Pseudocódigo ayuda implementación algoritmo Ramificación y Poda

Consideraciones:

- En `main()` únicamente debéis definir la matriz **B** de beneficios y a continuación llamar a la función `AsignacionTrivial(B,s,&nNodos)` y `AsignacionPrecisa(B,s,&nNodos)`, donde `nNodos` debe representar el número de nodos explorados con valor inicial `0` y `s` el vector solución.
- Debéis usar el **TAD lista** dentro de cada una de estas funciones para representar la LNV. Podéis usar el utilizado para la práctica de Hash. En este caso, tal y como está en la diapositiva 68 de teoría, debéis definir la constante **N** (número de personas y tareas) y el **NODO** (o `TIPOELEMENTO`) de cada elemento de la lista **en el archivo `lista.h`**:

```
#define N 3
typedef struct {
    int tupla[N];           //solución parcial del nodo
    int nivel;              //nivel del nodo
    int bact;               //beneficio actual
    int usadas[N];          //vector de tareas usadas (valores 0 o 1)
    float CI, BE, CS;       //cota inferior, beneficio estimado, cota superior
}TIPOELEMENTO;
typedef TIPOELEMENTO NODO;    //Para usar el nombre nodo en RyP
```

- Dentro de cada una de las funciones de asignación (`AsignacionTrivial()` y `AsignacionPrecisa()`) tendréis que realizar las inicializaciones necesarias (para el nodo raíz diapositiva 67, para el resto diapositivas 69-71):

```
////////////////////////////////////////////////////////////////////////////////
//INICIALIZACIONES
////////////////////////////////////////////////////////////////////////////////
TLISTA LNV;                  // Lista de nodos vivos
NODO raiz, x, y;             // Nodo raíz, nodo seleccionado (x) y nodo hijo de x (y)
float C=0;                   // Variable de poda

//Inicialización nodo raíz (diapositiva 67)
raiz.bact = 0;                  // No hay asignaciones. Beneficio acumulado es 0
raiz.nivel = -1;                // Primer nivel del árbol
raiz.tupla[N]={-1,-1,-1};       // tupla de solución sin tareas asignadas
raiz.usadas[N]={0,0,0}          // todas las tareas están sin usar
CI(&raiz,B);                    // Cota inferior de raíz (trivial o precisa)
CS(&raiz,B);                    // Cota superior de raíz (trivial o precisa)
BE(&raiz);                      // Beneficio estimado de la raíz

//Inicialización variable de poda (diapositiva 74)
C = raiz.CI;             // valor inicial variable de poda

//Inicialización lista de nodos vivos (diapositiva 74)
crearLista(&LNV);
// Guardo raíz como primer elemento de LNV
insertarElementoLista(&LNV, primeroLista(LNV), raiz);
```

- Las funciones por desarrollar para utilizar dentro de `AsignacionTrivial()` y `AsignacionPrecisa()` son las siguientes (las cotas CI y CS para estimaciones triviales están definidos en la diapositiva 69 y las cotas para estimaciones precisas están definidas en las diapositivas 70 y 71):

```
// FUNCIONES PARA CALCULAR COTAS Y BENEFICIO
void BE(NODO *x);                          //beneficio estimado trivial-precisa
void CI_trivial(NODO *x);                  //cota inferior estimación trivial (D69)
void CS_trivial(NODO *x, int B[][N]);      //cota superior estimación trivial (D69)
void CI_precisa(NODO *x, int B[][N]);      //cota inferior estimación precisa (D70)
void CS_precisa(NODO *x, int B[][N]);      //cota superior estimación precisa (D71)

//FUNCIONES NECESARIAS PARA PROCEDIMIENTO DE RyP
int Solucion(NODO x);                     //determina si x es solución (D68)
NODO Seleccionar(TLISTA LNV);             //Devuelve nodo según estrategia MB-LIFO

//Funciones privadas necesarias para CI_precisa() y CS_precisa()
int _AsignacionVoraz(NODO x, int B[][N]); //Devuelve valor asignación voraz (D70)
int _MaximosTareas(NODO x, int B[][N]);   //Devuelve valor máximos tareas (D71)
//Función privada necesaria para calcular la solución voraz cuando CI==CS
NODO _SolAsignacionVoraz(NODO x, int B[][N]);
```

