Progetti di Algoritmi e Strutture Dati

Università di Bologna, corso di laurea in Informatica per il Management

Anno Accademico 2022/2023, sessione autunnale.

Istruzioni

Il progetto consiste in **cinque** esercizi di programmazione da realizzare in Java. Lo svolgimento del progetto è obbligatorio per poter sostenere l'orale nella sessione cui il progetto si riferisce.

I progetti dovranno essere consegnati entro le ore **23:59 del 04/09/2023**. La prova orale potrà essere sostenuta nell'unico appello della sessione autunnale (è obbligatoria l'iscrizione tramite AlmaEsami).

L'esito dell'esame dipende sia dalla correttezza ed efficienza dei programmi consegnati, sia dal risultato della discussione orale, durante la quale verrà verificata la conoscenza della teoria spiegata in tutto il corso (quindi non solo quella necessaria allo svolgimento dei progetti). L'orale è importante: **una discussione insufficiente comporterà il non superamento della prova**.

Modalità di svolgimento dei progetti

I progetti devono essere **esclusivamente frutto del lavoro individuale di chi li consegna; è vietato discutere i progetti e le soluzioni con altri** (sia che si tratti di studenti del corso o persone terze). La similitudine tra progetti verrà verificata con strumenti automatici e, se confermata, comporterà l'immediato annullamento della prova per TUTTI gli studenti coinvolti senza ulteriori valutazioni dei progetti.

È consentito l'uso di algoritmi e strutture dati definite nella libreria standard Java, nonché di codice messo a disposizione dai docenti sulla pagina del corso o sulla piattaforma "Virtuale"; è responsabilità di ciascuno verificare che il codice sia corretto (anche e soprattutto quello fornito dai docenti!). **Non è consentito fare uso di altro codice, anche se liberamente disponibile in rete**.

I programmi devono essere realizzati come applicazioni a riga di comando. Ciascun esercizio deve essere implementato in un singolo file sorgente chiamato Esercizio*N*.java, (Esercizio1.java, Esercizio2.java eccetera). Il file deve avere una classe pubblica chiamata Esercizio*N*, contenente il metodo statico main(); altre classi, se necessarie, possono essere definite all'interno dello stesso file. I programmi **non devono specificare il package** (quindi **non devono contenere** l'intestazione "package Esercizio1;" o simili).

I programmi devono iniziare con un blocco di commento contenente nome, cognome, numero di matricola e indirizzo mail (@studio.unibo.it) dell'autore. Nel commento iniziale è possibile indicare per iscritto eventuali informazioni utili alla valutazione del programma (ad esempio, considerazioni sull'uso di strutture dati particolari, costi asintotici eccetera).

I programmi verranno compilati dalla riga di comando utilizzando Java 11 (OpenJDK 11) con il comando:

javac Esercizio*N*.java

ed eseguiti sempre dalla riga di comando con

java -cp . Esercizio*N* *eventuali_parametri_di_input*

I programmi non devono richiedere nessun input ulteriore da parte dell'utente. Il risultato deve essere stampato a video rispettando **scrupolosamente** il formato indicato in questo documento, perché i programmi subiranno una prima fase di controlli semiautomatici. **Non verranno accettati programmi che producono un output non conforme alle specifiche**.

Si può assumere che i dati di input siano sempre corretti. Vengono forniti alcuni esempi di input per i vari esercizi, con i rispettivi output previsti. **Un programma che produce il risultato corretto con i dati di input forniti non è necessariamente corretto**. I programmi consegnati devono funzionare correttamente su *qualsiasi* input: a tale scopo verranno testati anche con input differenti da quelli forniti.

Alcuni problemi potrebbero ammettere più soluzioni corrette; in questi casi – salvo indicazione diversa data nella specifica – il programma può restituirne una qualsiasi, anche se diversa da quella mostrata nel testo o fornita con i dati di input/output di esempio. Nel caso di esercizi che richiedano la stampa di risultati di operazioni in virgola mobile, i risultati che si ottengono possono variare leggermente in base all'ordine con

cui vengono effettuate le operazioni, oppure in base al fatto che si usi il tipo di dato `float` o `double`; tali piccole differenze verranno ignorate.

I file di input assumono che i numeri reali siano rappresentati usando il punto ('.') come separatore tra la parte intera e quella decimale. Questa impostazione potrebbe non essere il default nella vostra installazione di Java, ma è sufficiente inserire all'inizio del metodo `main()` la chiamata:

```
Locale.setDefault(Locale.US);
```

per impostare il separatore in modo corretto (importare `java.util.Locale` per rendere disponibile il metodo).

