Relazione sul Progetto del corso di Algoritmi

Gnocchi Marco

12 giugno 2019

Sommario

Di seguito si espone brevemente il problema la cui modellizzazione e soluzione costituiva il progetto d'esame, seguito da una descrizione degli algoritmi e strutture dati impiegati per portare a termine tali richieste, insieme ad un calcolo della complessità temporale e spaziale ad essi relativi.

Indice

1 Il Problema

Il quesito consisteva nella modellizzazione e simulazione di una rete di trasporto automatico con forti semplificazioni. Dopo aver acquisito informazioni variabili su la flotta di auto a guida autonoma a disposizione, la struttura di una rete stradale simulata, e le chiamate di una serie di clienti era necessario selezionare le vetture più adatte a evadere le singole richieste, tenendo traccia della loro autonomia, assicurandone eventualmente la ricarica, delle distanze percorse da ognuna, e del guadagno relativo alle corse. Prima della gestione della simulazione era richiesto di disporre i veicoli opportunamente sulla rete stradale, in modo da massimizzare la distanza tra essi, oltre che una breve elaborazione dei dati dei clienti.

Più nello specifico per poter andare a trattare questo problema, era necessario trovare modalità efficaci di modellizzare la rete stradale in modo che il calcolo dei cammini minimi tra un punto ed un altro fosse il più efficente possibile; stabilire regole di confronto che permettessero di selezionare, di volta in volta, il veicolo più adatto, facendo riferimento a delle specifiche fornite dal problema; e anche stabilire un metodo efficente per gestire il susseguirsi degli eventi discreti, consistenti in chiamate, ma anche corse e ricariche delle autovetture, e la loro elaborazione ordinata.

1.1 Dettagli aggiuntivi

Procedendo con maggior ordine e dettaglio, ciò che era necessario fare è quanto segue:

- elaborare oppurtunamente i parametri del problema.
- ordinare i clienti in ordine alfabetico e stamparne il nome. Ciò è fatto nell'assunzione che ogni cliente abbia un nome diverso ed effettui una sola chiamata.
- trovare durata e percorsi minimi di ognuno dei viaggi richiesti da ogni cliente e ordinarli per durata decrescente.
- stabilire la posizione iniziale delle vetture in modo che la prima sia in sede, ed ogni auto successiva venga posta nel punto della rete che massimizza la somma delle distanze minime dalle auto già fissate.
- gestire la simulazione vera e propria e stamparne i risultati:
  - Una chiamata ha un tempo minimo e massimo in cui il viaggio richiesto possono essere eseguiti: essere in grado di mandare una vettura nel punto di partenza del cliente entro il tempo minimo garantisce un bonus ai guadagni. Non è consentito terminare una corsa dopo il tempo massimo.

  - Ad ogni chiamata si sceglie l'auto in grado di terminare per prima il servizio richiesto. Se più auto terminerebbero il servizio allo stesso orario tra queste si sceglie quella che arriva dopo nel punto di partenza richiesto dal cliente, in modo da minimizzare i tempi in cui la vettura attende ferma. Se più vetture realizzano la parità anche secondo questo criterio, poichè ad ogni vettura è associato un numero univoco, si selezionerà quella di numero più basso.
  - Un auto è considerata disponibile per la corsa solo se in grado di terminarla entro il tempo massimo, se al termine avrebbe ancora carica sufficente a rientrare in sede, e se non è occupata in altre mansioni.
  - una vettura può arrivare al punto di partenza richiesto da un cliente in qualunque momento, ma non può iniziare il viaggio prima del tempo minimo. Tale tempo minimo rappresenta l'orario in cui il cliente è presente dove comunicato per il ritiro.
  - Se nessuna vettura è libera, si considera rifiutata la chiamata; è poi richiesto il numero totali di chiamate rifiutate.
  - Al termine di una corsa, se una vettura ha meno del 20% di carica, essa deve essere mandata a ricaricarsi.
  - Il tempo di ricarica di una vettura è fisso e non dipende dall'autonomia residua. Inoltre solo una vettura per volta può ricaricarsi: le altre attendono in coda. Si tiene conto del numero di ricariche.
  - La distanza totale percorsa dalle vetture durante servizio o rientro in sede per la ricarica deve essere misurata.
  - Il ricavo da una corsa è considerato uguale al tempo necessario per compiere il tragitto, più l'eventuale bonus, che è specifico della richiesta ed è comunicato al momento della chiamata. Si deve tener traccia dei ricavi totali.
  - La gestione degli eventi segue una gerarichia, in modo che, in linea di massima, i veicoli vengano liberati prima che impegnati: in questo modo, ad esempio, una vettura appena caricata è considerata libera per una chiamata giunta nel momento del termine della ricarica. L'ordine di priorità è: termine ricariche, inizio ricariche, fine corsa di una vettura e, solo in fine, l'arrivo di una chiamata.

