Ejercicio de evaluación (ITIS). *Backtracking*

Coloreado de aristas

El coloreado de aristas de un grafo no dirigido consiste en asignar un color **a cada arista** de un grafo de tal forma que dos aristas del mismo color no tengan un vértice común.

Se dispone de C colores. Colorear una arista tiene un coste que depende del color elegido.

Diseña un algoritmo de *backtracking* para colorear con C colores las aristas de un grafo de N vértices y M aristas con coste mínimo. Detalla lo siguiente:

1. El árbol de búsqueda: significado de las aristas y de los niveles.
2. El código del procedimiento
3. El programa llamador

Solución ej. de evaluación (ITIS). *Backtracking*

- La solución puede estar formada por una tupla de elementos $\langle x_1, \ldots, x_M \rangle$.
- El elemento x_i de la solución contiene el color asignado a la arista i.
- El grafo puede venir representado mediante un *array* con los vértices de todas las aristas (matriz de $M \times 2$ elementos).
- Cada nivel del árbol corresponde a una arista en el grafo.
- Cada nodo del árbol tiene tantos descendientes como colores se pueden asignar.
- Las soluciones están en el nivel M, número de aristas del grafo.
- Restricciones explícitas: $1 \leq x_i \leq C$
- Restricciones implícitas: dos aristas con el mismo color no pueden tener un vértice común.

Solución ej. de evaluación (ITIS). *Backtracking* (cont.)

- Una posible implementación es la siguiente:

```
proc colores(A[1..M,1..2],c[1..C],solAct[1..M],cActIni,solOpt[1..M],cOpt,etapa)
  desde color ← 1 hasta C hacer
    solAct[etapa] ← color ; cAct ← cActIni + c[color]
    si cAct < cOpt ∧ aceptable(A,solAct,etapa) entonces
      si etapa = M entonces
        cOpt ← cAct ; solOpt ← solAct
      si no
        colores(A,c,solAct,cAct,solOpt,cOpt,etapa+1)
      fin si
    fin si
  fin desde
fin proc
```

Solución ej. de evaluación (ITIS). *Backtracking* (cont.)

```
fun aceptable(A[1..M,1..2],solAct[1..M],etapa)
   aceptable ← cierto ; i ← 1
   mientras i < etapa ∧ aceptable hacer
      si solAct[i] = solAct[etapa] ∧ (A[i,1]=A[etapa,1] ∨ A[i,1]=A[etapa,2] ∨
         A[i,2]=A[etapa,1] ∨ A[i,2]=A[etapa,2]) entonces
         aceptable ← falso
      fin si
      i ← i + 1
   fin mientras
   devolver aceptable
fin fun

fun llamador(A[1..M,1..2],c[1..C],sol[1..M])
   crear solAct[1..M]
   coste ← ∞
   colores(A,c,solAct[1..M],0,sol[1..M],coste,1)
   devolver coste
fin fun
```

Ejercicio de evaluación (ITIG). *Backtracking*

Recubrimiento de vértices

Un recubrimiento de vértices de un grafo es un subconjunto de vértices R cuyos elementos son adyacentes a todos los demás vértices del grafo: Por cada vértice v del grafo, existe una arista que lo une con algún vértice de R, o bien $v \in R$.

Diseña un algoritmo de *backtracking* que obtenga un recubrimiento de vértices de tamaño mínimo de un grafo. Detalla lo siguiente:

1. El árbol de búsqueda: significado de las aristas y de los niveles.
2. El código del procedimiento
3. El programa llamador

Solución ej. de evaluación (ITIG). *Backtracking*

- Como debemos proporcionar un subconjunto de los vértices, podemos utilizar una tupla de valores booleanos $\langle x_1, \ldots, x_n \rangle$, donde x_i tiene valor 1 si el vértice i pertenece al conjunto, o 0 en caso contrario.
- El grafo puede venir representado mediante su matriz de adyacencia.
- Cada nivel del árbol corresponde a un vértice del grafo.
- Cada nodo del árbol tiene dos descendientes.
- Las soluciones están en el nivel n, número de vértices del grafo.
- Restricciones explícitas: $x_i \in \{0, 1\}$
- Restricciones implícitas: no tiene
- La solución debe cumplir que todos los vértices del grafo son seleccionados o adyacentes a alguno de los vértices seleccionados.

Solución ej. de evaluación (ITIG). *Backtracking* (cont.)

```
proc cover_basico(A[1..N,1..N],solAct[1..N],nActIni,sol[1..N],nOpt,etapa)
  desde v ← 0 hasta 1 hacer
    solAct[etapa] ← v
    nAct ← nActIni + v
    si etapa = N entonces
      si nAct < nOpt ∧ todoCubierto(A,solAct) entonces nOpt ← nAct ; sol ← solAct
    si no cover_basico(A,solAct,nAct,sol,nOpt,etapa+1)
  fin desde
fin proc

fun todoCubierto(A[1..N,1..N],solAct[1..N])
  i ← 1 ; cubierto ← cierto
  mientras cubierto ∧ i ≤ N hacer
    si solAct[i] = 0 entonces
      j ← 1 ; cubierto ← falso
      mientras ¬cubierto ∧ j ≤ N hacer
        si A[i,j] ∧ solAct[j] = 1 entonces cubierto ← cierto
        j ← j + 1
      fin mientras
    fin si
    i ← i + 1
  fin mientras
  devolver cubierto
fin fun
```

Solución ej. de evaluación (ITIG). *Backtracking* (cont.)

- Se puede mejorar el algoritmo anterior:
  - guardando los vértices que han sido cubiertos en la solución parcial (`coverIni[1..N], nCoverIni`),
  - podando las ramas que no pueden mejorar la solución

```
proc cover(A[1..N,1..N],solAct[1..N],nActIni,sol[1..N],nOpt,coverIni[1..N],nCoverIni,etapa)
  desde v ← 0 hasta 1 hacer
    solAct[etapa] ← v ; nAct ← nActIni + v
    si nAct < nOpt entonces
      si v = 1 entonces
        nCover ← nCoverIni ; cover ← coverIni // se asigna el array completo
        si ¬cover[etapa] entonces cover[etapa] ← cierto ; nCover ← nCover + 1
        desde i ← 1 hasta N hacer
          si A[etapa,i] ∧ ¬cover[i] entonces cover[i] ← cierto ; nCover ← nCover + 1
        fin desde
        si nCover = N entonces nOpt ← nAct ; sol ← solAct
        si no si etapa < N entonces cover(A,solAct,nAct,sol,nOpt,cover,nCover,etapa+1)
      si no si etapa < N entonces
        cover(A,solAct,nAct,sol,nOpt,coverIni,nCoverIni,etapa+1)
      fin si
    fin si
  fin desde
  solAct[etapa] ← 0
fin proc
```

Solución ej. de evaluación (ITIG). *Backtracking* (cont.)

- Procedimiento llamador:

proc llamador(A[1..N,1..N],sol[1..N])
  crear solAct[1..N], coverIni[1..N]
  desde i $\leftarrow$ 1 **hasta** N **hacer**
    solAct[i] $\leftarrow$ 0 ; coverIni[i] $\leftarrow$ falso
  fin desde
  cover(A,solAct,0,sol,∞,coverIni,0,1)
fin proc