Ulteriori requisiti

La correttezza delle soluzioni proposte deve essere dimostrabile. In sede di discussione dei progetti potrà essere richiesta la dimostrazione che il programma sia corretto. Per "dimostrazione" si intende una dimostrazione formale, del tipo di quelle descritte nel libro o viste a lezione per garantire la correttezza degli algoritmi. Argomentazioni fumose che si limitano a descrivere il programma riga per riga e altro non sono considerate dimostrazioni.

Il codice deve essere leggibile. Programmi incomprensibili e mal strutturati (ad es., contenenti metodi troppo lunghi, oppure un eccessivo livello di annidamento di cicli/condizioni – "if" dentro "if" dentro "while" dentro "if"…) verranno fortemente penalizzati o, nei casi più gravi, rifiutati.

Usare nomi appropriati per variabili, classi e metodi. L'uso di nomi inappropriati rende il codice difficile da comprendere e da valutare. L'uso di nomi di identificatori deliberatamente fuorviante potrà essere pesantemente penalizzato in sede di valutazione degli elaborati.

Commentare il codice in modo adeguato. I commenti devono essere usati per descrivere in modo sintetico i punti critici del codice, non per parafrasarlo riga per riga.

Esempio di commenti inutili	***Esempio di commento appropriato***

`v = v + 1;  // incrementa v`
`if ( v>10 ) { // se v e' maggiore di 10`
`  v = 0;  // setta v a zero`
`}`
`G.Kruskal(v); // esegui l'algoritmo di Kruskal`
	`// Individua la posizione i del primo valore`
	`// negativo nell'array a[]; al termine si ha`
	`// i == a.length se non esiste alcun`
	`// valore negativo.`
	`int i = 0;`
	`while ( i < a.length && a[i] >= 0 ) {`
	`  i++;`
	`}`

Ogni metodo deve essere preceduto da un blocco di commento che spieghi in maniera sintetica lo scopo di quel metodo.

Lunghezza delle righe di codice. Le righe dei sorgenti devono avere lunghezza contenuta (indicativamente minore o uguale a 80 caratteri). Righe troppo lunghe rendono il sorgente difficile da leggere e da valutare.

Usare strutture dati adeguate. Salvo dove diversamente indicato, è consentito l'utilizzo di strutture dati e algoritmi già implementato nella JDK. Decidere quale struttura dati o algoritmo siano più adeguati per un determinato problema è tra gli obbiettivi di questo corso, e pertanto avrà un impatto significativo sulla valutazione.

Modalità di consegna

I sorgenti vanno consegnati tramite la piattaforma “Virtuale” caricando i singoli file .java (Esercizio1.java,

Esercizio2.java, eccetera). Tutto il codice necessario a ciascun esercizio deve essere incluso nel relativo sorgente; non sono quindi ammessi sorgenti multipli relativi allo stesso esercizio.

Forum di discussione

È stato creato un forum di discussione sulla piattaforma "Virtuale". Le richieste di chiarimenti sulle specifiche degli esercizi (cioè sul contenuto di questo documento) **vanno poste esclusivamente sul forum** e non via mail ai docenti. Non verrà data risposta a richieste di fare debug del codice, o altre domande di programmazione: queste competenze devono essere già state acquisite, e verranno valutate come parte dell'esame.

Valutazione dei progetti

Gli studenti ammessi all'orale verranno convocati per discutere i progetti, secondo un calendario che verrà comunicato sulla pagina del corso. Di norma, **potranno accedere all'orale solo coloro che avranno svolto gli esercizi del progetto in modo corretto.**

La discussione includerà domande sugli esercizi consegnati e sulla teoria svolta a lezione. Chi non sarà in grado di fornire spiegazioni esaurienti sul funzionamento dei programmi consegnati durante la prova orale riceverà una valutazione insufficiente con conseguente necessità di rifare l'esame da zero in una sessione d'esame successiva su nuovi progetti. Analogamente, una conoscenza non sufficiente degli argomenti di teoria, anche relativi a temi non trattati nei progetti, comporterà il non superamento della prova.

