

Esquema general de vuelta atrás

```
1. proc vuelta-atrás(sol : tupla, e k : nat)
2.   preparar-recorrido-nivel(k)
3.   mientras ¬último-hijo-nivel(k) hacer
4.     sol[k] := siguiente-hijo-nivel(k)
5.     si es-solución?(sol, k) entonces
6.       tratar-solución(sol)
7.     si no
8.       si es-completable?(sol, k) entonces
9.         vuelta-atrás(sol, k + 1)
10.    fsi
11.    fsi
12.  fmientras
13. fproc
```

Cadenas de caracteres

```
1. proc variaciones-val(e  $n : nat^+$ ,  $sol[1..m]$  de  $1..n$ , e  $k : 1..m$ )
2.   para  $j = 1$  hasta  $n$  hacer
3.      $sol[k] := j$ 
4.     si no-repetida?( $sol, k$ ) entonces
5.       si  $k = m$  entonces imprimir( $sol$ ) { es una solución }
6.       si no variaciones-val( $n, sol, k + 1$ )
7.     fsi
8.   fsi
9.   fpara
10. fproc
```

```
1. fun no-repetida?( $sol[1..m]$  de  $nat, k : 1..n$ ) dev  $respuesta : bool$ 
2.    $i := 1$ 
3.   mientras  $sol[i] \neq sol[k]$  hacer
4.      $i := i + 1$ 
5.   fmientras
6.    $respuesta := (i = k)$ 
7. ffun
```

Vuelta atrás con marcaje

```
1. proc vuelta-atrás-marcaje(sol : tupla, e k : nat, m : marcador)
2.   preparar-recorrido-nivel(k)
3.   mientras ¬último-hijo-nivel(k) hacer
4.     sol[k] := siguiente-hijo-nivel(k)
5.     m := marcar(m, sol[k])
6.     si es-solución?(sol, k) entonces
7.       tratar-solución(sol)
8.     si no
9.       si es-completable?(sol, k, m) entonces
10.        vuelta-atrás-marcaje(sol, k + 1, m)
11.      fsi
12.    fsi
13.    m := desmarcar(m, sol[k])
14.  fmientras
15. fproc
```

Cadenas de caracteres con marcaje

$\forall i : 1 \leq i \leq n : usada[i] \Leftrightarrow i \text{ aparece en } sol[1..k].$

1. **proc** variaciones-va2(**e** $n : nat^+$, $sol[1..m]$ **de** $1..n$, **e** $k : 1..m$, $usada[1..n]$ **de** *bool*)
 2. **para** $j = 1$ **hasta** n **hacer**
 3. **si** $\neg usada[j]$ **entonces**
 4. $sol[k] := j$
 5. $usada[j] := \text{cierto}$ { marcar }
 6. **si** $k = m$ **entonces** imprimir(sol)
 7. **si no** variaciones-va2($n, sol, k + 1, usada$)
 8. **fsi**
 9. $usada[j] := \text{falso}$ { desmarcar }
 10. **fsi**
 11. **fpara**
 12. **fproc**
-
1. **proc** variaciones(**e** $n : nat^+$)
 2. **var** $sol[1..m]$ **de** $1..n$, $usada[1..n]$ **de** *bool*
 3. $usada[1..n] := [\text{falso}]$
 4. variaciones-va2($n, sol, 1, usada$)
 5. **fproc**

