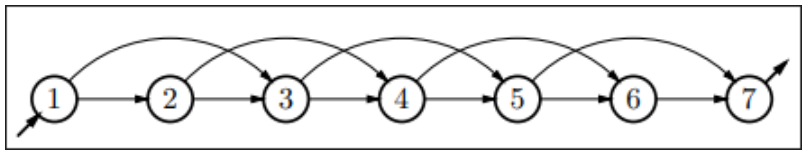
1. Embarcaderos

A lo largo de un río hay 7 embarcaderos conectados de la manera en que se muestra en la imagen. Se quiere obtener el recorrido *óptimo* entre el primer y el último embarcadero, siendo este recorrido el *de menor coste*. Se harán varias implementaciones.



Aproximación inicial

Se puede pensar que lo mejor sería llegar al embarcadero que menor coste tenga desde el embarcadero actual, pero esto *no siempre consigue la mejor solución al problema ya que hay rutas que no se consideran*

Aproximación recursiva

Se puede resolver este problema de manera correcta si lo resolvemos de forma recursiva mediante programación dinámica. Primero plantearemos la ecuación recursiva del problema.

Primero necesitamos determinar el **espacio de soluciones del problema**

Será toda secuencia de embarcaderos que cumpla lo siguiente:

- Empieza por 1 y acaba por el último embarcadero E
- Ninguno de los embarcaderos de la secuencia presenta saltos de más de 2 embarcaderos.

$$X = \{(e_1, e_2, \ldots, e_n) \in [1..E]^+ \mid e_1 = 1; \quad e_n = E; \quad 1 \leq e_i - e_{i-1} \leq 2, \quad 1 < i \leq n\}.$$

A continuación necesitamos definir una función objetivo que, en este caso **minimizar** para obtener la solución deseada. Usaremos una función C que calcule el coste del camino entre 2 embarcaderos.

$$(\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_n) = \arg \min_{(e_1, e_2, \ldots, e_n) \in X} C((e_1, e_2, \ldots, e_n)).$$

$$\min_{(e_1, e_2, \ldots, e_n) \in X} C((e_1, e_2, \ldots, e_n)).$$

Vemos que buscamos tanto el camino de menor coste como el menor coste asociado a dicho camino.

ECUACIÓN RECURSIVA

Necesitamos ahora buscar una expresión recursiva que defina el problema a resolver. Utilizaremos una expresión general que obtiene el coste mínimo posible entre 2 embarcaderos y demostraremos cómo se puede "desglosar" en llamadas recursivas de talla menor para obtener una solución válida al problema.

$$\min_{(e_1,e_2,\ldots,e_n)\in X} \sum_{1<i\leq n} c(e_{i-1},e_i) = \min_{\substack{(e_1,e_2,\ldots,e_n)|e_1=1;e_n=E;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n}} \sum_{1<i\leq n} c(e_{i-1},e_i).$$

Partiendo de esta expresión, y mediante la propiedad de los sumatorios en la que puedes extraer el último elemento y sumárselo fuera del mismo, extenderemos la expresión a lo siguiente:

$$\min_{\substack{(e_1,e_2,\ldots,e_n)|e_1=1;e_n=E;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n}} \sum_{1<i\leq n} c(e_{i-1},e_i) = \min_{\substack{(e_1,e_2,\ldots,e_{n-1})|e_1=1;1\leq E-e_{n-1}\leq 2;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n-1}} \left(\left(\sum_{1<i\leq n-1} c(e_{i-1},e_i)\right)+c(e_{n-1},E)\right).$$

Como sabemos que el penúltimo embarcadero solo puede ser E - 1 o E - 2, podemos describir esta expresión de la manera siguiente:

$$\min_{\substack{(e_1,e_2,\ldots,e_n)|e_1=1;e_n=E;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n}} \sum_{1<i\leq n} c(e_{i-1},e_i) = \min_{j\in\{E-1,E-2\}} \left(\min_{\substack{(e_1,e_2,\ldots,e_{n-1})|e_1=1;e_{n-1}=j;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n-1}} \left(\left(\sum_{1<i\leq n-1} c(e_{i-1},e_i)\right)+c(j,E)\right)\right).$$

Utilizando ahora la propiedad del mínimo donde el mínimo de una suma a+b es igual a el mínimo de a + b... volvemos a reformular para obtener...

$$\min_{j\in\{E-1,E-2\}} \left(\boxed{\min_{\substack{(e_1,e_2,\ldots,e_{n-1})|e_1=1;e_{n-1}=j;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n-1}} \left(\left(\sum_{1<i\leq n-1} c(e_{i-1},e_i)\right)+c(j,E)\right)}\right) = \min_{j\in\{E-1,E-2\}} \left(\left(\min_{\substack{(e_1,e_2,\ldots,e_{n-1})|e_1=1;e_{n-1}=j;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n-1}} \sum_{1<i\leq n-1} c(e_{i-1},e_i)\right)+c(j,E)\right).$$