La valutazione dei progetti sarà determinata dai parametri seguenti:

- Correttezza dei programmi implementati;
- Efficienza dei programmi implementati;
- Chiarezza del codice: codice poco comprensibile, ridondante o inefficiente comporterà penalizzazioni, indipendentemente dalla sua correttezza. **L'uso di nomi di identificatori fuorvianti o a casaccio verrà fortemente penalizzato**.
- Capacità dell'autore/autrice di spiegare e giustificare le scelte fatte, di argomentare sulla correttezza e sul costo computazionale del codice e in generale di rispondere in modo esauriente alle richieste di chiarimento e/o approfondimento da parte dei docenti.

Checklist

Viene riportata in seguito un elenco di punti da controllare prima della consegna:

1. Ogni esercizio è stato implementato in un UNICO file sorgente Esercizio*N*.java?
2. I programmi compilano dalla riga di comando come indicato in questo documento?
3. I sorgenti includono all'inizio un blocco di commento che riporta cognome, nome, numero di matricola e indirizzo di posta (@studio.unibo.it) dell'autore?
4. I programmi consegnati producono il risultato corretto usando gli esempi di input forniti?

Esercizio 1.

È necessario monitorare i siti web attivi del pianeta Orion e, per questo scopo, vi viene richiesto di realizzare una struttura dati contenente le informazioni relative all'insieme dei siti web del pianeta.

Il singolo sito web viene rappresentato da una coppia <k, np>, dove:

- k, intero positivo, è una chiave identificativa del sito.
- np: numero di pagine presenti nel sito.

Orion comunica che il numero di **possibili** siti web è di circa un miliardo ma il numero di siti web **attivi** è non superiore a 10000.

Inoltre, viene segnalato che le chiavi che identificano i siti web, sono uniformemente distribuite nell'intervallo $[1, 1\times10^9]$, mentre il valore di np è un intero nell'intervallo [1, 700];

Si richiede di realizzare una struttura dati contenente le informazioni relative all'insieme dei siti web attivi nel pianeta e di implementare metodi per:

a) verificare che il sito corrispondente ad una data chiave (verifica(k)) sia presente nella struttura dati e, nel caso di presenza, stampare (stampa(k)) le informazioni che lo caratterizzano (<k, np>);

b) inserire, dopo aver verificato che non sia già presente, un nuovo sito, (inserisci (k, np));

c) stampare il numero di siti web effettivamente presenti nella struttura, size().

Si richiede inoltre che la struttura dati progettata garantisca per (verifica(k)) e (inserisci (k, np)) un numero medio di accessi (NMA) alla struttura che non sia superiore a T (T=10, T=5).
Si noti che il numero di siti attivi previsti è non superiore a 10000.

Per verificare la soluzione proposta si richiede di:

- utilizzare opportuni generatori di numeri pseudo-casuali per inserire le coppie <k, np> nella struttura dati e utilizzare come seme iniziale il proprio numero di matricola;
- generare casualmente 300 chiavi (k); invocare, per ognuna delle chiavi generate la funzione verifica(k) e memorizzare in un file di output i risultati ottenuti in forma di coppie (k, numero di accessi riportati), utilizzare come seme per generare le chiavi il valore 7105419;
- stimare il valore NMA come media dei valori calcolati al punto precedente.

Nota: in questo esercizio non è consentito usare alcuna delle implementazioni delle interfacce Map (o Dictionary) già fornite con Java; quindi, non è consentito usare HashMap, TreeMap e ogni altra classe derivata da Map o Dictionary.

Esercizio 2.

La società Icarus deve gestire informazioni fornite da una particolare applicazione. L'applicazione produce coppie, **<x, q>**; il primo elemento presente nella generica coppia, **x**, è un intero positivo, mentre il secondo, **q**, consiste di una stringa di caratteri contenente esclusivamente lettere minuscole.

Un esempio di coppie prodotte dall'applicazione è il seguente:
170 albergo
130 campionato
90 fiume
20 patate
8 frutta
222 eletto
51 sentieri
130 monti
208 valalla
180 promontorio.

Una volta memorizzate le coppie fornite dall'applicazione, si richiede di progettare e implementare algoritmi per le seguenti operazioni:

1) ricerca e stampa delle coppie con $a \leq x \leq b$, e lunghezza della stringa $\leq s$
2) stampa di tutte le coppie aventi valori del primo elemento $x \geq c$

Il programma accetta come parametro il nome del file contenente le coppie (intero, stringa), separate da spazi.
Si noti che il valore di n, numero delle coppie, deve essere desunto dal numero di coppie presenti in tale file.
Da riga di comando vengono anche inserite le richieste relative ai punti:
1) valori di a, b e s;
2) valore di c.
I risultati della richiesta devono essere stampati
Prevedere un comando per interrompere l'esecuzione del programma.
Discutere la complessità computazionale della soluzione proposta.