Coloreado de grafos: todas las soluciones

```
1. proc coloreado-grafo(e  $G : \text{grafo}[n]$ , e  $m : \text{nat}^+$ ,  $\text{sol}[1..n]$  de  $1..m$ , e  $k : 1..n$ )
2.   para  $\text{color} = 1$  hasta  $m$  hacer
3.      $\text{sol}[k] := \text{color}$ 
4.     si  $\text{test-color}(G, \text{sol}, k)$  entonces
5.       si  $k = n$  entonces  $\text{imprimir}(\text{sol})$ 
6.       si no  $\text{coloreado-grafo}(G, m, \text{sol}, k + 1)$ 
7.     fsi
8.   fsi
9.   fpara
10. fproc

1. fun  $\text{test-color}(G : \text{grafo}[n], \text{sol}[1..n]$  de  $1..m, k : 1..n)$  dev  $\text{respuesta} : \text{bool}$ 
2.    $\text{respuesta} := \text{cierto} ; j := 1$ 
3.   mientras  $j < k \wedge \text{respuesta}$  hacer
4.     si  $\text{está-arista?}(j, k, G) \vee \text{está-arista?}(k, j, G)$  entonces
5.        $\text{respuesta} := \text{sol}[j] \neq \text{sol}[k]$ 
6.     fsi
7.      $j := j + 1$ 
8.   fmientras
9. ffun
```

Coloreado de grafos: una solución

```
1. proc coloreado-grafo2(e  $G : \text{grafo}[n]$ , e  $m : \text{nat}^+$ ,  $\text{sol}[1..n]$  de  $1..m$ , e  $k : 1..n$ ,  
2.            $\text{encontrada} : \text{bool}$ )  
3.    $\text{color} := 1$   
4.   mientras  $\text{color} \leq m \wedge \neg \text{encontrada}$  hacer  
5.      $\text{sol}[k] := \text{color}$   
6.     si test-color( $G, \text{sol}, k$ ) entonces  
7.       si  $k = n$  entonces  $\text{encontrada} := \text{cierto}$   
8.       si no coloreado-grafo2( $G, m, \text{sol}, k + 1$ )  
9.     fsi  
10.  fsi  
11.   $\text{color} := \text{color} + 1$   
12.  fmientras  
13. fproc  
  
1. proc colorear(e  $G : \text{grafo}[n]$ , e  $m : \text{nat}^+$ ,  $\text{sol}[1..n]$  de  $1..m$ )  
2.    $\text{encontrada} := \text{falso}$   
3.   coloreado-grafo2( $G, m, \text{sol}, 1, \text{encontrada}$ )  
4.   si  $\text{encontrada}$  entonces imprimir( $\text{sol}$ )  
5.   si no imprimir(No se puede colorear)  
6.   fsi  
7. fproc
```

Suma de subconjuntos

```
1. proc conseguir-cantidad1(e  $P[1..n]$  de  $real^+$ , e  $C : real$ ,  $sol[1..n]$  de  $0..1$ , e  $k : 1..n$ ,
2.            $suma : real$ ,  $éxito : bool$ )
3.   { hijo izquierdo — añadir peso }
4.   si  $suma + P[k] \leq C$  entonces
5.      $sol[k] := 1$ 
6.      $suma := suma + P[k]$    { marcar }
7.     si  $suma = C$  entonces
8.        $éxito := cierto$  ;  $sol[k + 1..n] := [0]$ 
9.     si no
10.      si  $k < n \wedge suma < C$  entonces
11.        conseguir-cantidad1( $P, C, sol, k + 1, suma, éxito$ )
12.      fsi
13.    fsi
14.       $suma := suma - P[k]$    { desmarcar }
15.    fsi
16.    { hijo derecho — no añadir peso, no puede generar solución }
17.    si  $\neg éxito \wedge k < n$  entonces
18.       $sol[k] := 0$ 
19.      conseguir-cantidad1( $P, C, sol, k + 1, suma, éxito$ )
20.    fsi
21. fproc
```

