Ejercicio 2: La conexión de “El Pedregal”.

Descripción:

- En un archipiélago, con multitud de pequeñas islas cercanas, hay puentes que unen ciertos pares de islas entre sí. Para cada puente (que puede ser de dirección única), además de saber la isla de origen y la isla de destino, se conoce su anchura (número entero mayor que 0).
- La anchura de un camino, formado por una sucesión de puentes, es la anchura mínima de las anchuras de todos los puentes que lo forman.
- Para cada par de islas se desea saber cuál es el camino de anchura máxima que las une (siempre que exista alguno).

Este ejercicio cumple las condiciones para poder solucionarse utilizando programación dinámica ya que cumple las siguientes características.

Naturaleza n-etápica:

Representaremos las diferentes ciudades y sus comunicaciones como un grafo *G* donde las ciudades serían los nodos/vértices y los puentes las aristas que unen los distintos nodos.

Para encontrar el camino más ancho desde un vértice *i* a otro *j* en un grafo *G,* veríamos que vértice debe ser el segundo, cual el tercero, etc. hasta alcanzar el *j*.

Una sucesión optimal de decisiones proporcionará entonces el camino de anchura máxima.

Por lo tanto, este problema tiene una clara naturaleza n-etápica.

Verificación del POB:

Sea $i, i_1, \ldots, i_k, j$ el camino con máxima anchura desde i hasta j, comenzando con el vértice inicial *i*, se ha tomado decisión de ir al vértice i_1.

Como resultado, ahora el estado del problema está definido por el vértice i_1, y lo que se necesita es encontrar un camino desde i_1 hasta *j*.

Está claro que la sucesión $i, i_1, i_2, \ldots, i_k, j$ debe constituir un camino con anchura máxima entre i_1 y *j.* Si no:

- Sea $i_1, r_1, \ldots, r_q, j$ un camino más corto entre i_1 y *j*, entonces $i, i_1, r_1, r_2, \ldots r_q, j$ es un camino entre *i* y *j* que es más corto que el camino $i, i_1, i_2, i_3, \ldots, i_k, j$.
- Como eso **es una contradicción, se verifica el POB** y también puede aplicarse a este problema.

Planteamiento de una recurrencia:

Denominaremos A(i, j) la anchura del puente que va de la isla *i* a la isla *j*. Si no hay puente en esa dirección entonces A(i, j) = ∞. Obsérvese que conviene A(i, i) = 0 puesto que para ir de una isla a ella misma no debe existir ninguna restricción.

Definimos, de manera recursiva, la siguiente función:

- Max A(i,j,k) = máxima anchura de los caminos que van desde la isla *i* hasta la isla *j* , pudiendo pasar por las islas {1, …, k}.
- Max A(i,j,0) = A(i,j).
- Max A(i,j,k) = Max{Max A(i,j,k-1), Min{Max A(i,k,k-1), Max A(k,j,k-1)}} si k >0.

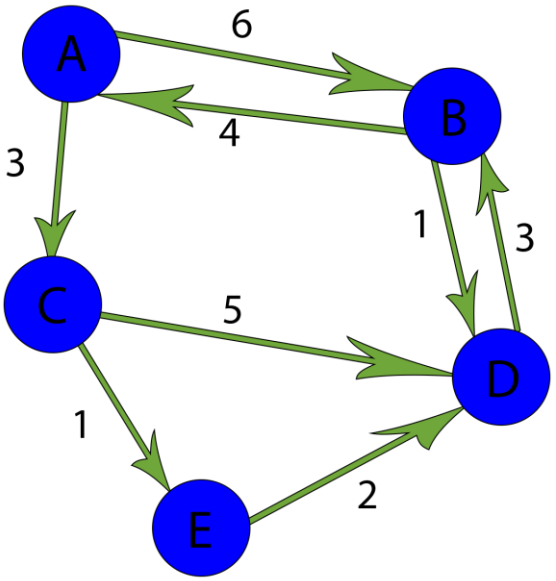
Así podemos diseñar un algoritmo como el siguiente, que calcula los valores de una tabla M(1..n, 1..n), de manera que al terminar: M(i,j) = Max A(i,j,n) ∀i, j.

```
proc MÁXIMA_ANCHURA(A, M)
  M(1..n, 1..n) ← A(1..n, 1..n)
  for k in 1..n loop
    for i in 1..n loop
      for j in 1..n loop
        if M(i,k) < M(k,j) then aux ← M(i,k)
        else aux ← M(k,j)
        if M(i,j) < aux then M(i,j) ← aux
```

Implementación C++:

```
#define N 5

int main(int argc, char const *argv[]) {

  int M[N][N];
  int aux;

  inicializarMatriz(M);

  for(int k = 0; k < N; k++){
    for(int i = 0; i < N; i++){
      for(int j = 0; j < N; j++){
        aux = (M[i][k] < M[k][j])?M[i][k]:M[k][j];
        if((M[i][j] < aux) && (i != j)) M[i][j] = aux;
      }
    }
  }

  for(int i = 0; i < N; i++){
    for(int j = 0; j < N; j++){
      cout << M[i][j] << " ";
    }
    cout << endl;
  }

  return 0;
}
```

Caso de ejemplo (enfoque adelantado):



Dado este grafo como ejemplo, vamos a ver el resultado que devuelve el programa para poder verificar los resultados.

Matriz Inicial:

	A	B	C	D	E
A	0	6	3	∞	∞
B	4	0	∞	1	∞
C	∞	∞	0	5	1
D	∞	3	∞	0	∞
E	∞	∞	∞	2	0

Resultado programa:

	A	B	C	D	E
A	0	6	3	3	1
B	4	0	3	3	1
C	3	3	0	5	1
D	3	3	3	0	1
E	2	2	2	2	0

Podemos observar en el resultado que da lo esperado y efectivamente calcula los caminos más anchos entre todos los posibles caminos.