Esercizio 3.

Un'impresa edile ogni anno valuta i suoi dipendenti in vista di una promozione o di un premio o un incremento di stipendio utilizzando un indice (un indice di impegno o di competenza) associato a ciascun dipendente. Questo indice è un numero reale *non negativo* e più è alto questo indice, più il dipendente è considerato competente/attivo. Supponiamo che gli indici associati ai dipendenti siano stati ordinati in ordine non decrescente (I[1] <= *I*[2] <= ... <= *I*[*n*], *I*[*j*] è l'indice del *j*-esimo dipendente e ci sono in totale *n* dipendenti). A questo punto, l'impresa ha definito in qualche modo una soglia (un numero reale non negativo) *k.* Verranno premiati/promossi solo i dipendenti più bravi, il cui indice di competenza/impegno sia strettamente maggiore di *k*.

- Scrivere un algoritmo efficiente, basato sulla tecnica ***Divide-et-Impera***, che, dato il vettore ordinato *I*[1..*n*] e la soglia *k*, calcola il numero totale di dipendenti premiati/promossi. Il programma deve accettare sulla riga di comando il nome di un file di input il cui contenuto abbia la seguente struttura:

```
 -------------------------
| INPUT FILE              |
-------------------------
```

numero di dipendenti
10

vettore *l*
0.015899959834469013
0.20387478195313158
0.3359524025416939
0.34690742873967684
0.45823330506267057
0.617314071997303
0.6465821602909256
0.8575884598068334
0.9468595742485053
0.9513577109193919

soglia *k*
0.5

Il programma deve stampare a video il numero di dipendenti premiati/promossi:

Dipendenti premiati: 5

- Determinare il costo computazionale asintotico dell'algoritmo proposto al punto 1.

Esercizio 4.

Consideriamo una rete di telecomunicazione chiamata **Abilene**, che è composta da 12 nodi e 15 link bidirezionali/edges come illustrato nella seguente figura.

