Cognome.....

Nome.....

Matricola.....

Algoritmi e Strutture Dati

Prova scritta del 14 luglio 2017

TEMPO DISPONIBILE: 2 ore

Le risposte agli esercizi 1, 2, 3 devono essere scritte negli appositi riquadri su questo foglio (risposte scritte su altri fogli non saranno considerate). La soluzione dell'esercizio 4 va scritta su uno dei fogli di protocollo forniti. Le brutte copie NON devono essere consegnate. *Ricordatevi di scrivere cognome e nome su tutti i fogli.*

1. Considerate la seguente sequenza di numeri memorizzata in un array che deve essere ordinata in modo crescente:
`28 22 16 30 14 15 12 29 31`

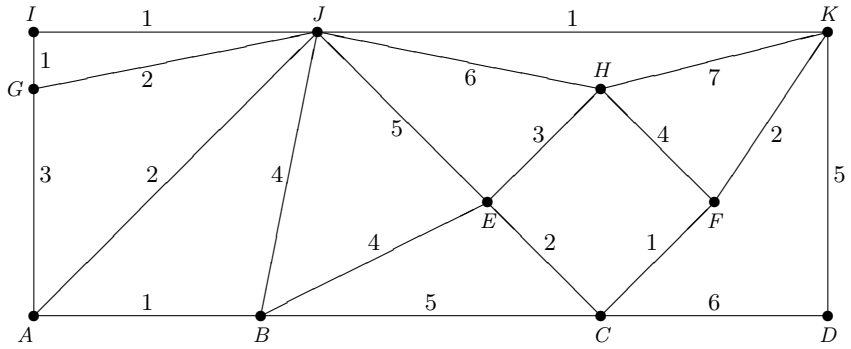
(a) Supponete di ordinare la sequenza mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento. Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate ricorsive di `quickSort`.

(b) Supponete di ordinare la sequenza mediante l'algoritmo `heapSort`. Indicate il contenuto dell'array dopo averlo trasformato in uno heap.

2. Sia G un grafo non orientato connesso con n vertici.

(a) Cosa si può affermare rispetto al numero di archi di ogni albero ricoprente il grafo G?

Svolgete i punti (b) e (c) considerando il grafo pesato rappresentato nella figura a destra, in cui le lettere indicano i nomi dei vertici e i numeri i pesi degli archi.



(b) Applicate l'algoritmo di Kruskal per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

(c) Applicate l'algoritmo di Prim, a partire dal vertice A, per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

3. Nel riquadro che segue ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

(a) Per ordinare un vettore di n elementi, l'algoritmo `heapSort` ha necessità di utilizzare per la rappresentazione dello heap una quantità di memoria aggiuntiva che cresce al crescere di n. ☐

(b) Per ordinare un vettore di n elementi, l'algoritmo `mergeSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(c) Per ordinare un vettore di n elementi, l'algoritmo `quickSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(d) A causa delle chiamate ricorsive, il numero di *spostamenti* di chiave effettuati dall'algoritmo `quickSort` può risultare esponenziale rispetto al numero di elementi da ordinare. ☐

(e) Se esiste un algoritmo per risolvere in tempo polinomiale il problema di soddisfacibilità, allora è possibile stabilire in tempo polinomiale se un grafo non orientato G di n vertici contiene una *cricca* (o *clique*) di k vertici, con G e k forniti in input. ☐

(f) L'algoritmo greedy permette sempre di ottenere in tempo polinomiale la soluzione ottima di problemi che, risolti altrimenti, richiederebbero tempo esponenziale. ☐

(g) Mediante l'algoritmo di Strassen, è possibile moltiplicare due matrici quadrate $n \times n$ contenenti numeri interi, effettuando $O(n^2)$ moltiplicazioni. ☐

(h) Mediante la visita in ampiezza, è possibile elencare in modo non decrescente tutte le chiavi contenute in un albero di ricerca AVL. ☐

(i) Siano dati due vettori ordinati in *modo crescente*, contenenti rispettivamente n ed m numeri interi, tutti diversi tra loro. Utilizzando al più $n + m - 1$ confronti è possibile ricavare un vettore ordinato in *modo descrescente* contenente gli interi presenti nei due vettori dati. ☐

(j) Quando la tecnologia riuscirà ad aumentare significativamente le velocità dei computer rispetto a quelle attuali, sarà finalmente possibile avere algoritmi efficienti per risolvere ogni problema nella classe NP. ☐

4. 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).

Cognome.....

Nome.....

Matricola.....

