GIUGNO 2019

WP3.2 Integrazione di algoritmi di routing optmization all'in-door navigation per il calcolo dinamico

D3.2 Report del processo di integrazione della indoor navigation routine relativa la ricalcolo dinamico del percorso.

SOFTWARE

REPORT DEL PROCESSO DI INTEGRAZIONE DELLA INDOOR NAVIGATION ROUTINE RELATIVA LA RICALCOLO DINAMICO DEL PERCORSO.

PB

Sintesi:

Questo sudio è stato effettuato per integrare l'algoritmo di navigazione indoor, con l'architettura software a supporto. Disegnato ed analizzato l'algoritmo per il calcolo del percorso minimo, questo è stato poi codificato per una applicazione mobile. L'applicazione mobile è stata disegnata in modo che possa autonomamente dare indicazioni sul percorso ottimale da effettuare, per raggiungere il luogo stabilito, anche in assenza di connettività. Inoltre qualora la connettività con l'applicazione del coordinatore delle operazioni fosse stabilita, l'interazione tra le due applicazioni consentirebbe il ricalcolo dinamico del percorso tenendo conto delle nuove disposizioni impartite.

Algoritmo percorso ottimale e ricalcolo dinamico

Per poter rappresentare in maniera ottimale la mappa, si è deciso di utilizzare un grafo non orientato e pesato. Il Grafo in questione G=(V,E) è una tupla di due insiemi V ed E, in cui V ha come elementi dei vertici contenenti informazioni riguardo posizione ed altri aspetti dei beacon usati per implementare la funzionalità di tracciamento indoor, mentre E contiene al suo interno coppie di tali vertici che rappresentano corridoi o, in generale, dei collegamenti da un punto ad un altro. Come precedentemente accennato, il grafo in questione contiene al suo interno anche informazioni riguardanti il "peso" di ogni percorso (n.b. sequenza di archi) poiché mediante questi sarà possibile applicare in maniera ottimale l'algoritmo di ricalcolo dei percorso migliore implementato per l'applicativo.

Soffermiamo ora la nostra attenzione sull'applicazione mobile e sul suo funzionamento al primo avvio.

L'app all'avvio preleva dal server il grafo dei percorsi percorribili per permetterle di funzionare anche in locale, senza bisogno di essere connessi alla rete wifi. Essa è in grado di individuare l'utente all'interno del piano e, inoltre, in base al punto di interesse che l'utente intende raggiungere, ne calcola il percorso minimo (partendo dalla posizione attuale) avvalendosi del grafo precedentemente caricato.

Il vertice più vicino rispetto alla posizione reale viene stabilito in base ai segnali (trasmessi dai beacon) ricevuti. La ricezione di questi segnali avviene tramite il protocollo EddyStone-UID, implementato attraverso delle librerie android.

Supponiamo di avere un grafo con n vertici contraddistinti da numeri interi {1,2,...,n} e che uno di questi nodi sia quello di partenza e un altro quello di destinazione. Il peso sull'arco che congiunge i nodi j e k lo indichiamo con *p(j,k)*. A ogni nodo, al termine dell'analisi, devono essere associate due etichette, *f(i)* che indica il peso totale del cammino (la somma dei pesi sugli archi percorsi per arrivare al nodo i-esimo) e *J(i)* che indica il nodo che precede *i* nel cammino minimo. Inoltre definiamo due insiemi *S* e *T* che contengono rispettivamente i nodi a cui sono già state assegnate le etichette e quelli ancora da scandire.

1. *Inizializzazione*.
    - Poniamo *S*={1}, *T*={2,3,...,n}, f(1)=0, J(1)=0.
    - Poniamo f(i)=p(1,i), J(i)=1 per tutti i nodi adiacenti ad 1.
    - Poniamo f(i)= ∞, per tutti gli altri nodi.
2. *Assegnazione etichetta permanente*
    - Se f(i)= ∞ per ogni i in T **STOP**
    - Troviamo j in T tale che f(j)=min f(i) con i appartenente a T
    - Poniamo T=T{j} e S=S∪{j}
    - Se T=Ø **STOP**
3. *Assegnazione etichetta provvisoria*
    - Per ogni i in T, adiacente a j e tale che f(i)>f(j)+p(j,i) poniamo:
        - f(i)=f(j)+p(j,i)

- J(i)=j

- Andiamo al passo 2

La complessità computazionale dell'algoritmo può essere espressa in funzione della cardinalità dei due insiemi V ed E ossia, rispettivamente, il numero di nodi e degli archi appartenenti al grafo sul quale viene eseguito. L'algoritmo utilizza una coda a priorità su cui vengono effettuate tre operazioni: la costruzione della coda, l'estrazione dell'elemento minimo e la riduzione del valore di un elemento. La struttura dati utilizzata per l'implementazione della coda di priorità determina la complessità di queste tre operazioni e, di conseguenza, quella dell'algoritmo.

Di seguito sono riportate le complessità delle operazioni in coda e dell'algoritmo nel caso in cui le code di priorità siano implementate tramite array, heap binarie o heap di Fibonacci.

	Costruire la coda	Estrarre il minimo	Ridurre un valore	Algoritmo
Arrays	$\Theta(\|V\|)$	$O(\|V\|)$	$\Theta(1)$	$O(\|V\|^2 + \|E\|)$
Heap binarie	$\Theta(\|V\|)$	$O(\log_2 \|V\|)$	$O(\log_2 \|V\|)$	$O((\|V\| + \|E\|) \log_2 \|V\|)$
Heap di Fibonacci	$\Theta(\|V\|)$	$O(\log_2 \|V\|)$	$\Theta(1)$	$O(\|V\| \log_2 \|V\| + \|E\|)$

È interessante notare che, nel caso in cui il grafo **non** sia sufficientemente sparso la soluzione basata sugli array è più efficiente di quella implementata tramite le heap binarie.

Mentre per quanto riguarda il calcolo del cammino minimo dal vertice sorgente alla destinazione selezionata, viene ricavato in base all'algoritmo di Dijkstra applicato al grafo dei percorsi percorribili del piano in cui si trova l'utente.

Per l'implementazione si è deciso di utilizzare l'heap binario in modo da velocizzare le operazioni di ricerca e calcolo del percorso minimo.

Il ricalcolo del percorso minimo, qualora venisse segnalato ad esempio che un corridoio non risulta più percorribile, si avvale della conoscenza del grafo aggiornata ricavata dal messaggio inviato dal responsabile, che segnala eventuali cambiamenti/ malfunzionamenti tramite un ulteriore applicativo web che funge da pannello di controllo.

Di seguito sono riportati degli screenshot che mostrano l'interfaccia dell'applicazione web dell'utente finale. (Figura 1 e Figura 2).

Figura 1: Schermata mappa.

Figura 2: Schermata calcolo percorso minimo.