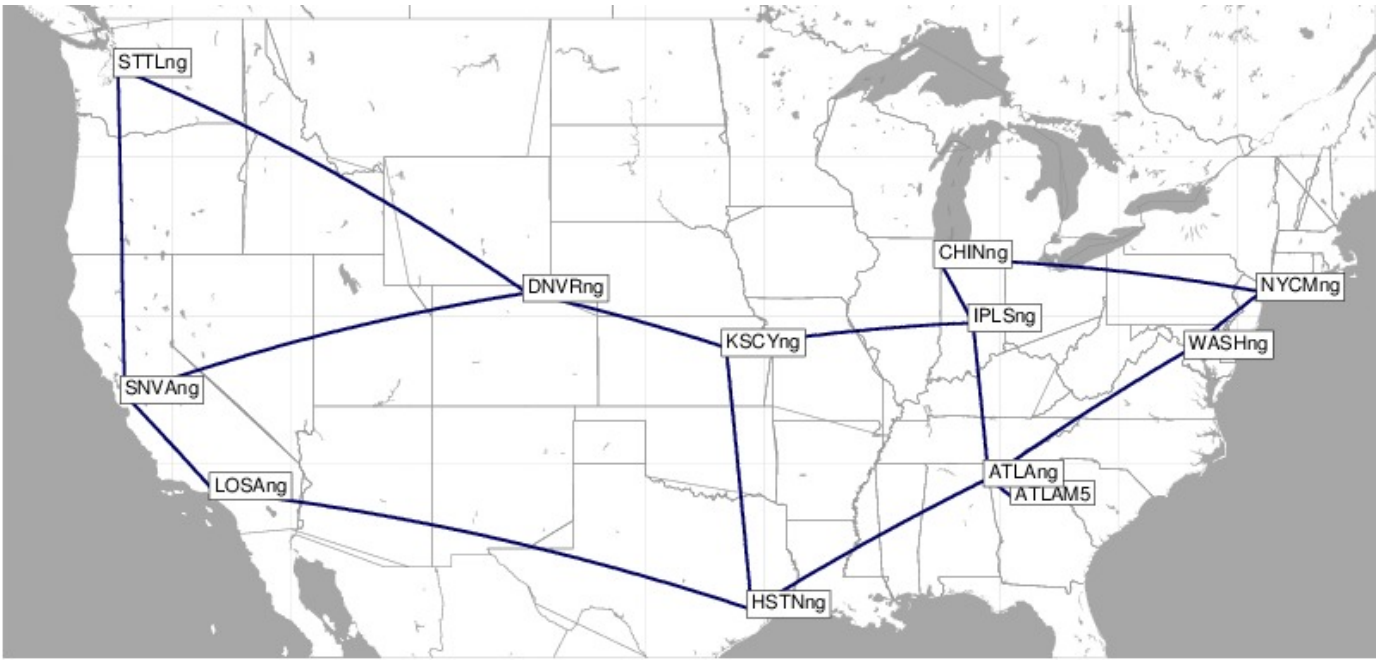

Figura 1: Abilene network topology (http://sndlib.zib.de/home.action?show=/problems.overview.action%3Fframeset)

Le specifiche di questa rete (un grafo non orientato e pesato) sono riportate qui sotto (***Sorgente:*** ***http://sndlib.zib.de/home.action?show=/problems.overview.action%3Fframeset***):

```
# network abilene

# NODE SECTION
#
# <node_id> [(<longitude>, <latitude>)]

NODES (
  ATLAM5 ( -84.3833 33.75 )
  ATLAng ( -85.50 34.50 )
  CHINng ( -87.6167 41.8333 )
  DNVRng ( -105.00 40.75 )
  HSTNng ( -95.517364 29.770031 )
  IPLSng ( -86.159535 39.780622 )
  KSCYng ( -96.596704 38.961694 )
  LOSAng ( -118.25 34.05 )
  NYCMng ( -73.9667 40.7833 )
  SNVAng ( -122.02553 37.38575 )
  STTLng ( -122.30 47.60 )
  WASHng ( -77.026842 38.897303 )
)

# LINK SECTION
#
# <link_id> ( <source> <target> ) <pre_installed_capacity> <pre_installed_capacity_cost> <routing_cost>
<setup_cost> ( {<module_capacity> <module_cost>}* )

LINKS (
  ATLAM5_ATLAng ( ATLAng ATLAM5 ) 9920.00 0.00 0.00 0.00 ( 40000.00 133.00 )
  ATLAng_HSTNng ( HSTNng ATLAng ) 9920.00 0.00 0.00 0.00 ( 40000.00 1081.00 )
  ATLAng_IPLSng ( IPLSng ATLAng ) 2480.00 0.00 0.00 0.00 ( 40000.00 591.00 )
  ATLAng_WASHng ( WASHng ATLAng ) 9920.00 0.00 0.00 0.00 ( 40000.00 901.00 )
  CHINng_IPLSng ( IPLSng CHINng ) 9920.00 0.00 0.00 0.00 ( 40000.00 260.00 )
  CHINng_NYCMng ( NYCMng CHINng ) 9920.00 0.00 0.00 0.00 ( 40000.00 1147.00 )
  DNVRng_KSCYng ( KSCYng DNVRng ) 9920.00 0.00 0.00 0.00 ( 40000.00 745.00 )
  DNVRng_SNVAng ( SNVAng DNVRng ) 9920.00 0.00 0.00 0.00 ( 40000.00 1516.00 )
  DNVRng_STTLng ( STTLng DNVRng ) 9920.00 0.00 0.00 0.00 ( 40000.00 1573.00 )
  HSTNng_KSCYng ( KSCYng HSTNng ) 9920.00 0.00 0.00 0.00 ( 40000.00 1028.00 )
  HSTNng_LOSAng ( LOSAng HSTNng ) 9920.00 0.00 0.00 0.00 ( 40000.00 2196.00 )
  IPLSng_KSCYng ( KSCYng IPLSng ) 9920.00 0.00 0.00 0.00 ( 40000.00 903.00 )
  LOSAng_SNVAng ( SNVAng LOSAng ) 9920.00 0.00 0.00 0.00 ( 40000.00 505.00 )
  NYCMng_WASHng ( WASHng NYCMng ) 9920.00 0.00 0.00 0.00 ( 40000.00 336.00 )
  SNVAng_STTLng ( STTLng SNVAng ) 9920.00 0.00 0.00 0.00 ( 40000.00 1138.00 )
)

# DEMAND SECTION
#
# <demand_id> ( <source> <target> ) <routing_unit> <demand_value> <max_path_length>

DEMANDS (
  IPLSng_STTLng ( IPLSng STTLng ) 1 3580.00 UNLIMITED
  CHINng_ATLAM5 ( CHINng ATLAM5 ) 1 2770.00 UNLIMITED
  HSTNng_STTLng ( HSTNng STTLng ) 1 1745.00 UNLIMITED
…
)
```