Algoritmi e Strutture Dati

Prova scritta del 14 luglio 2017

TEMPO DISPONIBILE: 2 ore

Le risposte agli esercizi 1, 2, 3 devono essere scritte negli appositi riquadri su questo foglio (risposte scritte su altri fogli non saranno considerate). La soluzione dell'esercizio 4 va scritta su uno dei fogli di protocollo forniti. Le brutte copie NON devono essere consegnate. *Ricordatevi di scrivere cognome e nome su tutti i fogli.*

1. Considerate la seguente sequenza di numeri memorizzata in un array che deve essere ordinata in modo crescente:
`28 22 16 30 14 15 12 29 31`

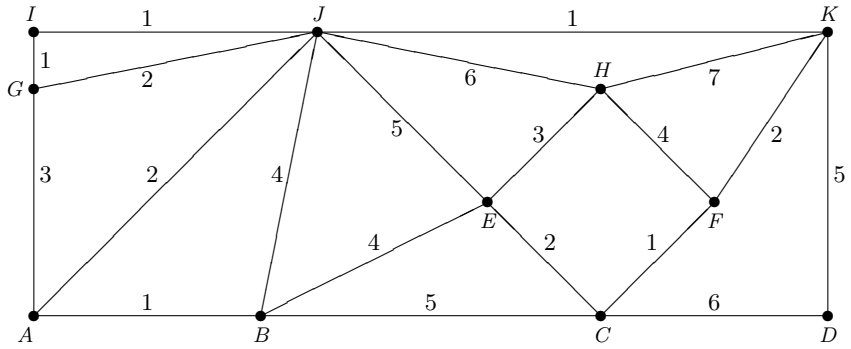
(a) Supponete di ordinare la sequenza mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento. Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate ricorsive di `quickSort`.

(b) Supponete di ordinare la sequenza mediante l'algoritmo `heapSort`. Indicate il contenuto dell'array dopo averlo trasformato in uno heap.

2. Sia G un grafo non orientato connesso con n vertici.

(a) Cosa si può affermare rispetto al numero di archi di ogni albero ricoprente il grafo G?

Svolgete i punti (b) e (c) considerando il grafo pesato rappresentato nella figura a destra, in cui le lettere indicano i nomi dei vertici e i numeri i pesi degli archi.



(b) Applicate l'algoritmo di Kruskal per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

(c) Applicate l'algoritmo di Prim, a partire dal vertice A, per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

3. Nel riquadro che segue ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

(a) Mediante la visita in ampiezza, è possibile elencare in modo non decrescente tutte le chiavi contenute in un albero di ricerca AVL. ☐

(b) Mediante l'algoritmo di Strassen, è possibile moltiplicare due matrici quadrate $n \times n$ contenenti numeri interi, effettuando $O(n^2)$ moltiplicazioni. ☐

(c) Siano dati due vettori ordinati in *modo crescente*, contenenti rispettivamente n ed m numeri interi, tutti diversi tra loro. Utilizzando al più $n + m - 1$ confronti è possibile ricavare un vettore ordinato in *modo descrescente* contenente gli interi presenti nei due vettori dati. ☐

(d) Quando la tecnologia riuscirà ad aumentare significativamente le velocità dei computer rispetto a quelle attuali, sarà finalmente possibile avere algoritmi efficienti per risolvere ogni problema nella classe NP. ☐

(e) Per ordinare un vettore di n elementi, l'algoritmo `heapSort` ha necessità di utilizzare per la rappresentazione dello heap una quantità di memoria aggiuntiva che cresce al crescere di n. ☐

(f) Per ordinare un vettore di n elementi, l'algoritmo `mergeSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(g) Per ordinare un vettore di n elementi, l'algoritmo `quickSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(h) A causa delle chiamate ricorsive, il numero di *spostamenti* di chiave effettuati dall'algoritmo `quickSort` può risultare esponenziale rispetto al numero di elementi da ordinare. ☐

(i) Se esiste un algoritmo per risolvere in tempo polinomiale il problema di soddisfacibilità, allora è possibile stabilire in tempo polinomiale se un grafo non orientato G di n vertici contiene una *cricca* (o *clique*) di k vertici, con G e k forniti in input. ☐

(j) L'algoritmo greedy permette sempre di ottenere in tempo polinomiale la soluzione ottima di problemi che, risolti altrimenti, richiederebbero tempo esponenziale. ☐

4. 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).

Cognome.....

Nome.....

Matricola.....