1. $\{ resto = \sum_{i=k}^n P[i] \wedge P[1] \leq \dots \leq P[n] \}$
2. **proc** conseguir-cantidad2(**e** $P[1..n]$ **de** $real^+$, **e** $C : real$, $sol[1..n]$ **de** $0..1$, **e** $k : 1..n$,
3. $suma, resto : real, \acute{e}xito : bool$)
4. $resto := resto - P[k]$ { marcar común a los dos hijos }
5. $sol[k] := 1$ { hijo izquierdo }
6. $suma := suma + P[k]$ { marcar }
7. **si** $suma = C$ **entonces**
8. $\acute{e}xito := cierto ; sol[k + 1..n] := [0]$
9. **si no**
10. **si** $k < n \wedge suma + P[k + 1] \leq C$ **entonces**
11. conseguir-cantidad2($P, C, sol, k + 1, suma, resto, \acute{e}xito$)
12. **fsi**
13. **fsi**
14. $suma := suma - P[k]$ { desmarcar }
15. **si** $\neg \acute{e}xito$ **entonces**
16. $sol[k] := 0$ { hijo derecho }
17. **si** $k < n \wedge suma + resto \geq C \wedge suma + P[k + 1] \leq C$ **entonces**
18. conseguir-cantidad2($P, C, sol, k + 1, suma, resto, \acute{e}xito$)
19. **fsi**
20. **fsi**
21. $resto := resto + P[k]$ { desmarcar común }
22. **fproc**

```
1. fun suma-subconjunto( $P[1..n]$  de  $real^+$ ) dev  $\langle \acute{e}xito : bool, sol[1..n] \text{ de } 0..1 \rangle$ 
2.    $peso := 0$ 
3.   para  $i = 1$  hasta  $n$  hacer  $peso := peso + P[i]$  fpara    $\{ peso = \sum_{i=1}^n P[i] \}$ 
4.    $suma := 0 ; resto := peso$ 
5.    $\acute{e}xito := falso$ 
6.   conseguir-cantidad2( $P, peso/2, sol, 1, suma, resto, \acute{e}xito$ )
7. ffun
```

Estimación de la eficiencia: algoritmo de Monte Carlo

```
1. fun estimación(instancia del problema) dev num-nodos : nat+
2.   v := raíz del árbol para la instancia del problema
3.   num-nodos := 1
4.   m := 1
5.   mprod := 1   { número de nodos prometedores al nivel en que nos encontramos }
6.   mientras m ≠ 0 hacer
7.     t := número de hijos de v
8.     mprod := mprod * m
9.     num-nodos := num-nodos + mprod * t
10.    m := número de hijos prometedores de v
11.    si m ≠ 0 entonces
12.      v := hijo prometedor de v elegido aleatoriamente
13.    fsi
14.  fmientras
15. ffun
```

```

1. fun estimar-coloreado( $G : \text{grafo}[n]$ ,  $nc : \text{nat}^+$ ) dev  $\text{num-nodos} : \text{nat}^+$ 
2. var  $\text{sol}[1..n]$  de  $1..m, , \text{prometedores} : \text{conjunto}$ 
3.    $k := 0 ; \text{num-nodos} := 1 ; \text{mprod} := 1$ 
4.   mientras  $m \neq 0 \wedge k < n$  hacer
5.      $\text{mprod} := \text{mprod} * m$ 
6.      $\text{num-nodos} := \text{num-nodos} + \text{mprod} * nc$ 
7.      $k := k + 1$ 
8.     { cálculo del número de hijos prometedores }
9.      $m := 0$ 
10.     $\text{prometedores} := \text{cjto-vacío}()$ 
11.    para  $col = 1$  hasta  $nc$  hacer
12.       $\text{sol}[k] := col$ 
13.      si  $\text{test-color}(G, \text{sol}, k)$  entonces
14.         $m := m + 1$ 
15.         $\text{añadir}(col, \text{prometedores})$ 
16.      fsi
17.    fpara
18.      si  $m \neq 0$  entonces
19.         $col := \text{selección-aleatoria}(\text{prometedores})$ 
20.         $\text{sol}[k] := col$ 
21.      fsi
22.    fmientras
23.  ffun

