Ejercicio 4: El problema del turista en Manhattan

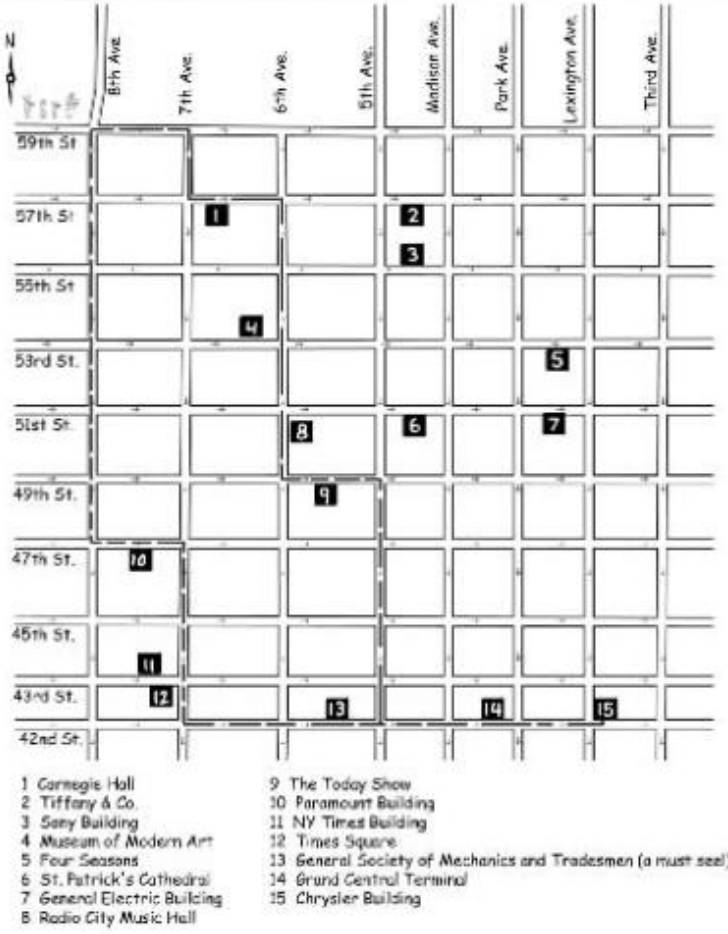
Descripción:

En la isla de Manhattan, cierto turista quiere aprovechar al máximo sus vacaciones visitando el mayor número de lugares turísticos, pero en dicho recorrido se encuentra con varias restricciones que lo limitan:

-Comienza en la parte noroeste del mapa (esquina superior izquierda).

-Solo se puede desplazar hacia el sur y hacia el este

Con estos requisitos podemos deducir que el final de su recorrido se haya en la parte más al sureste del mapa (esquina inferior derecha) y una vez en este punto el número de lugares visitados tiene que ser máximo.



Naturaleza n-etápica:

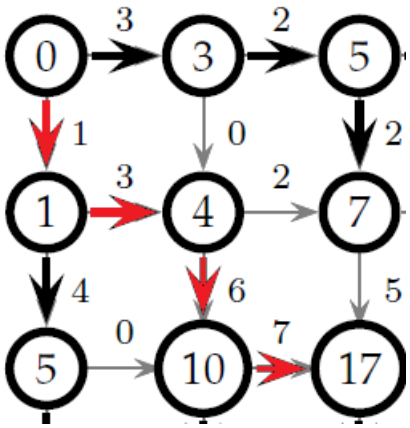
Para este problema el mapa de la ciudad se representará como un grafo donde cada arista representará una calle y cada vértice una intersección de calles. Cada una de las aristas tendrá un valor asociado que indicará el número de sitios turísticos que hay en esa calle. Cada vértice tendrá a su vez otro valor asociado que indicará el número de sitios turísticos visitados hasta el momento priorizando la ruta que maximice el número de estos.

En este problema se puede observar una clara naturaleza n-etápica. Ya que el cálculo de cada nuevo vértice debe hacerse una vez hallados sus predecesores. No se puede calcular un vértice aislado ni una solución sin antes haber pasado por cada una de las etapas previas para la construcción del grafo.