Algoritmi e Strutture Dati

Prova scritta del 14 luglio 2017

TEMPO DISPONIBILE: 2 ore

Le risposte agli esercizi 1, 2, 3 devono essere scritte negli appositi riquadri su questo foglio (risposte scritte su altri fogli non saranno considerate). La soluzione dell'esercizio 4 va scritta su uno dei fogli di protocollo forniti. Le brutte copie NON devono essere consegnate. *Ricordatevi di scrivere cognome e nome su tutti i fogli.*

1. Considerate la seguente sequenza di numeri memorizzata in un array che deve essere ordinata in modo crescente:
`30 24 18 32 16 17 14 31 33`

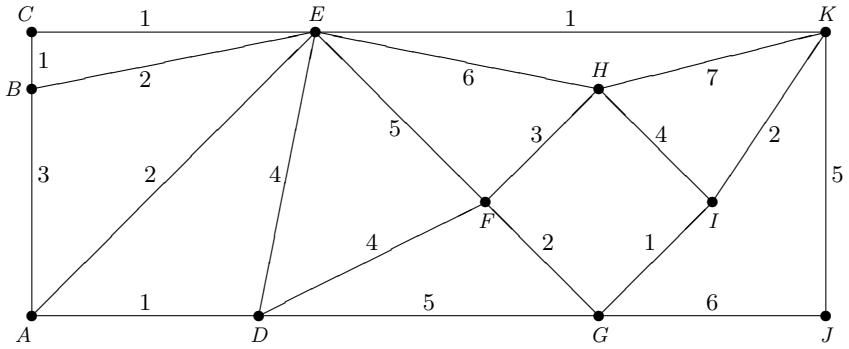
(a) Supponete di ordinare la sequenza mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento. Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate ricorsive di `quickSort`.

(b) Supponete di ordinare la sequenza mediante l'algoritmo `heapSort`. Indicate il contenuto dell'array dopo averlo trasformato in uno heap.

2. Sia G un grafo non orientato connesso con n vertici.

(a) Cosa si può affermare rispetto al numero di archi di ogni albero ricoprente il grafo G?

Svolgete i punti (b) e (c) considerando il grafo pesato rappresentato nella figura a destra, in cui le lettere indicano i nomi dei vertici e i numeri i pesi degli archi.



(b) Applicate l'algoritmo di Kruskal per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

(c) Applicate l'algoritmo di Prim, a partire dal vertice A, per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

3. Nel riquadro che segue ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

(a) Mediante l'algoritmo di Strassen, è possibile moltiplicare due matrici quadrate $n \times n$ contenenti numeri interi, effettuando $O(n^2)$ moltiplicazioni. ☐

(b) Mediante la visita in ampiezza, è possibile elencare in modo non decrescente tutte le chiavi contenute in un albero di ricerca AVL. ☐

(c) Siano dati due vettori ordinati in *modo crescente*, contenenti rispettivamente n ed m numeri interi, tutti diversi tra loro. Utilizzando al più $n + m - 1$ confronti è possibile ricavare un vettore ordinato in *modo descrescente* contenente gli interi presenti nei due vettori dati. ☐

(d) Se esiste un algoritmo per risolvere in tempo polinomiale il problema di soddisfacibilità, allora è possibile stabilire in tempo polinomiale se un grafo non orientato G di n vertici contiene una *cricca* (o *clique*) di k vertici, con G e k forniti in input. ☐

(e) L'algoritmo greedy permette sempre di ottenere in tempo polinomiale la soluzione ottima di problemi che, risolti altrimenti, richiederebbero tempo esponenziale. ☐

(f) Per ordinare un vettore di n elementi, l'algoritmo `heapSort` ha necessità di utilizzare per la rappresentazione dello heap una quantità di memoria aggiuntiva che cresce al crescere di n. ☐

(g) Per ordinare un vettore di n elementi, l'algoritmo `mergeSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(h) Per ordinare un vettore di n elementi, l'algoritmo `quickSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(i) A causa delle chiamate ricorsive, il numero di *spostamenti* di chiave effettuati dall'algoritmo `quickSort` può risultare esponenziale rispetto al numero di elementi da ordinare. ☐

(j) Quando la tecnologia riuscirà ad aumentare significativamente le velocità dei computer rispetto a quelle attuali, sarà finalmente possibile avere algoritmi efficienti per risolvere ogni problema nella classe NP. ☐

4. 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).

Cognome.....

Nome.....

Matricola.....