```

Ciclos hamiltonianos

```
1. proc ciclo-hamiltoniano-va(e  $G : \text{grafo-val}[n]$ ,  $sol[1..n]$  de  $1..n$ , e  $k : 1..n$ ,  $usado[1..n]$  de  $bool$ )
2.   para  $vértice = 2$  hasta  $n$  hacer
3.     si  $\neg usado[vértice] \wedge \text{gv-está-arista?}(sol[k - 1], vértice, G)$  entonces
4.        $sol[k] := vértice$ 
5.        $usado[vértice] := \text{cierto}$    { marcar }
6.     si  $k = n$  entonces
7.       { falta comprobar que se cierra el ciclo }
8.       si  $\text{gv-está-arista?}(sol[n], 1, G)$  entonces imprimir( $sol$ ) fsi
9.     si no ciclo-hamiltoniano-va( $G, sol, k + 1, usado$ )
10.    fsi
11.     $usado[vértice] := \text{falso}$    { desmarcar }
12.  fsi
13.  fpara
14. fproc
```

```
1. proc ciclo-hamiltoniano(e  $G : \text{grafo-val}[n]$ )
2.  var  $sol[1..n]$  de  $1..n$ ,  $usado[1..n]$  de  $bool$ 
3.     $sol[1] := 1$ 
4.     $usado[1] := \text{cierto}$  ;  $usado[2..n] := [\text{falso}]$ 
5.    ciclo-hamiltoniano-va( $G, sol, 2, usado$ )
6.  ffun
```

Esquema de optimización

```
1. proc vuelta-atrás-optimización(sol : tupla, e k : nat, valor : valor,
2.                               sol-mejor : tupla, valor-mejor : valor)
3.   preparar-recorrido-nivel(k)
4.   mientras ¬último-hijo-nivel(k) hacer
5.     sol[k] := siguiente-hijo-nivel(k)
6.     valor := actualizar(valor, sol, k)
7.     si es-solución?(sol, k) entonces
8.       si mejor(valor, valor-mejor) entonces
9.         sol-mejor := sol ; valor-mejor := valor
10.      fsi
11.     si no
12.       si es-completable?(sol, k) ∧ es-prometedor?(sol, k, valor, valor-mejor) entonces
13.         vuelta-atrás-optimización(sol, k + 1, valor, sol-mejor, valor-mejor)
14.       fsi
15.     fsi
16.     valor := desactualizar(valor, sol, k)
17.   fmientras
18. fproc
```

Problema del viajante

```
1. proc vendedor-va(e  $G : \text{grafo-val}[n]$ , e  $\text{mín}G : \text{real}$ ,  $\text{sol}[1..n]$  de  $1..n$ , e  $k : 1..n$ ,  $\text{coste} : \text{real}$ ,
2.            $\text{usado}[1..n]$  de  $\text{bool}$ ,  $\text{sol-mejor}[1..n]$  de  $1..n$ ,  $\text{coste-mejor} : \text{real}_\infty$ )
3.    $\text{anterior} := \text{sol}[k - 1]$ 
4.   para  $\text{vértice} = 2$  hasta  $n$  hacer
5.     si  $\neg \text{usado}[\text{vértice}] \wedge \text{gv-está-arista?}(\text{anterior}, \text{vértice}, G)$  entonces
6.        $\text{sol}[k] := \text{vértice}$ 
7.        $\text{usado}[\text{vértice}] := \text{cierto} \quad \{ \text{marcar} \}$ 
8.        $\text{coste} := \text{coste} + \text{gv-valor}(\text{anterior}, \text{sol}[k], G)$ 
9.     si  $k = n$  entonces
10.      si  $\text{gv-está-arista?}(\text{sol}[n], 1, G) \wedge_c$ 
11.         $\text{coste} + \text{gv-valor}(\text{sol}[n], 1, G) < \text{coste-mejor}$  entonces
12.           $\text{sol-mejor} := \text{sol}$ 
13.           $\text{coste-mejor} := \text{coste} + \text{gv-valor}(\text{sol}[n], 1, G)$ 
14.      fsi
```

```
15.      si no      {  $k \neq n$  }
16.           $coste-estimado := coste + (n - k + 1) * mínG$ 
17.          si  $coste-estimado < coste-mejor$  entonces      { se puede mejorar  $sol-mejor$  }
18.               $vendedor-va(G, mínG, sol, k + 1, coste, usado, sol-mejor, coste-mejor)$ 
19.          fsi
20.      fsi
21.           $usado[vértice] := falso$       { desmarcar }
22.           $coste := coste - gv-valor(anterior, sol[k], G)$ 
23.      fsi
24.  fpara
25. fproc
```