- Compiere una stima per eccesso del guadagno che, a fronte di un orizzonte temporale di servizio limitato, scelga le chiamate di maggior valore e le serva con precedenza rispetto a quelle che porterebbero ad un minor guadagno. Qui si è fatta una grossa approssimazione fingendo che i veicoli possano prestarsi tempo a vicenda e che non abbiano bisogno di muoversi sulla rete per prendere in carico una chiamata, ma solo per evaderla.

2 La modellizzazione e le scelte

2.1 I punti centrali del modello

In questa sezione si pone l'accento su i sottoproblemi di maggiore importanza, su quali approcci sono stati scelti per modellizzarli e risolverli e su quali ragioni mi abbiano portato a selezionare tali approcci. Viene in fine indicata anche la complessità temporale e spaziale degli algoritmi implementati.

2.1.1 Trovare i cammini minimi per muoversi in Città

Il problema fondante e centrale dell'elaborazione era la gestione efficace del calcolo di minime distanze su una rete stradale. Essendo questo l'unico problema direttamente dipendente dalla modellizzazione della rete, si è scelto di modellizzare quest'ultima come un grafo, tramite l'uso di liste di adiacenza, e di impiegare l'algoritmo di Dijkstra per calcolare quelle distanze. Questo approccio è stato preferito perchè computazionalmente minimo tra le possibilità note: L'algoritmo di Dijkstra ha infatti molto da guadagnare dal più rapido accesso alla lista dei nodi adiacenti ad un nodo dato garantita dall'Implementazione per liste di adiacenza, rispetto a quella data dalle matrici di adiacenza, ed essendo implementato sfruttando gli Heap per l'identificazione del nodo più vicino a quelli già raggiunti, ha una complessità di $O(m \log n)$dove m è il numero di archi e n di vertici. Nella risoluzione del problema sono state Tuttavia impegate due diverse versioni di tale algoritmo: una che tenesse traccia dei predecessori, ricostruendo in calce il cammino compiuto, ed una versione che se ne dimenticasse. Nel primo caso alla complessità già dichiarata si aggiunge quella necessaria a ricomporre il cammino che è $O(n)$ nel caso peggiore. Osserviamo infine che in una rete stradale è piuttosto plausibile supporre che m sia più vicino a n che a n^2, suo limite superiore, poichè difficilmente esiste una strada che collega in modo diretto quasi ogni coppia di punti sulla rete. In entrambi i casi la complessità spaziale è $\Theta(n)$, con quella del secondo algoritmo maggiore della prima di circa una costante, dipendente dall'implementazione, moltiplicata per $2n$.

Si evidenzia che nell'implementazione di Dijkstra che tiene presente del percorso compiuto, si è utilizzato l'algoritmo partendo dalla destinazione invece che dal punto di partenza, in modo che il vettore dei predecessori, di cui l'algoritmo tiene naturalmente conto divenisse vettore dei successori. Ciò è stato fatto per agevolare la ricostruzione del percorso senza conoscerne a priori la lunghezza. Tuttavia la falsa assunzione che il cammino minimo fosse unico, fatta prima di implementare in tal modo l'algoritmo, potrebbe causare differenze minime nei risultati, senza però aumentare la durata del percorso trovato. I percorsi alternativi così trovati non sono, dal punto di vista algoritmico, errati, in quanto sono comunque un elemento dell'inisieme delle soluzioni del problema, nell'istanza considerata.

2.1.2 La selezione dei Taxi

Altra questione particolarmente fondamentale per la risoluzione del problema è quella della scelta delle vetture: Per queste si è scelta una semplice implementazione in cui una struttura conteneva tutte le informazioni comuni ad ogni auto, oltre che un array con puntatori a strutture che rappresentassero le singole vetture, dotate di tutte le informazioni specifiche della singola istanza.

Per implementare questa selezione si sono fatte alcune considerazioni su come i criteri che erano proposti dalle specifiche risultassero equivalenti a criteri sul tempo di attesa della vettura nei confronti del cliente. Si è dato come tempo di idle (o di attesa della vettura) la differenza tra l'orario di arrivo del cliente nel punto di ritiro e l'orario di arrivo del taxi nel medesimo punto. Si osserva che, se vi sono vetture con tempo di idle positivo o nullo, esse realizzano necessariamente il medesimo tempo di arrivo a destinazione, che è per altro sempre inferiore al tempo di arrivo a destinazione di una qualunque vettura con tempo di idle negativo. La stessa osservazione vale anche per il premio di puntualità: una vettura è in grado di realizzarlo se e solo se ha tempo di idle non negativo. Usando delle funzioni di gestione degli heap che tenessero automaticamente in conto del terzo criterio (ovvero ordinassero le vetture con indice minore prima di quelle di indice maggiore, a parità di tempo di idle), si sono costruiti uno heap al minimo per le vetture di idle positivo, ed uno al massimo per quelle ad idle negativo. Inoltre si è verificata la disponibilità di vetture ad idle positivo prima che di vetture ad idle negativo. Questo approccio ha permesso di mantenere un'equivalenza logica tra i criteri richiesti e quelli impiegati, senza dover analizzare sempre tutte le vetture. La complessità temporale dell'algoritmo ottenuto è di $O(a^2 + am \log n)$ nel caso peggiore, mentre ha sempre una complessità spaziale dell'ordine di $\Theta(a + n)$, dato che i due heap sono sempre composti, e al limite subito distrutti.