Per semplicità andiamo ad associare ai nodi un *id* intero che va da 0 a 11 al posto degli id ATLAM5, ATLAng, …, WASHn.
- Il programma da realizzare deve accettare come unico parametro il nome di un file di input con i dati forniti di cui sopra e deve calcolare i cammini minimi fra tutte le coppie di nodi. Il ***peso*** associato ad un link/arco *(i,j)*, ***w_ij***, è calcolato così: w_ij = max_(capacità pre-installate di tutti i link)/capacità pre-installata sul link *(i,j)*, ovvero, usando la notazione dei dati relativi alla topologia Abilene, w_ij = max_(pre_installed_capacity di tutti i link) / pre_installed_capacity sul link *(i,j)*.

Si richiede di usare in questo esercizio l'***algoritmo di Bellman-Ford*** per calcolare i cammini minimi. Il programma deve stampare a video *tutte le coppie di nodi* (partendo dal nodo 0 e cosi' via fino al nodo 11), il costo totale del cammino minimo e il cammino minimo (la sorgente, i nodi intermedi e la destinazione) tra ogni coppia.

- Calcolare il costo computazionale totale del programma in funzione del numero di nodi e di archi.

Esempio di output atteso per i dati di input forniti (si nota che possono esistere più di un cammino di costo minimo fra una coppia di nodi):

```
  s    d        dist path
---- ---- ------------ --------------------
   0    0       0.0000 0
   0    1       1.0000 0->1
   0    2       4.0000 0->1->11->8->2
   0    3       4.0000 0->1->4->6->3
   0    4       2.0000 0->1->4
   0    5       4.0000 0->1->4->6->5
   0    6       3.0000 0->1->4->6
   0    7       3.0000 0->1->4->7
   0    8       3.0000 0->1->11->8
   0    9       4.0000 0->1->4->7->9
   0   10       5.0000 0->1->4->7->9->10
   0   11       2.0000 0->1->11

  s    d        dist path
---- ---- ------------ --------------------
   1    0       1.0000 1->0
   1    1       0.0000 1
   1    2       3.0000 1->11->8->2
   1    3       3.0000 1->4->6->3
   1    4       1.0000 1->4
   1    5       3.0000 1->4->6->5
   1    6       2.0000 1->4->6
   1    7       2.0000 1->4->7
   1    8       2.0000 1->11->8
   1    9       3.0000 1->4->7->9
   1   10       4.0000 1->4->7->9->10
   1   11       1.0000 1->11

  s    d        dist path
---- ---- ------------ --------------------
   2    0       4.0000 2->8->11->1->0
   2    1       3.0000 2->8->11->1
   2    2       0.0000 2
   2    3       3.0000 2->5->6->3
   2    4       3.0000 2->5->6->4
   2    5       1.0000 2->5
   2    6       2.0000 2->5->6
   2    7       4.0000 2->5->6->4->7
   2    8       1.0000 2->8
   2    9       4.0000 2->5->6->3->9
   2   10       4.0000 2->5->6->3->10
   2   11       2.0000 2->8->11
```

s	d	dist	path
3	0	4.0000	3->6->4->1->0
3	1	3.0000	3->6->4->1
3	2	3.0000	3->6->5->2
3	3	0.0000	3
3	4	2.0000	3->6->4
3	5	2.0000	3->6->5
3	6	1.0000	3->6
3	7	2.0000	3->9->7
3	8	4.0000	3->6->5->2->8
3	9	1.0000	3->9
3	10	1.0000	3->10
3	11	4.0000	3->6->4->1->11

s	d	dist	path
4	0	2.0000	4->1->0
4	1	1.0000	4->1
4	2	3.0000	4->6->5->2
4	3	2.0000	4->6->3
4	4	0.0000	4
4	5	2.0000	4->6->5
4	6	1.0000	4->6
4	7	1.0000	4->7
4	8	3.0000	4->1->11->8
4	9	2.0000	4->7->9
4	10	3.0000	4->7->9->10
4	11	2.0000	4->1->11