Problema de la mochila

1. $\{ \frac{V[1]}{P[1]} \geq \frac{V[2]}{P[2]} \geq \dots \geq \frac{V[n]}{P[n]} \}$
2. **proc** mochila-va(**e** $P[1..n]$, $V[1..n]$ **de** $real^+$, **e** $M : real$, $sol[1..n]$ **de** $0..1$, **e** $k : 1..n$,
3. $peso, beneficio : real, sol-mejor[1..n]$ **de** $0..1, beneficio-mejor : real$)
4. { hijo izquierdo — coger objeto, no hacemos estimación }
5. $sol[k] := 1$
6. $peso := peso + P[k]; beneficio := beneficio + V[k]$ { marcar }
7. **si** $peso \leq M$ **entonces**
8. **si** $k = n$ **entonces**
9. $sol-mejor := sol; beneficio-mejor := beneficio$
10. **si no**
11. mochila-va($P, V, M, sol, k + 1, peso, beneficio, sol-mejor, beneficio-mejor$)
12. **fsi**
13. **fsi**
14. $peso := peso - P[k]; beneficio := beneficio - V[k]$ { desmarcar }

```
15. { hijo derecho — no coger objeto, no se marca pero sí se hace estimación }
16. sol[k] := 0
17. beneficio-estimado := cálculo-estimación(P, V, M, k, peso, beneficio)
18. si beneficio-estimado > beneficio-mejor entonces
19.     si k = n entonces
20.         sol-mejor := sol ; beneficio-mejor := beneficio
21.     si no
22.         mochila-va(P, V, M, sol, k + 1, peso, beneficio, sol-mejor, beneficio-mejor)
23.     fsi
24. fsi
25. fproc
```

```

1. {  $\frac{V[1]}{P[1]} \geq \frac{V[2]}{P[2]} \geq \dots \geq \frac{V[n]}{P[n]}$  }
2. fun cálculo-estimación( $P[1..n]$ ,  $V[1..n]$  de  $real^+$ ,  $M : real^+$ ,  $k : 1..n$ ,  $peso$ ,  $beneficio : real$ )
3.           dev  $estimación : real$ 
4.    $hueco := M - peso$  ;  $estimación := beneficio$ 
5.    $j := k + 1$ 
6.   mientras  $j \leq n \wedge P[j] \leq hueco$  hacer
7.     { podemos coger el objeto  $j$  entero }
8.      $hueco := hueco - P[j]$  ;  $estimación := estimación + V[j]$ 
9.      $j := j + 1$ 
10.  fmientras
11.  si  $j \leq n$  entonces { quedan objetos por probar }
12.    { fraccionamos el objeto  $j$  (solución voraz) }
13.     $estimación := estimación + (hueco/P[j]) * V[j]$ 
14.  fsi
15. ffun
1. fun mochila-principal( $P[1..n]$ ,  $V[1..n]$  de  $real^+$ ,  $M : real^+$ )
2.           dev  $\langle sol-mejor[1..n]$  de  $0..1$ ,  $beneficio-mejor : real \rangle$ 
3. var  $sol[1..n]$  de  $0..1$ 
4.    $peso := 0$  ;  $beneficio := 0$ 
5.    $beneficio-mejor := -1$    { peor que cualquier solución }
6.   mochila-va( $P$ ,  $V$ ,  $M$ ,  $sol$ ,  $1$ ,  $peso$ ,  $beneficio$ ,  $sol-mejor$ ,  $beneficio-mejor$ )
7. ffun

```