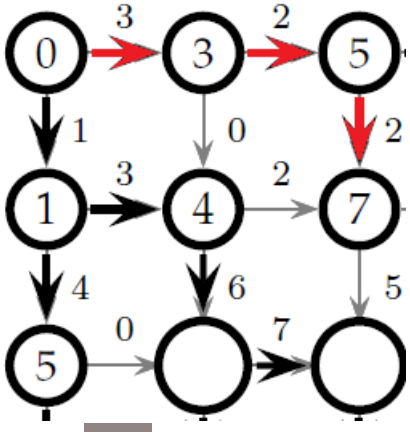
Verificación del POB:

El principio de optimalidad de Bellman se verifica si toda solución óptima a un problema está compuesta por soluciones óptimas de sus subproblemas. En este problema es fácil comprobar que no se cumple dicho principio con un ejemplo simple.

Tenemos un grafo 3X3 donde hemos hallado la solución óptima que es {0, 1, 4, 10, 17}



Pero si nos remontamos a una etapa anterior donde se estaba resolviendo un subproblema del problema actual podemos veremos que la solución cambia siendo {0, 3, 5, 7}



Por ello podemos decir que **este problema no verifica el POB**.

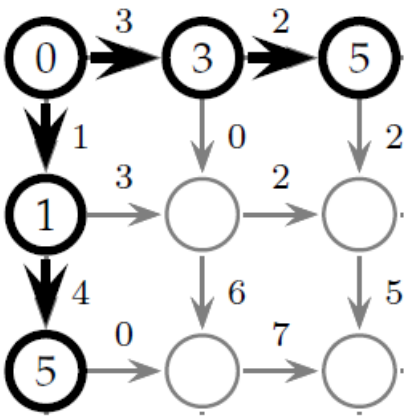
Planteamiento de una recurrencia:

En una primera fase del algoritmo se calcula los vértices en vertical y en horizontal del grafo. Después aplicamos la ecuación recurrente.

Llamaremos $S_{i,j}$ a un vértice que se haya en la posición i,j

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{Valor de la arista entre } (i-1,j) \text{ y } (i,j) \\ s_{i,j-1} + \text{Valor de la arista entre } (i,j-1) \text{ y } (i,j) \end{cases}$$

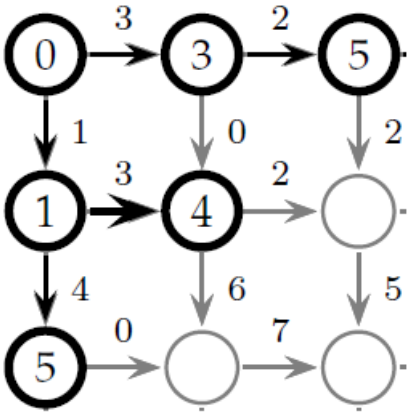
Para probar la ecuación podemos usar un ejemplo que determine el valor que tener el $S_{1,1}$ en el siguiente grafo (con la horizontal y vertical previamente calculada)



Utilizando la formula anterior tenemos

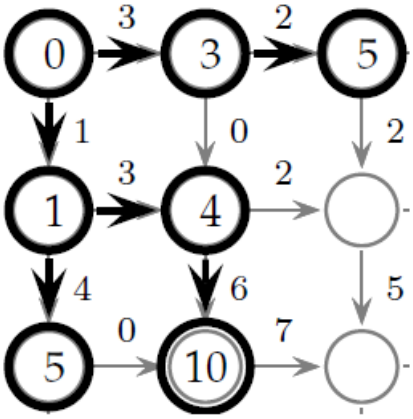
Etapa 1

$S_{1,1}$ = MAX((3+0) y (3+1))



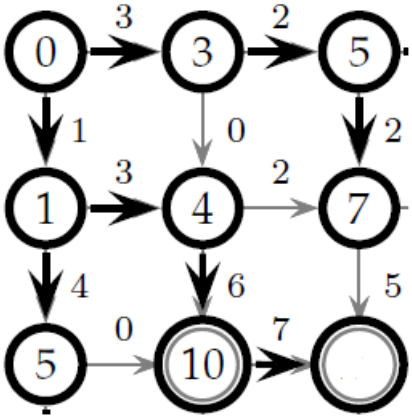
Etapa 2

$S_{2,1}$ = MAX((5+0) y (4+6))