En la siguiente imagen podremos ver que tanto la parte izquierda de la igualdad como la parte interior de la derecha son muy similares. Solo se diferencian en que la parte izquierda es una expresión **general** para ir del embarcadero 1 al embarcadero E mientras que la parte derecha representa el coste hasta llegar al antepenúltimo o último embarcadero. Llegados a este punto podemos ver la recursión en la fórmula.

$$\min_{\substack{(e_1,e_2,\ldots,e_n)|e_1=1;e_n=E;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n}} \sum_{1<i\leq n} c(e_{i-1},e_i) = \min_{j\in\{E-1,E-2\}} \left(\left(\min_{\substack{(e_1,e_2,\ldots,e_{n-1})|e_1=1;e_{n-1}=j;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n-1}} \sum_{1<i\leq n-1} c(e_{i-1},e_i)\right)+c(j,E)\right).$$

Si abusamos de la notación para definir una función C(j) que calcula el coste desde el primer embarcadero hasta el embarcadero j...

$$C(j) = \min_{\substack{(e_1,e_2,\ldots,e_n)|e_1=1;e_n=j;\\ 1\leq e_i-e_{i-1}\leq 2,1<i\leq n}} \sum_{1<i\leq n} c(e_{i-1},e_i).$$

Podremos obtener el coste para C(E) de la manera siguiente...

$$C(E) = \min\left\{\begin{array}{l} C(E-2)+c(E-2,E),\\ C(E-1)+c(E-1,E)\end{array}\right\}.$$

El razonamiento para c(E) aplica también para cualquier otro embarcadero. Si queremos calcular el coste del camino más barato entre el primer embarcadero y un embarcadero i cualquiera simplemente cambiamos esa E por una i.

Con esta fórmula ya tenemos la base de nuestra recursión. Ahora sólo necesitamos considerar nuestros casos base:

- Si la recursión alcanza el embarcadero 1, hemos de devolver 0 ya que es un caso especial de embarcadero donde no hay ningún coste porque es desde el que se empieza.
- Si la recursión alcanza el embarcadero 2 se debe añadir el coste que se obitene de viajar del embarcadero 1 al 2, ya que es el único caso posible.

$$C(i) = \begin{cases} 0, & \text{si } i = 1; \\ c(1,2), & \text{si } i = 2; \\ \text{mín}(C(i-2) + c(i-2,i), C(i-1) + c(i-1,i)), & \text{si } i > 2, \end{cases} \tag{8.1}$$

Veamos ahora la implementación de este método:

In [6]:
```
def recursive_cheapest_price(E, c):
  def C(i):
    if i == 1: return 0
    elif i == 2: return c(1,2)
    else: return min(C(i-1)+c(i-1,i), C(i-2)+c(i-2,i))
  return C(E)
```

Ahora veremos un ejemplo de ejecución...

In [8]:
```
def c(i, j):
  edges = {
    (1, 2): 4,
    (1, 3): 6,
    (2, 3): 3,
    (2, 4): 7,
    (3, 4): 1,
    (3, 5): 4,
    (4, 5): 2,
    (4, 6): 7,
    (5, 6): 4,
    (5, 7): 5,
    (6, 7): 9
  }
  return edges.get((i, j), edges.get((j, i), float('inf')))

print(recursive_cheapest_price(7, c))
```

14

Se puede hacer también una **versión del método que almacene todos los cálculos de los embarcaderos visitados para así no tener que volverlos a calcular**. Esto ahorra tiempo y espacio ya que **convierte un problema exponencial en uno de orden de E (númer de embarcaderos)**

In [ ]:
```
def memo_recursive_cheapest_price(E, c):
  R = {}
  def C(i):
    if i == 1: R[1] = 0
    elif i == 2: R[2] = c(1,2)
    else:
      if i-2 not in R: C(i-2)
      if i-1 not in R: C(i-1)
      R[i] = min(R[i-1]+c(i-1,i), R[i-2]+c(i-2,i))
    return R[i]
  return C(E)
```

Aproximación iterativa

El algoritmo recursivo paga un sobrecoste por llamadas recursivas. Es posible eliminarlo si hacemos nuestro problema iterativo. Con ello, hemos de analizar y resolver en orden las dependencias del problema, desde el embarcadero 1 hasta el i.

In [9]:
```
def iterative_cheapest_price(E, c):
  C = {}
  C[1] = 0
  C[2] = c(1,2)
  for i in range(3, E+1):
    C[i] = min(C[i-1] + c(i-1,i), C[i-2] + c(i-2,i))
  return C[E]
```

In [10]:
```
print(iterative_cheapest_price(7, c))
```

```
14
```