

nodes : 0, 1, 2, 3, 4

best path: $0 \rightarrow 4 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 0 \dots$

path = [4, 2, 0, 1, 3]

Letture soluzione:

Start: 0 next = path[0] = 4
↓
4 next = path[4] = 3
↓
3 next = path[3] = 1
↓
1 next = path[1] = 2
↓
2 next = path[2] = 0
↓
0

Trovo nodo + vicino:

Costruisco la matrice min-edges:

↳ Precomputed

0	<div></div>
1	
⋮	
i	
nnodes-1	

$\rightarrow K_{i,1} = \arg\min_j \{C_{i,j}\}$, $K_{i,1}$ il 2° più piccolo, ecc...

$K_{i,1} = i \quad \forall i$ visto che il nodo + vicino a 'i' è 'i' \rightarrow posso eliminare la prima colonna

Trovo soluzione: path = [-1, -1, -1, -1, -1]

\rightarrow start: 0 nodo + vicino: 4
↓
path = [4, -1, -1, -1, -1]

\rightarrow node = 4 nodo + vicino: 3
↓
path = [4, -1, -1, -1, 3]

\rightarrow node = 3 nodo + vicino: 1
↓
path = [4, -1, -1, 1, 3]

Vantaggi: se path[i] == -1, allora
il nodo i non è stato visitato
↓
risparmio memoria rispetto ad avere
una lista che conta se il nodo i
è stato visitato

Vantaggi: `find_min()` trova il nodo più vicino a `i`, quindi mi basta leggere il primo elemento della riga `i` $\Rightarrow O(1)$

Se il primo elemento è già nel path guardo il successivo, e così via.

Non devo mai leggere tutta la riga dato che spostandomi i nodi più vicini

restano diversi

Sono costretto a leggere tutta la riga solo all'ultima iterazione di un caso limite.

Con questo metodo esploro pochi nodi, contro $O(\Omega(n))$ se devo trovare il minimo ogni volta

Test: su un'istanza di 500 nodi c'è un guadagno
di 6 secondi (da 53 sec a 47 sec \Rightarrow miglioramento dell'11%)