Se pretende implementar mediante programación dinámica iterativa la función recursiva:

```
unsigned f( unsigned x, unsigned v[] ) {
  if (x==0)
    return 0;
  unsigned m = 0;
  for ( unsigned k = 0; k < x; k++ )
     m = max( m, v[k] + f( x-k, v ) );
  return m;
}
```

¿Cuál es la mejor estructura para el almacén?

Seleccione una:

- a. `int A[][]` ✗
- b. `int A[]`
- c. `int A`

De los problemas siguientes, indicad cuál no se puede tratar eficientemente como los otros dos:

Seleccione una:

- a. El problema de cortar un tubo de forma que se obtenga el máximo beneficio posible.
- b. El problema del cambio, o sea, el de encontrar la manera de entregar una cantidad de dinero usando el mínimo de monedas posibles.
- c. El problema de la mochila sin fraccionamiento y sin restricciones en cuanto al dominio de los pesos de los objetos y de sus valores.

Se pretende implementar mediante programación dinámica iterativa la función recursiva:

```
unsigned f( unsigned x, unsigned v[] ) {
  if (x==0)
    return 0;
  unsigned m = 0;
  for ( unsigned k = 0; k < x; k++ )
     m = max( m, v[k] + f( x-k, v ) );
  return m;
}
```

¿Cuál es la mejor complejidad espacial que se puede conseguir?

Seleccione una:

- a. $O(x)$
- b. $O(x^2)$
- c. $O(1)$

El problema de encontrar el árbol de recubrimiento de coste mínimo para un grafo no dirigido, conexo y ponderado ...

Seleccione una:

- a. ... se puede resolver siempre con una estrategia voraz.
- b. ... no se puede resolver en general con una estrategia voraz.
- c. sólo se puede resolver con una estrategia voraz si existe una arista para cualquier par de vértices del grafo.

¿Cuál de los siguientes pares de problemas son equivalentes en cuanto al tipo de solución (óptima, factible, etc.) aportada por el método voraz?

Seleccione una:

- a. El fontanero diligente y el problema del cambio.
- b. El fontanero diligente y la mochila continua.
- c. El fontanero diligente y la asignación de tareas.

Se pretende implementar mediante programación dinámica iterativa la función recursiva:

```
unsigned f( unsigned x, unsigned v[] ) {
  if (x==0)
    return 0;
  unsigned m = 0;
  for ( unsigned k = 0; k < x; k++ )
     m = max( m, v[k] + f( x-k, v ) );
  return m;
}
```

¿Cuál es la mejor estructura para el almacén?

Seleccione una:

- a. `int A`
- b. `int A[][]`
- c. `int A[]`

De los problemas siguientes, indicad cuál no se puede tratar eficientemente como los otros dos:

Seleccione una:

- a. El problema de la mochila sin fraccionamiento y sin restricciones en cuanto al dominio de los pesos de los objetos y de sus valores. ✓
- b. El problema de cortar un tubo de forma que se obtenga el máximo beneficio posible.
- c. El problema del cambio, o sea, el de encontrar la manera de entregar una cantidad de dinero usando el mínimo de monedas posibles.

La respuesta correcta es: El problema de la mochila sin fraccionamiento y sin restricciones en cuanto al dominio de los pesos de los objetos y de sus valores.

Se pretende implementar mediante programación dinámica iterativa la función recursiva:

```
unsigned f( unsigned x, unsigned v[] ) {
  if (x==0)
    return 0;
  unsigned m = 0;
  for ( unsigned k = 0; k < x; k++ )
     m = max( m, v[k] + f( x-k, v ) );
  return m;
}
```

¿Cuál es la mejor complejidad espacial que se puede conseguir?

Seleccione una:

- a. $O(1)$
- b. $O(x)$
- c. $O(x^2)$

Se pretende implementar mediante programación dinámica iterativa la función recursiva:

```
unsigned f( unsigned y, unsigned x){  // suponemos y >= x
    if (x==0 || y==x) return 1;
    return f(y-1, x-1) + f(y-1, x);
}
```

¿Cuál es la mejor complejidad espacial que se puede conseguir?

Seleccione una:

- a. $O(y)$
- b. $O(y^2)$
- c. $O(1)$

La respuesta correcta es: $O(y)$

4. Si un problema de optimización lo es para una función que toma valores continuos ...

   (a) La programación dinámica iterativa siempre es mucho más eficiente que la programación dinámica iterativa en cuanto al uso de memoria.

   [(b)] La programación dinámica recursiva puede resultar mucho más eficiente que la programación dinámica iterativa en cuanto al uso de memoria.