- La función `Seleccionar()` debe tener en cuenta que los nodos se insertan en la lista siempre por el principio, pues se sigue una estrategia LIFO pero, dentro de la lista, debéis seleccionar el primer nodo que encontréis con beneficio máximo, por tanto hay que recorrer la lista para buscarlo.
- Dentro del procedimiento descrito en la diapositiva 74 tenemos que generar los hijos de cada nodo `x` seleccionado. Esto está descrito en la diapositiva 67, en la que sustituimos la llamada a la función `Usada()` por el análisis del valor almacenado en el vector `x.usadas[]`:

```
AsignacionTrivial(int B[][N], NODO s)
   …
   //Inicialización
   …
   MIENTRAS LNV≠∅
      x = Seleccionar(LNV) //Selecciona x
      LNV = LNV - {x}      //Elimino x de LNV
      SI x.CS > C          //RAMIFICAMOS: GENERAMOS CADA HIJO
         PARA i = 0..N HACER             //PARA CADA HIJO y DE x
            y.nivel := x.nivel + 1
            y.tupla := x.tupla
            y.usadas := x.usadas
            SI NOT(x.usada[i]) ENTONCES   //NODO VÁLIDO
               y.tupla[y.nivel] := i      //Tarea i a persona nivel
               y.usadas[i] := 1
               y.bact := x.bact + B[y.nivel][i]
               //Calcular CI, CS y BE para nodo y
               …
               SI (Solución(y) AND y.bact>s.bact) ENTONCES
                  s := y.tupla
                  C := max(C,y.bact)
               SINO SI (NOT Solución(y) AND y.CS>C) ENTONCES
                  LNV := LNV + {y} //inserto por el principio (LIFO)
                  C := max(C, y.CI)
               FINSI
            FINSI
         FINPARA
      FINSI
   FINMIENTRAS
```

- En el caso de la estimación precisa de las cotas, hemos visto en el ejemplo de la diapositiva 89 que, cuando `CI==CS`, se aplica una solución voraz directa, por lo que hemos de contemplar ese caso en el algoritmo de la diapositiva del cuadro anterior añadiendo esto como primera condición (texto en azul) y cortando la exploración con un break dado que desde ese punto ya se alcanza la solución voraz. También debemos considerar si ya la raíz cumple esto, dando lugar a una solución voraz desde el principio (segundo texto en azul, que podría evaluarse también antes del MIENTRAS, lo que evitaría meter la raíz en la LNV):

```
AsignacionPrecisa(int B[][N], NODO s)
   …
   //Inicialización
   …
   MIENTRAS LNV≠∅
      x = Seleccionar(LNV) //Selecciona
      LNV=LNV-{x}          //Elimino x de LNV
      SI x.CS > C          //RAMIFICAMOS: GENERAMOS CADA HIJO
         PARA i = 0..N HACER             //PARA CADA HIJO y DE x
            y.nivel := x.nivel + 1
            y.tupla := x.tupla
            y.usadas := x.usadas
            SI NOT(x.usada[i]) ENTONCES   //NODO VÁLIDO
               y.tupla[y.nivel] := i      //Tarea i a persona nivel
               y.usadas[i] := 1
               y.bact := x.bact + B[y.nivel][i]
               //Calcular CI, CS y BE para nodo y
               …
               SI (NOT Solución(y) AND y.CS>=C AND y.CS==y.CI) ENTONCES
                  y=_SolAsignacionVoraz(y,B);
                  break; //terminé la búsqueda, ya no analizo los demás hermanos
               SI (Solución(y) AND (y.bact>s.bact)) ENTONCES
                  s := y.tupla
                  C := max(C,y.bact)
               SINO SI (NOT Solución(y) AND y.CS>C) ENTONCES
                  LNV := LNV + {y}
                  C := max(C, y.CI)
               FINSI
            FINSI
         FINPARA
      SINO SI (x.CS==C Y x.CS==x.CI) ENTONCES //nodo x seleccionado es solución voraz
         y=_SolAsignacionVoraz(y,B);
      FINSI
   FINMIENTRAS
```

- La función que calcula la solución voraz en el caso en que CI==CS es la siguiente:

```
NODO _SolAsignacionVoraz(NODO x, int B[][N])
    int Bmax // Para cada fila, beneficio máximo entre las tareas no usadas
    int tmax // Para cada nivel, tarea no usada de mayor beneficio

    PARA i = x.nivel+1..N-1 HACER              //PARA CADA nivel siguiente al actual
        //Busco en la fila i la tarea con beneficio máximo no usada
        Bmax := -1
        PARA j=0..N-1 HACER                    //Pruebo tareas j para nivel i
            SI (NOT x.usadas[j] && M[i][j] > Bmax)
                Bmax := M[i][j]   // Se guarda el beneficio asociado a la asignación
                tmax := j         // Se guarda tarea con beneficio máximo
            FINSI
        FINPARA
        // Actualizo x en nivel i con tarea tmax con beneficio Bmax
        x.tupla[i] := tmax      // Guardo tarea con Bmax para persona i
        x.usadas[tmax] := 1      // La tarea tmax fue usada
        x.bact := x.bact+Bmax    // Se actualiza el beneficio acumulado para el nodo
    FINPARA

    x.nivel := N - 1 // Necesario para que después pueda verificarse como solución
    DEVOLVER x       // Se devuelve el nodo hoja con la solución
```