s	d	dist	path
5	0	4.0000	5->6->4->1->0
5	1	3.0000	5->6->4->1
5	2	1.0000	5->2
5	3	2.0000	5->6->3
5	4	2.0000	5->6->4
5	5	0.0000	5
5	6	1.0000	5->6
5	7	3.0000	5->6->4->7
5	8	2.0000	5->2->8
5	9	3.0000	5->6->3->9
5	10	3.0000	5->6->3->10
5	11	3.0000	5->2->8->11

s	d	dist	path
6	0	3.0000	6->4->1->0
6	1	2.0000	6->4->1
6	2	2.0000	6->5->2
6	3	1.0000	6->3
6	4	1.0000	6->4
6	5	1.0000	6->5
6	6	0.0000	6
6	7	2.0000	6->4->7
6	8	3.0000	6->5->2->8
6	9	2.0000	6->3->9
6	10	2.0000	6->3->10
6	11	3.0000	6->4->1->11

s	d	dist	path
7	0	3.0000	7->4->1->0
7	1	2.0000	7->4->1
7	2	4.0000	7->4->6->5->2
7	3	2.0000	7->9->3
7	4	1.0000	7->4
7	5	3.0000	7->4->6->5
7	6	2.0000	7->4->6

```
  7    7       0.0000 7
  7    8       4.0000 7->4->1->11->8
  7    9       1.0000 7->9
  7   10       2.0000 7->9->10
  7   11       3.0000 7->4->1->11

  s    d         dist path
---- ---- ------------ --------------------
  8    0       3.0000 8->11->1->0
  8    1       2.0000 8->11->1
  8    2       1.0000 8->2
  8    3       4.0000 8->2->5->6->3
  8    4       3.0000 8->11->1->4
  8    5       2.0000 8->2->5
  8    6       3.0000 8->2->5->6
  8    7       4.0000 8->11->1->4->7
  8    8       0.0000 8
  8    9       5.0000 8->2->5->6->3->9
  8   10       5.0000 8->2->5->6->3->10
  8   11       1.0000 8->11

  s    d         dist path
---- ---- ------------ --------------------
  9    0       4.0000 9->7->4->1->0
  9    1       3.0000 9->7->4->1
  9    2       4.0000 9->3->6->5->2
  9    3       1.0000 9->3
  9    4       2.0000 9->7->4
  9    5       3.0000 9->3->6->5
  9    6       2.0000 9->3->6
  9    7       1.0000 9->7
  9    8       5.0000 9->3->6->5->2->8
  9    9       0.0000 9
  9   10       1.0000 9->10
  9   11       4.0000 9->7->4->1->11

  s    d         dist path
---- ---- ------------ --------------------
 10    0       5.0000 10->3->6->4->1->0
 10    1       4.0000 10->3->6->4->1
 10    2       4.0000 10->3->6->5->2
 10    3       1.0000 10->3
 10    4       3.0000 10->3->6->4
 10    5       3.0000 10->3->6->5
 10    6       2.0000 10->3->6
 10    7       2.0000 10->9->7
 10    8       5.0000 10->3->6->5->2->8
 10    9       1.0000 10->9
 10   10       0.0000 10
 10   11       5.0000 10->3->6->4->1->11

  s    d         dist path
---- ---- ------------ --------------------
 11    0       2.0000 11->1->0
 11    1       1.0000 11->1
 11    2       2.0000 11->8->2
 11    3       4.0000 11->1->4->6->3
 11    4       2.0000 11->1->4
 11    5       3.0000 11->8->2->5
 11    6       3.0000 11->1->4->6
 11    7       3.0000 11->1->4->7
 11    8       1.0000 11->8
 11    9       4.0000 11->1->4->7->9
 11   10       5.0000 11->1->4->7->9->10
 11   11       0.0000 11
```

Esercizio 5.

Riprendiamo la stessa rete (lo stesso grafo) dell'esercizio precedente con le stesse specifiche e andiamo a calcolare i cammini minimi fra tutte le coppie di nodi con l'uso della ***programmazione dinamica***. Anche in questo esercizio, il programma deve stampare a video le coppie di nodi e il costo totale del cammino minimo fra ogni coppia di nodi.

- Calcolare il costo computazionale totale del programma in funzione del numero di nodi e di archi.

Esempio di output: si veda l'esempio di output dell'Esercizio 4 (si nota che possono esistere più di un cammino di costo minimo fra una coppia di nodi).