Enunciados Actividad 2

Luis Felipe Garzón Bonilla 1089931169

Steven Grisales López 1034289634

Este archivo contiene adjunto solo las capturas del código ya que en este está todo documentado de cómo funciona y para qué sirve o cuales fueron los ejercicios propuestos.

Parte 1:

```
% Definición de hechos:
conexion(vancouver,edmonton,16).
conexion(vancouver,calgary,13).
conexion(edmonton,saskatoon,12).
conexion(calgary,edmonton,4).
conexion(calgary,regina,14).
conexion(saskatoon,calgary,9).
conexion(saskatoon,winnipeg,20).
conexion(regina,saskatoon,7).
conexion(regina,winnipeg,4).

/* Preguntas:
 * 1-¿Existe conexión entre Saskatoon y Vancover -> conexion(saskatoon,vancouver,_)
 * 2-¿Con qué nodo esta conectado Regina y cuál es el costo de cada conexión? -> conexion(regina,Destino,Costo)
 * Prolog va en orden de arriba para abajo */

/* Reglas */
costo_nodo(Origen,Destino,Costo):- conexion(Origen,Intermedio,C1), conexion(Intermedio,Destino,C2),
    Costo is C1 + C2.

/* Preguntas de las reglas anteriores:
 1-Construir regla para determinar si un nodo tiene aristas.
 2-Construir una regla para determinar cuál es el costo para ir de un nodo X a un Z pasando por Y. */


% ¿Es posible viajar desde Edmonton a Calgary? (llamada recursiva infinita):
camino(X,Y):- conexion(X,Y,_), !.
camino(X,Y):- conexion(X,Z,_), camino(Z,Y), !.
/* Este mismo (el de arriba) es usado para verificar si se puede viajar desde un sitio a otro,
sin importar los nodos que hayan entre ellos. */

/* Este cumple la misma función que el de arriba, solo que nos muestra los tres primeros caminos posibles
 * y nos dice cuanto cuesta en total el viaje de esos tres primeros viajes. */
costo_nodoc(Origen, Destino, Costo) :- conexion(Origen, Destino, Costo), !.
costo_nodoc(Origen, Destino, Costo) :- conexion(Origen, Intermedio, C1), costo_nodoc(Intermedio, Destino, C2),
    Costo is C1 + C2.
```

Parte 2:

```
% Definir un diccionario para almacenar los términos de la serie de Fibonacci que ya se han calculado:
:- dynamic fib_dict/2.

% Definir el caso base para fibonacci(0) y fibonacci(1):
fibonacci(0, 0).
fibonacci(1, 1).

% Definir el caso recursivo para fibonacci(N), donde N es un número entero positivo mayor a 1:
fibonacci(N, F) :-
    N > 1,
    ( fib_dict(N, F) -> true ; (
        N1 is N - 1,
        N2 is N - 2,
        fibonacci(N1, F1),
        fibonacci(N2, F2),
        F is F1 + F2,
        asserta(fib_dict(N, F))
    )).

/* La primera línea define el caso base de la serie de Fibonacci,
 *  donde el primer término es 0 y el segundo término es 1.
 * La segunda línea también define un caso base para la serie de Fibonacci,
 * donde el segundo término es 1 y el primer término es 0.
 * La tercera línea define el caso recursivo de la serie de Fibonacci,
 * donde se calcula el término N-ésimo sumando los dos términos anteriores.
 * La regla establece que si N es mayor que 1, entonces el término N-ésimo es
 * la suma de los términos N-1 y N-2, que se calculan recursivamente llamando a la función fibonacci(N1, F1)
 * y fibonacci(N2, F2). El resultado se calcula sumando F1 y F2, y se asigna a F.
 * Por ejemplo, si se llama a fibonacci(6, F), Prolog devolverá F=8,
 * que es el sexto término de la serie de Fibonacci. */

% Otra versión de fibonacci:
my_fibonacci(_, _, 0).

my_fibonacci(A, B, N):-
         write(A),
         write(' '),
         N1 is N-1,
         C is A+B,
         my_fibonacci(B, C, N1).
```