2.1.3 Disporre i veicoli sulla rete

Altra problematica chiave per il successo della simulazione era quella legata al corretto posizionamento delle vetture. Per realizzare quanto richiesto in questo caso si è deciso di costruire uno heap al massimo, i cui singoli elementi avessero un indice, identico a quello del nodo del grafo ad essi collegato, e un peso (su cui è stato applicato il criterio di heap) che venisse di volta in volta aggiornato, partendo da 0, e sommandovi di volta in volta la distanza del nodo rappresentato dall'ultimo nodo scelto. Il primo nodo è stato scelto arbitrariamente come quello di indice 1, rappresentante la sede operativa della società di trasporti, mentre i successivi erano il massimo dello heap, estratto di volta in volta. Il procedimento è stato ripetuto di volta in volta, ricomponendo da capo lo heap dopo ogni aggiornamento dei pesi, in quanto

pesantemente modificati. Una stima per eccesso della complessità temporale è data da $O(anm \log n)$ poichè aggiorna uno heap di n elementi a volte, richiamando Dijkstra ogni volta. Lo spazio necessario è invece dell'ordine di $\Theta(n)$

2.1.4 L'elaborazione degli eventi discreti

Per simulare gli avvenimenti descritti dal problema si è sfruttata una simulazione ad eventi discreti, ovvero si è costruita una lista di oggetti dotati di una relazione d'ordine totale basata sull'orario e sul tipo di evento rappresentato (oltre che, in casi estremi, il numero del veicolo associato, se presente) che venisse elaborata dalla testa alla coda, in cui ogni elaborazione poteva eventualmente generare a sua volta eventi. La buona posizione del modello è garantita dalla non esistenza di chiamate con orario uguale (cosa che non renderebbe più totale la relazione d'ordine fornita nelle specifiche) e dall'impossibilità di espandere all'infinito la lista di eventi, cosa garantita dal limite alla tipologia di tipi di evento generati e generabili. Per essere precisi perdere quest'ultima proprietà del problema non renderebbe mal posto il problema, quanto più non computabile in tempo finito il modello.

Per gestire una lista di eventi così implementata sono state necessarie funzioni in grado di scorrere la lista ed inserire nuovi eventi in punti qualsiasi, funzioni che permettessero di rimuovere l'elemento in testa, e in fine delle condizioni che permettessero di riconoscere il termine della coda: garantite dalla modalità di inizializzazione ed inserimento degli eventi.

3 Note

Si fa presente che i calcoli della complessità ignorano le chiamate a funzioni di allocazione e deallocazione, che sono, in pessima approssimazione, considerate costanti, per uniformarle a quanto supposto per la dichiarazione statica delle variabili. Questa semplificazione è dovuta al fatto che funzioni simili hanno complessità non conoscibile a priori, che esula dall'algoritmo in cui sono inserite, poichè fanno riferimento a strutture su cui la procedura chiamante non ha controllo, e non sono note a priori oppure non sono garantite dallo standard, avendo dunque complessità dipendente dalla struttura della memoria con cui lavorano. Per motivi simili si è supposta la complessità delle funzioni di lettura (fscanf()) e scrittura (printf()) come proporzionale linearmente alla dimensione dell'input: non sono infatti pubblicate delle complessità per queste funzioni insieme alla libreria standard.

Inoltre si segnala come per il calcolo della complessità spaziale si è cercato di determinare l'ordine di grandezza del massimo spazio usato contemporaneamente, in quanto è frequente che dello spazio impiegato da un'istruzione sia liberato da un'altra. Calcolare lo spazio totale di utilizzo conteggiando

come necessariamente distinto ogni bit usato in ogni dato momento dell'esecuzione sarebbe stato semplicemente troppo distante dalla realtà, in quanto avrebbe fortemente sovrastimato il quantitativo di memoria necessario all'esecuzione degli algoritmi impiegati.

Si aggiunge anche che era mia intenzione suddividere ulteriormente la libreria fornita in base alle strutture impiegate, tuttavia ciò non mi è stato possibile senza ulteriori conoscenze riguardo alla gestione di libreria con cross-referencing: argomento comunque non coperto dal corso, ed esterno ai suoi obiettivi.