Esercizio 4, prova scritta del 14 luglio 2017

Un rappresentante deve visitare alcuni clienti che si trovano in diverse città. A tale scopo deve determinare il percorso più breve che permetta di visitare ciascuna città e tornare, infine, alla città di partenza.

Supponendo che le città siano $c_1, c_2, \ldots, c_n$, dove c_1 è la città di partenza, indichiamo con $d_{i,j}$ la lunghezza del percorso più breve per andare dalla città c_i alla città c_j, $i, j = 1, \ldots, n$. Si vuole trovare una permutazione $(\pi_1, \pi_2, \ldots, \pi_n)$ di $(1, 2, \ldots, n)$, che fornisca la sequenza in cui visitare le città in modo che il percorso totale risulti minino.

Nota: Chiamiamo *distanza percorsa con la permutazione* $(\pi_1, \pi_2, \ldots, \pi_n)$ la somma delle distanze percorse per passare da una città all'altra, secondo l'ordine dato dalla permutazione, tornando infine alla città iniziale. Formalmente può essere espressa come $\sum_{i=1}^{n-1} d_{\pi_i, \pi_{i+1}} + d_{\pi_n, \pi_1}$. Il problema è dunque quello di trovare una permutazione con distanza percorsa minima.

Esempio: Supponete $n = 4$, con le distanze d_{ij} date dalla matrice riportata a destra. La distanza percorsa con la permutazione $(1, 3, 2, 4)$ è $40 + 50 + 40 + 42 = 172$, mentre quella percorsa con $(1, 4, 2, 3)$ è $44 + 26 + 50 + 39 = 159$.

$$\begin{pmatrix} 0 & 20 & 40 & 44 \\ 20 & 0 & 50 & 40 \\ 39 & 50 & 0 & 30 \\ 42 & 26 & 30 & 0 \end{pmatrix}$$

Cosa si richiede:

(a) Descrivete *sinteticamente a parole* e poi ad alto livello in pseudocodice, un algoritmo che utilizzando la *tecnica greedy* determini una permutazione cercando di minimizzare la distanza percorsa.

(b) Fornite una stima dei tempi di calcolo dell'algoritmo ottenuto in funzione del numero n di città.

(c) Indicate se l'algoritmo trova sempre una soluzione ottima oppure no, motivando la risposta. In particolare, in caso di risposta negativa presentate un esempio in cui non viene trovato l'ottimo (indicate la matrice delle distanze in ingresso, la soluzione ottenuta dall'algoritmo che avete scritto e una soluzione ottima, con le rispettive distanze percorse).

Possibile soluzione

Una semplice soluzione *greedy* del problema consiste nel visitare le città a partire da quella di partenza (c_1) e proseguire di volta in volta con la città più vicina che non sia ancora stata visitata. Per determinare la città più vicina si scandisce la riga corrispondente alla città che si sta visitando, e si determina l'elemento minimo considerando solo quelli corrispondenti a città non ancora visitare. L'indice (di colonna) di tale elemento, è l'indice della città da visitare successivamente.

Possiamo scrivere il seguente pseudocodice, che riceve in input il numero n di città e la matrice $n \times n$ delle distanze (supponiamo di avere indici da 1 a n, corrispondenti alle città).

```
input n, D                          //numero città, matrice distanze D
permutazione ← (1)                  //sequenza delle città
daVisitare ← {2, ..., n}            //insieme delle città che restano da visitare
ultima ← 1                          //ultima città visitata
WHILE daVisitare ≠ ∅ DO
    min ← ∞                         //determina nella riga corrispondente all'ultima città visitata
    FOR j ← 2 TO n DO               //l'indice j ∈ daVisitare t.c. D[ultima, j] è minima
        IF j ∈ daVisitare AND D[ultima, j] < min DO
            min ← D[ultima, j]
            prossima ← j                    //indice del minimo
    permutazione ← permutazione; prossima   //aggiunge la prossima città da visitare alla permutazione
    daVisitare ← daVisitare \ {prossima}    //e la elimina dall'insieme
    ultima ← prossima
output permutazione
```

Osservando che il ciclo esterno viene eseguito n volte (ad ogni iterazione si visita una nuova città) e che, per ogni esecuzione del ciclo esterno, il ciclo interno viene eseguito n volte per cercare la città più vicina, si può concludere che il tempo di calcolo è $O(n^2)$ (costo uniforme).

L'algoritmo, in generale, non trova la soluzione ottima. Ad esempio, per la matrice fornita nel testo dell'esercizio:

$$\begin{pmatrix} 0 & 20 & 40 & 44 \\ 20 & 0 & 50 & 40 \\ 39 & 50 & 0 & 30 \\ 42 & 26 & 30 & 0 \end{pmatrix}$$

l'algoritmo fornisce in output la permutazione $(1, 2, 4, 3)$, corrispondente alla distanza $20 + 40 + 30 + 39 = 129$, mentre una soluzione migliore (e ottima) è $(1, 3, 4, 2)$, corrispondente a $40 + 30 + 26 + 20 = 116$.

Note

- Il problema è quello del *commesso viaggiatore*, noto come problema computazionalmente "difficile" (nella versione di decisione il problema è NP-completo).
- Una soluzione greedy alternativa consiste nell'ordinare le distanze contenute nella matrice in ordine non decrescente (attenzione: occorre ricordare, per ciascuna distanza, la sua posizione nella matrice, quindi gli elementi saranno delle triple della forma $(d_{i,j}, i, j)$, che possiamo pensare come archi di un grafo completo pesato). Si esaminano gli elementi nell'ordine ottenuto, selezionando l'elemento $(d_{i,j}, i, j)$ solo se rispetta alcuni vincoli che garantiscono che l'insieme di archi scelto possa appartenere a un *cammino hamiltoniano* nel grafo (non è stato ancora selezionato alcun arco uscente dal vertice i ed entrante nel vertice j, la scelta dell'arco (i, j) non crea cicli, salvo all'ultimo passo). In questo modo si ottiene un insieme di n archi che forma un ciclo. La permutazione richiesta può essere ottenuta percorrendo il ciclo dal vertice 1.