Un circuito ✖ hamiltoniano ✖ es un camino que pasa por cada arista una y solo una vez. Un camino ✖ euleriano ✖ es un camino cerrado que recorre cada arista exactamente una vez, teniendo como inicio y fin el mismo vértice.

camino cerrado | euleriano | grafo | hamiltoniano | trayecto

Respuesta incorrecta.

La respuesta correcta es:
Un **[camino] [euleriano]**es un camino que pasa por cada arista una y solo una vez. Un [circuito] [euleriano] es un camino cerrado que recorre cada arista exactamente una vez, teniendo como inicio y fin el mismo vértice.

Determinar si un grafo puede ser coloreado con 2 colores, está en P, pero con tres colores es NP-completo, incluso cuando se restringe a los grafos planos.

Seleccione una:

- Verdadero ✔
- Falso

La respuesta correcta es 'Verdadero'

El algoritmo mergesort es siempre más rápido que el insertion sort.

Seleccione una:

- Verdadero
- Falso ✔

En conjuntos pequeñps, ejemplo menos de 63 elementos, el insert es más rápido.

La respuesta correcta es 'Falso'

La complejidad del algoritmo de búsqueda binaria, utilizando el paradigma divide y vencerás, tiene como complejidad $\theta(logn)$

Seleccione una:

- Verdadero ✖
- Falso

Es O(logn) theta es cuado siempre presenta esa función, pero en el mejor caso el elemento se encuentra al medio del vector y es el primer elemento en ser comparado.

La respuesta correcta es 'Falso'

Si un grafo posee un circuito Euleriano, entonces cada vértice del grado tiene grado par, a excepción de un vértice.

Seleccione una:

- Verdadero
- Falso ✔

es par siempre

La respuesta correcta es 'Falso'

Si hay algún algoritmo polinomial para resolver un problema NP-hard, entonces hay algoritmos para resolver todos los problemas de NP en tiempo polinomial, esto significaría que P=NP.

Seleccione una:

- Verdadero ✔
- Falso

La respuesta correcta es 'Verdadero'

Se puede utilizar el algoirtmo BFS para encontrar el camino más corto entre dos vértices

Seleccione una:

- Verdadero
- Falso ✖

La respuesta correcta es 'Verdadero'

Un grafo es conexo si es posible caminar desde cualquier vértice a cualquier otro vértice a través de una secuencia de aristas adyacente.

Seleccione una:

- (•) Verdadero ✔
- ( ) Falso

La respuesta correcta es 'Verdadero'

La multiplicación de la matriz de la cadena (o cadena matriz problema de ordenación, MCOP) es un problema de optimización que puede resolverse utilizando programación dinámica . Dada una secuencia de matrices, el objetivo es encontrar la manera más eficiente para multiplicar estas matrices . El problema no es en realidad para realizar las multiplicaciones, sino simplemente para decidir la secuencia de las multiplicaciones de matrices involucradas.

Hay muchas opciones, ya que la multiplicación de matrices es asociativa . En otras palabras, no importa cómo es el producto entre paréntesis , el resultado obtenido seguirá siendo el mismo. Por ejemplo, para cuatro matrices A , B , C , y D , tendríamos:

((AB) C) D = (A (BC)) D = (AB) (CD) = A ((BC) D) = A (B (CD)).

Sin embargo, el orden en que el producto está entre paréntesis afecta al número de operaciones aritméticas sencillas necesarios para calcular el producto, que es la complejidad computacional . Por ejemplo, si A es una matriz de 10 × 30, B es una matriz de 30 x 5, y C es una matriz de 5 x 60, entonces

informática (AB) C necesita (10 × 30 × 5) + (10 x 5 x 60) = 1500 + 3000 = 4500 operaciones, mientras

computar A (BC) necesita (30 x 5 x 60) + (10 x 30 x 60) = 9000 + 18000 = 27000 operaciones.

El problema también puede ser revisado en los slides de Loureiro en la parte de programación dinámica. Allí ya esta el algoritmo para encontrar el mínimo numero de operaciones, no obstante falta la parte de encontrar el orden correcto.

Dada la siguiente entrada

```
// ENTRADA PROPUESTA
        int p[7];

        p[0]=30;
        p[1]=35;
        p[2]=15;
        p[3]=5;
        p[4]=10;
        p[5]=20;
        p[6]=25;
```

La multiplicación de la matriz de la cadena (o cadena matriz problema de ordenación, MCOP) es un problema de optimización que puede

resolverse utilizando programación dinámica . Dada una secuencia de matrices, el objetivo es encontrar la manera más eficiente para multiplicar estas matrices . El problema no es en realidad para realizar las multiplicaciones, sino simplemente para decidir la secuencia de las multiplicaciones de matrices involucradas.

