,west).
  ```

  En nuestro ejemplo, los `bears` (NFC North), tendrán que jugar partidos contra los equipos de la NFC East: dos visitando y dos como local.

  Esto resulta en cuatro (4) partidos, dos (2) de local y dos (2) de visitante. Note que hay más de una forma de escoger local y visitante, y la liga estudia las combinaciones posibles para determinar cuál es más conveniente desde el punto de vista comercial y logístico.

- El equipo `T` jugará un partido contra cada otro rival de *otra* división de la *otra* conferencia: los juegos inter-conferencia. Con dos equipos, jugará en casa; y con los otros dos equipos, jugará de visitante.

  Cada año, la liga establece cual será esa otra división particular. Para el año 2016, encontrará la siguiente información

  ```
  inter(east,west).
  inter(north,east).
  inter(south,north).
  inter(west,south).
  ```

  Donde el primer argumento indica la división en la AFC y el segundo argumento indica la división en la NFC.

En nuestro ejemplo, los `bears` (NFC North), tendrán que jugar partidos contra los equipos de la AFC South: dos visitando y dos como local.

Esto resulta en cuatro (4) partidos, dos (2) de local y dos (2) de visitante. Note que hay más de una forma de escoger local y visitante, y la liga estudia las combinaciones posibles para determinar cuál es más conveniente desde el punto de vista comercial y logístico.

- El equipo `T` jugará un partido contra los otros dos equipos que terminaron en la posición `p` de las restantes divisiones de su propia conferencia, con los cuales no tiene partido divisional ni intra-conferencia.

  En nuestro ejemplo, los `bears` jugarán dos partidos más, uno contra el cuarto de la NFC South y otro contra el cuarto de la NFC West: uno como visitante y otro como local.

  Esto resulta en dos (2) partidos.

Si ha seguido la explicación hasta este punto notará que para *cualquier* equipo, se tienen dieciséis partidos balanceados en relación al año anterior, pues se juegan cuatro partidos contra primeros, cuatro contra segundos, cuatro contra terceros y cuatro contra cuartos, independientemente de la posición en la cual haya quedado el equipo.

Las restricciones finales.

Los dieciséis partidos deben distribuirse sobre diecisiete (17) semanas. Eso quiere decir que cada equipo tiene un día libre durante la primera mitad de la temporada.

Las fechas y equipos particulares son escogidos por la NFL con criterios que escapan a nuestro ejercicio de programación, de modo que estableceremos un criterio simple: entre las semanas 1 y 8, ambas inclusive, habrá cuatro (4) equipos libres en cada semana.

Dicho de otra forma:

- Habrá catorce juegos cada semana, en las semanas 1 a 8, porque están jugando todos los equipos excepto los cuatro que están de descanso en cada semana.
- Habrá dieciséis juegos cada semana, en las semanas 9 a 17, porque están jugando todos los equipos

Finalmente, se desea que los últimos tres (3) juegos divisionales de *todas* las divisiones, ocurran en las últimas cuatro semanas. Si, está bien escrito – piénselo.

El problema

Se desea que Ud. escriba un predicado `schedule/0` que calcule y muestre calendarios balanceados para la temporada regular 2016 de la NFL, presentándolos en pantalla con un formato similar a:

```
:-? schedule.
Week 1
------
steelers at patriots
packers at bears
chiefs at texans
browns at jets
colts at bills
dolphins at redskins
panthers at jaguars
seahawks at rams
saints at cardinals
lions at chargers
titans at buccaneers
bengals at raiders
ravens at broncos
giants at cowboys

Bye: eagles, falcons, vikings, 49ers
....
```

Si en una semana no hay "byes", la última línea no debe aparecer.

Debe ser evidente para Ud. que hay más de una solución al problema, por lo tanto el predicado debe permitir consultar **todas** las soluciones aprovechando *backtracking*.

¿Por dónde comienzo?

La construcción de calendarios deportivos es un problema **muy** complejo. Sólo para su consumo, y en caso que no sea un conocedor de la NFL, existen aún más restricciones en cuanto a la construcción del calendario (e.g., sólo se hacen dos viajes a la costa oeste, hay ciertos equipos que no juegan en Thanksgiving Day, y hay juegos que no pueden hacerse porque el mismo día hay juegos de la MLB). De modo que el problema que Ud. está enfrentando aquí está bastante simplificado – es un problema de la vida real, y Prolog se ha usado por años para resolver este tipo de problemas.

Se trata de un problema *"generate-and-test"* con un espacio de soluciones **muy** grande, pero que Ud. puede descomponer en varios predicados simples. Como mínimo le sugiero:

- Un predicado que proponga la lista de "byes" – todas las listas de ocho elementos, cada una de las cuales tiene cuatro equipos diferentes.
- Un predicado que calcule todos los rivales para un equipo `T` particular y genere los enfrentamientos necesarios, pero sin un orden particular.

- Un predicado que dado un calendario propuesto, compruebe que se verifican todas las condiciones.

Si escribe el predicado por "fuerza bruta", encontrará que demora muchísimo en producir resultados – eso no es accidental, porque el espacio de soluciones es enorme. Tiene que ser astuto en la forma en que aplica los filtros.

Si lo desea, puede usar las capacidades de Prolog para trabajar con predicados dinámicos modificables a tiempo de ejecución. En ese caso, por favor indique cuáles predicados son dinámicos, qué representan y asegúrese que al terminar cada corrida, la base de datos está en el mismo estado inicial.

Por cierto, si busca en internet "¿cómo hacer calendarios para deportes?" va a encontrar algoritmos con DP o imperativos. Por un lado, no sirven para nada en Prolog, y por otro lado, son para campeonatos "round-robin" de construcción simple, muy diferentes a la NFL.

Detalles de la Entrega

La entrega se hará en un archivo `pP-<carnet1>-<carnet2>.tar.gz`, donde `<carnet1>` y `<carnet2>` son los números de carnet de ambos integrantes del grupo. El archivo *debe* estar en formato TAR comprimido con GZIP – ignoraré, sin derecho a pataleo, cualquier otro formato que yo puedo descomprimir pero que *no quiero* recibir.

Ese archivo, al expandirlo, debe producir un *directorio* que *sólo* contenga:

- El archivo `pei.pro` con la solución a la sección **Pares e Impares**
- El archivo `vagones.pro` con la solución a la sección **Reorganización de Trenes**
- El archivo `nfl.pro` con la solución a la sección **Calendario NFL**
- Sus predicados **deben** estar organizados de acuerdo con las mejores prácticas de programación Prolog, y de manera que se vea correctamente en **cualquier** editor.
- Sus archivos **deben** estar completa y correctamente documentados, siguiendo las mejores prácticas de documentación para Prolog, al menos para la descripción de los predicados.
- Su implantación será verificada utilizando GNU Prolog 1.3 sobre Debian GNU/Linux. Si Ud. quiere trabajar con SWI Prolog, puede hacerlo, pero **no** puede emplear predicados específicos provistos por SWI Prolog – asegúrese que sus soluciones funcionan con GNU Prolog antes de entregarlas.

El proyecto debe ser entregado por correo electrónico a mi dirección de contacto a más tardar el viernes 2016-04-01 a las 23:59. Cada minuto de retraso en la entrega le restará un (1) punto de la calificación final.