   (c) El uso de memoria de la programación dinámica iterativa y de la programación dinámica recursiva es el mismo independientemente de si el dominio es discreto o continuo.

10. La mejor solución que se conoce para el problema de la mochila continua sigue el esquema ...

    [(a)] ...*divide y vencerás.*

    (b) ...*ramificación y poda.*

    (c) ...*voraz.*

17. Uno de estos tres problemas no tiene una solución eficiente que siga el esquema de programación dinámica

    (a) El problema de la mochila discreta.

    [(b)] El problema de las torres de Hanoi

    (c) El problema de cortar un tubo de longitud n en segmentos de longitud entera entre 1 y n de manera que se maximice el precio de acuerdo con una tabla que da el precio para cada longitud.

19. Decid cuál de estas tres es la cota pesimista más ajustada al valor óptimo de la mochila discreta:

    (a) El valor de la mochila continua correspondiente

    (b) El valor de una mochila que contiene todos los objetos aunque se pase del peso máximo permitido

    [(c)] El valor de la mochila discreta que se obtiene usando un algoritmo voraz basado en el valor específico de los objetos

38. ¿Cuál de estos problemas tiene una solución eficiente utilizando *programación dinámica*?

    (a) El problema de la asignación de tareas.

    [(b)] El problema del cambio.

    (c) La mochila discreta sin restricciones adicionales.

21. El siguiente programa resuelve el problema de cortar un tubo de longitud n en segmentos de longitud entera entre 1 y n de manera que se maximice el precio de acuerdo con una tabla que da el precio para cada longitud, pero falta un trozo. ¿Qué debería ir en lugar de XXXXXXX?

```
void fill(price r[]) {
for (index i=0;i<=n;i++) r[i]=-1;
}

price cutrod(price p[], r[], length n) {
price q;
if (r[n]>=0) return r[n];
if (n==0) q=0;
else {
       q=-1;
       for (index i=1;i<=n;i++)
       q=max(q,p[i]+cutrod(XXXXXXX));
}
r[n]=q;
return q;
}
```

(a) p, r-1, n

(b) p, r, n-r[n]

[(c)] p, r, n-i

26. Dadas las siguientes funciones:

```
// Precondición: { 0 <= i < v.size(); i < j <= v.size() }
unsigned f( const vector<unsigned>&v, unsigned i, unsigned j ) {
  if( i == j+1 )
    return v[i];
  unsigned sum = 0;
  for( unsigned k = 0; k < j - i; k++ )
    sum += f( v, i, i+k+1) + f( v, i+k+1, j );
  return sum;
}

unsigned g( const vector<unsigned>&v ) {
  return f( v, v.begin(), v.end() );
}
```

Se quiere reducir la complejidad temporal de la función `g` usando programación dinámica iterativa. ¿cuál sería la complejidad espacial?

(a) cúbica

[(b)] cuadrática

(c) exponencial

7. Uno de estos tres problemas no tiene una solución trivial y eficiente que siga el esquema voraz.

[(a)] El problema del cambio.

(b) El problema de la mochila discreta sin limitación en la carga máxima de la mochila.

(c) El problema de la mochila continua.

9. El siguiente programa resuelve el problema de cortar un tubo de longitud n en segmentos de longitud entera entre 1 y n de manera que se maximice el precio de acuerdo con una tabla que da el precio para cada longitud, pero falta un trozo. ¿Qué debería ir en lugar de **XXXXXXX**?

```
void fill(price m[]) {
for (index i=0;i<=n;i++) m[i]=-1;
}

price cutrod(length n, price m[], price p[]) {
price q;
if (m[n]>=0) return m[n];
if (n==0) q=0;
else {
        q=-1;
        for (index i=1;i<=n;i++)
            q=max(q,p[i]+cutrod(XXXXXXX));
}
m[n]=q;
return q;
}
```

(a) `n-m[n],m,p`

(b) `n-i,m,p`

(c) `n,m[n]-1,p`

26. Decid cuál de estas tres es la cota pesimista más ajustada al valor óptimo de la mochila discreta:

    (a) El valor de una mochila que contiene todos los objetos restantes aunque se pase del peso máximo permitido.

    (b) El valor de la mochila continua correspondiente.

    (c) El valor de la mochila discreta que se obtiene usando un algoritmo voraz basado en el valor específico de los objetos.

27. ¿Cuál de estos problemas tiene una solución eficiente utilizando *programación dinámica*?

    (a) El problema del cambio.

    (b) El problema de la asignación de tareas.

    (c) La mochila discreta sin restricciones adicionales.