1 Solución alternativa para circuito_2

En el enunciado del problema se propuso un algoritmo para circuito_2 que no garantiza en todos los casos el número mínimo de destinos que satisface a todos los clientes, sino que obtiene una aproximación razonable, a cambio de ser bastante eficiente. En esta sección presentaremos otro algoritmo (más costoso) que sí obtiene el mínimo exacto. Para ello aplicaremos el esquema de "backtracking" y nos veremos obligados a retocar ligeramente los módulos Cliente y Cjt_Clientes.

El esquema consiste en ir construyendo soluciones para el problema quedándonos en todo momento con la mejor e intentando ahorrarnos la comprobación de las soluciones inviables. En este caso, probamos subconjuntos de destinos que satisfacen a todos los clientes y nos vamos quedando con el tamaño del más pequeño. Los destinos se van probando mediante una función recursiva que planteamos como una inmersión de la función original:

```
funcion i-circuito_2(c: Cjt_Clientes; i: nat) ret n:nat
{Pre: i>=0}

{Post: n = "mínimo número de destinos que satisface a todos los clientes
no marcados de c[1..i]"}
```

Completamos la función original mediante la correspondiente llamada a la inmersión

```
funcion circuito_2 (c: Cjt_Clientes) dev n:nat;
{Pre: no hay ningún cliente marcado en c }

n:= i-circuito_2(c, N)

{Post: n es el mínimo número de destinos que satisface a todos los clientes de c}
```

Para diseñar i-circuito_2 primero se intenta satisfacer al cliente *i*-ésimo, si aún no lo está, probando iterativamente todos los destinos. Cada vez que se prueba un destino, se marcan todos los clientes satisfechos por él y se aplica la correspondiente llamada recursiva para tratar al resto de clientes (eso significa que momentánemente consideramos a dicho destino como un candidato a aparecer en el conjunto más pequeño que satisface a todos los clientes). Una vez actualizado el valor mínimo se desmarcan los clientes previamente marcados.

Nótese que la operación de desmarcado no existe hasta el momento. Además, ahora cada cliente marcado ha de recordar qué destino ha sido el responsable de marcarlo, para así poder ser desmarcado después de probar dicho destino. Como resultado, el módulo Cjt_Clientes queda con una operación modificada y otra nueva

```
accion marcar_cl_sat (e/s c: Cjt_Clientes; ent i: nat);
{Pre: 1<=i<=M }

  var j: nat;
```

```
  {Inv: se han marcado los clientes no marcados de c en [1..j-1] satisfechos
        por el destino i; 1<=j<=N+1}
  {Cota: N-j+1}

  j:=1;
  mientras j<=N hacer
    si consultar_preferencia (c[j],i) y marcado (c[j])=0 entonces marcar(c[j],i) fsi;
    j:=j+1;
  fmientras

{Post: se han marcado los clientes no marcados de c satisfechos por el destino i }

accion desmarcar_cl_sat (e/s c: Cjt_Clientes; ent i: nat);
{Pre: 1<=i<=M }

  var j: nat;

  {Inv: se han desmarcado los clientes de c en [1..j-1] marcados
        por el destino i; 1<=j<=N+1}
  {Cota: N-j+1}

  j:=1;
  mientras j<=N hacer
    si marcado (c[j])=i entonces desmarcar(c[j]) fsi;
    j:=j+1;
  fmientras

{Post: se han desmarcado los clientes de c marcados por el destino i }
```

El módulo `Cliente` queda con dos operaciones modificadas y una nueva. Su campo `sat` pasa de ser booleano a ser natural y su valor será el destino que lo marcó o 0 en caso contrario (aunque no hemos detallado su código, también habría que modificar la operación `leer_cliente`).

```
accion marcar (e/s cl: Cliente, ent i:nat);
{Pre: - }
  cl.sat:=i
{Post:  cl pasa a estar marcado por el destino i}

accion desmarcar (e/s cl: Cliente);
{Pre: - }
  cl.sat:=0
{Post:  cl pasa a estar desmarcado}

funcion marcado (cl: Cliente) dev i: nat;
{Pre: - }
  i:=cl.sat
```

```
{Post:  si i>0, indica que ha sido marcado por el destino;
        si i=0, indica que no ha sido marcado }
```

Con estos cambios, ya podemos diseñar la operación i-circuito_2:

```
funcion i-circuito_2(c: Cjt_Clientes; i: nat) ret n:nat
{Pre: i>=0}

  var j, m: nat fvar;

  si i=0 --> n:=0;
  [] i>0 --> si marcado(c[i])>0 ---> n:=i-circuito_2(c,i-1)
             sino
               j:=1; n:=M;
               {Inv n = "mínimo número de destinos que satisface al cliente c[i]
                    con destinos en [1..j-1] y a todos los clientes no marcados
                    de c[1..i-1] con cualquier destino"}
               mientras j <= M hacer
                 si consultar_preferencia(c[i],j) --->
                    marcar_cl_sat(c,j);
                    {se han marcado los clientes no marcados de c satisfechos por j}
                    m=i-circuito_2(c,i-1);
                    {HI: m = "mínimo número de destinos que satisface
                         a todos los clientes no marcados de c[1..i-1]"}
                    si m+1<n ---> n:=m+1 fsi; // el +1 es por el destino j
                    desmarcar_cl_sat(c,j);
                 fsi
                 j:=j+1
               fmientras
             fsi
  fsi

{Post: n = "mínimo número de destinos que satisface a todos los clientes
no marcados de c[1..i]"}
```