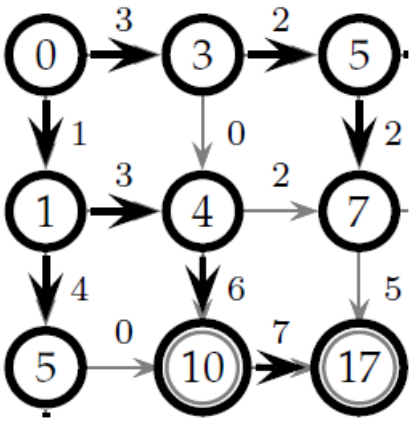
Etapa 3

$S_{1,2}$ = MAX((4+2) y (5+2))



Etapa 4

$S_{2,2}$ = MAX((10+7) y (7+5))



Código

Pseudocódigo

MANHATTANTOURIST($\downarrow w, \rightarrow w, n, m$)

```
1  s_{0,0} ← 0
2  for i ← 1 to n                                   } n
3      s_{i,0} ← s_{i-1,0} + ↓w_{i,0}
4  for j ← 1 to m                                   } m
5      s_{0,j} ← s_{0,j-1} + →w_{0,j}
6  for i ← 1 to n                                   } n*m
7      for j ← 1 to m
8          s_{i,j} ← max { s_{i-1,j} + ↓w_{i,j}
                         { s_{i,j-1} + →w_{i,j}
9  return s_{n,m}
```

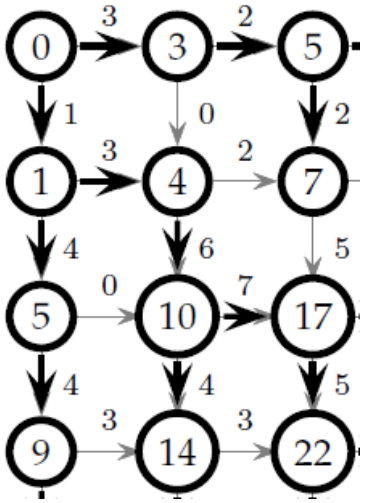
O(nm)

Implementación en Java

```
private static void turista(Vertice[][] ve, Arista[] a, int n, int m) {

    for (int i = 1; i < n; i++) {
        usadaI = devuelveArista(a, ve[0][i - 1], ve[0][i]);
        ve[0][i].setValor(ve[0][i - 1].getValor() + usadaI.getValor());
    }
    for (int j = 1; j < m; j++) {
        usadaJ = devuelveArista(a, ve[j - 1][0], ve[j][0]);
        ve[j][0].setValor(ve[j - 1][0].getValor() + usadaJ.getValor());
    }

    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {

            usadaI = devuelveArista(a, ve[j][i - 1], ve[j][i]);
            usadaJ = devuelveArista(a, ve[j - 1][i], ve[j][i]);
            if ((ve[j][i - 1].getValor() + usadaI.getValor()) > (ve[j - 1][i].getValor() + usadaJ.getValor())) {
                ve[j][i].setValor(ve[j][i - 1].getValor() + usadaI.getValor());
                anadeSolucion(ve[j][i], usadaI);

            } else {
                ve[j][i].setValor(ve[j - 1][i].getValor() + usadaJ.getValor());
               anadeSolucion(ve[j][i], usadaJ);
            }
        }
    }
}
```

Salida

Para el siguiente grafo



Esta sería la salida de nuestro algoritmo

```
Salida - TuristaManhattan (run)
run:
Desde 1,0 a 1,1 con valor 3 ---> Total: 4
Desde 1,1 a 2,1 con valor 6 ---> Total: 10
Desde 2,1 a 2,2 con valor 7 ---> Total: 17
Desde 2,2 a 3,2 con valor 5 ---> Total: 22
BUILD SUCCESSFUL (total time: 0 seconds)
```