DOCUMENTAZIONE ESERCIZIO 3: EMERGENCY CALL

L’esercizio 3 richiede l’implementazione della funzione emergency\_call(G, pos, v, k) che, preso in input un grafo diretto e connesso G che rappresenta la rete stradale cittadina (i vertici sono gli incroci e gli archi sono le strade), un dizionario pos che contiene la posizione delle volanti, il luogo v dove intervenire ed il numero k di volanti richieste per l’intervento, individua le volanti che devono essere allertate.

Il problema è stato risolto implementando una versione modificata dell’algoritmo di Dijkstra, chiamata sul nodo v di interesse, in cui non si punta a conoscere i cammini più brevi tra il nodo v e tutti gli altri nodi del grafo G, ma si va alla ricerca delle k volanti più vicine. In questo caso, quindi, si ricercano le volanti nei nodi che hanno, ad ogni passo, il costo più basso per raggiungere il nodo v.

Affinché l’algoritmo possa essere eseguito nel modo corretto, emergeny\_call esegue alcune operazioni per preparare i dati necessari alla risoluzione del problema. Ad esempio, in input viene fornito il dizionario pos, che ha per chiave la volante e per valore l’incrocio in cui si trova. Tuttavia, risulta maggiormente di interesse avere un dizionario che abbia per chiave l’incrocio e come valore la lista delle volanti presenti in quell’incrocio. Inoltre, come per l’algoritmo originale di Dijkstra, è importante avere una AdaptableHeapPriorityQueue che mantenga le informazioni relative ai nodi e al costo da pagare per essere raggiunti. Si usa tale struttura dati perché rende efficiente la restituzione del minimo oltre che la modifica delle chiavi, ovvero dei costi associati ai nodi e che permette di ordinarli in ordine crescente. L’inserimento ha costo pari a O(log n), la modifica pari a O(log n) e la rimozione del minimo pari a O(log n).

Fatte queste operazioni preliminari, si entra in un ciclo che terminerà o nel caso in cui la PriorityQueue è vuota, e nel caso k sia diverso da 0 viene lanciata un’eccezione, o quando saranno state trovate le k volanti richieste. Ad ogni iterazione, si estrae dalla coda il nodo che ha il costo minimo, si aggiungono le eventuali volanti presenti in tale nodo alla soluzione e si decrementa k. Se questo arriva a 0, la soluzione, che consiste in una lista di k volanti, viene restituita. Inoltre, ad ogni iterazione vengono aggiornate le etichette di ciascun vertice non ancora parte della soluzione che è raggiungibile da quello appena considerato, proprio come nell’algoritmo originale.

La soluzione proposta, date le strutture dati utilizzate, presenta una complessità temporale pari a O(A + m log n), dove A è il numero di volanti, m è il numero di archi e n è il numero di vertici. Questo valore è ottenuto considerando che all’algoritmo di Dijkstra è stato aggiunto un ciclo che viene ripetuto un numero di volte pari al numero di volanti in circolazione. Inoltre, vi è anche la ricerca e l’aggiunta delle volanti alla soluzione, che richiede al più un costo pari a O(k), per cui non influisce all’interno del ciclo perché è sicuramente di costo inferiore al ciclo in cui vengono rilassati gli archi.