Algoritmi e Strutture Dati

Prova scritta del 14 luglio 2017

TEMPO DISPONIBILE: 2 ore

Le risposte agli esercizi 1, 2, 3 devono essere scritte negli appositi riquadri su questo foglio (risposte scritte su altri fogli non saranno considerate). La soluzione dell'esercizio 4 va scritta su uno dei fogli di protocollo forniti. Le brutte copie NON devono essere consegnate. *Ricordatevi di scrivere cognome e nome su tutti i fogli.*

1. Considerate la seguente sequenza di numeri memorizzata in un array che deve essere ordinata in modo crescente:
`30 24 18 32 16 17 14 31 33`

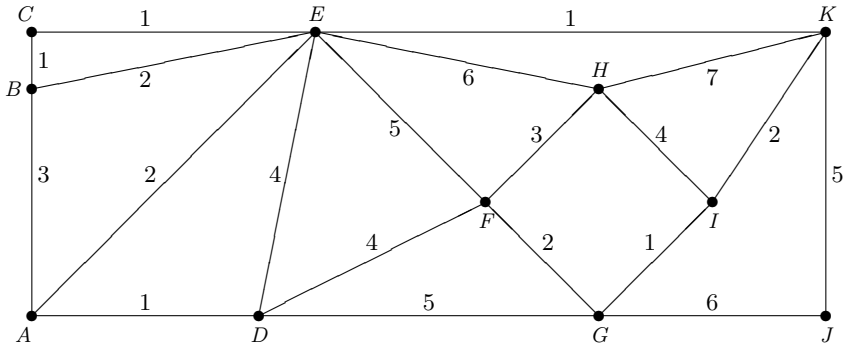
(a) Supponete di ordinare la sequenza mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento. Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate ricorsive di `quickSort`.

(b) Supponete di ordinare la sequenza mediante l'algoritmo `heapSort`. Indicate il contenuto dell'array dopo averlo trasformato in uno heap.

2. Sia G un grafo non orientato connesso con n vertici.

(a) Cosa si può affermare rispetto al numero di archi di ogni albero ricoprente il grafo G?

Svolgete i punti (b) e (c) considerando il grafo pesato rappresentato nella figura a destra, in cui le lettere indicano i nomi dei vertici e i numeri i pesi degli archi.



(b) Applicate l'algoritmo di Kruskal per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

(c) Applicate l'algoritmo di Prim, a partire dal vertice A, per individuare un albero ricoprente di costo minimo. Elencate gli archi selezionati, in un ordine con cui vengono scelti dall'algoritmo.

3. Nel riquadro che segue ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

(a) Siano dati due vettori ordinati in *modo crescente*, contenenti rispettivamente n ed m numeri interi, tutti diversi tra loro. Utilizzando al più $n + m - 1$ confronti è possibile ricavare un vettore ordinato in *modo descrescente* contenente gli interi presenti nei due vettori dati. ☐

(b) Se esiste un algoritmo per risolvere in tempo polinomiale il problema di soddisfacibilità, allora è possibile stabilire in tempo polinomiale se un grafo non orientato G di n vertici contiene una *cricca* (o *clique*) di k vertici, con G e k forniti in input. ☐

(c) L'algoritmo greedy permette sempre di ottenere in tempo polinomiale la soluzione ottima di problemi che, risolti altrimenti, richiederebbero tempo esponenziale. ☐

(d) Mediante l'algoritmo di Strassen, è possibile moltiplicare due matrici quadrate $n \times n$ contenenti numeri interi, effettuando $O(n^2)$ moltiplicazioni. ☐

(e) Mediante la visita in ampiezza, è possibile elencare in modo non decrescente tutte le chiavi contenute in un albero di ricerca AVL. ☐

(f) Quando la tecnologia riuscirà ad aumentare significativamente le velocità dei computer rispetto a quelle attuali, sarà finalmente possibile avere algoritmi efficienti per risolvere ogni problema nella classe NP. ☐

(g) Per ordinare un vettore di n elementi, l'algoritmo `heapSort` ha necessità di utilizzare per la rappresentazione dello heap una quantità di memoria aggiuntiva che cresce al crescere di n. ☐

(h) Per ordinare un vettore di n elementi, l'algoritmo `mergeSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(i) Per ordinare un vettore di n elementi, l'algoritmo `quickSort` ha necessità di utilizzare una quantità di memoria aggiuntiva che cresce al crescere di in n. ☐

(j) A causa delle chiamate ricorsive, il numero di *spostamenti* di chiave effettuati dall'algoritmo `quickSort` può risultare esponenziale rispetto al numero di elementi da ordinare. ☐

4. 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).