Hay muchas opciones, ya que la multiplicación de matrices es asociativa . En otras palabras, no importa cómo es el producto entre paréntesis , el resultado obtenido seguirá siendo el mismo. Por ejemplo, para cuatro matrices A , B , C , y D , tendríamos:

> ((AB) C) D = (A (BC)) D = (AB) (CD) = A ((BC) D) = A (B (CD)).

Sin embargo, el orden en que el producto está entre paréntesis afecta al número de operaciones aritméticas sencillas necesarios para calcular el producto, que es la complejidad computacional . Por ejemplo, si A es una matriz de 10 × 30, B es una matriz de 30 x 5, y C es una matriz de 5 x 60, entonces

> informática (AB) C necesita (10 × 30 × 5) + (10 x 5 x 60) = 1500 + 3000 = 4500 operaciones, mientras computar A (BC) necesita (30 x 5 x 60) + (10 x 30 x 60) = 9000 + 18000 = 27000 operaciones.

El problema también puede ser revisado en los slides de Loureiro en la parte de programación dinámica. Allí ya esta el algoritmo para encontrar el mínimo numero de operaciones, no obstante falta la parte de encontrar el orden correcto.

Dada la siguiente entrada

```
// ENTRADA PROPUESTA
        int p[7];

        p[0]=30;
        p[1]=35;
        p[2]=15;
        p[3]=5;
        p[4]=10;
        p[5]=20;
        p[6]=25;
```

- Codifique en c++ la solución al problema, no sólo dando como respuesta el número mínimo de operaciones si no el orden. **ESTE EJERCICIO PUNTUA SOLO SI LA SOLCUIÓN, ES DECIR EL ORDEN DE LA MULTIPLICACIÓN ES MOSTRADA EN CONSOLA.** Imaginemos 3 matrices entonces la presentación
 será algo así (AB) C, donde evidentemente primero es a y b luego el

resultado de eso se multiplica con C, adicionalmente muestre la matriz de programación dinámica utilizada.

- Explique el algoritmo

- En un archivo pdf suba el código con la explicación aparte, no como comentario, así como screenshots del algoritmo con la entrada propuesta mostrados en consola. La parte de los screenshots son importantes así tenga la explicación sin screenshot del algoritmo corriendo lo demás no vale.

SOLUCION

```
/******************************************************************************

                            Online C++ Compiler.
                Code, Compile, Run and Debug C++ program online.
Write your code in this editor and press "Run" button to compile and
execute it.

*******************************************************************************/

#include <iostream>

using namespace std;

#include <stdlib.h>
#include <stdio.h>

/*
 * according to introduction to algorithm,
 * we alloc array(n+1)but make use of the last n
 * just for convinence
 */
void MATRIX_CHAIN_ORDER(int *p,int n,int **m,int **s)
{
   int i;
   int row = n+1;
   //initialize our array
   for(i=1;i<=n;i++)
      *((int*)m+row*i+i) = 0;
```

```
   int l;
   for(l=2;l<=n;l++)
   {
      for(i=1;i<=(n-l+1);i++)
      {
         int j = i+l-1;
         *((int*)m+row*i+j) = -1;
         int k;
         for(k=i;k<=(j-1);k++)
         {
            int tmp1 = *((int*)m+row*i+k);
            int tmp2 = *((int*)m+row*(k+1)+j);
            int q = tmp1+tmp2+p[i-1]*p[k]*p[j];

            int old = *((int*)m+row*i+j);
            if(q<old || old == -1)
            {
               *((int*)m+row*i+j) = q;
               *((int*)s+row*i+j) = k;
            }
         }
      }
   }

}

void PRINT_OPTIMAL_PARRNS(int **s,int i,int j,int row)
{
   if(i==j)
      printf("A%d",i);
   else{
      printf("(");
      PRINT_OPTIMAL_PARRNS(s,i,*((int*)s+row*i+j),row);
      PRINT_OPTIMAL_PARRNS(s,*((int*)s+row*i+j)+1,j,row);
      printf(")");
   }
}
```

```
int main()
{
   int n = 6;
   int p[7];
   p[0]=30;
   p[1]=35;
   p[2]=15;
   p[3]=5;
   p[4]=10;
   p[5]=20;
   p[6]=25;

   int m[n+1][n+1];
   int s[n+1][n+1];

   MATRIX_CHAIN_ORDER(p,n,(int**)m,(int**)s);
   int i,j;
   for(i=1;i<=n;i++)
   {
      for(j=i;j<=n;j++)
      {
         printf("%d  ",m[i][j]);
      }
      printf("\n");
   }

   PRINT_OPTIMAL_PARRNS((int**)s,1,6,7);

   return 0;
}
```