Progetti di Algoritmi e Strutture Dati

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

Anno Accademico 2022/2023, sessione invernale.

Istruzioni

Il progetto consiste in **quattro** 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** di giovedì **18/01/2024**. La prova orale potrà essere sostenuta in uno dei due appelli della sessione invernale (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

```
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
```

Esempio di commento appropriato

```
// 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.

L'associazione “Librerie Storiche” decide di mettere on-line i dati dei propri iscritti.
I dati caratterizzanti ogni singolo iscritto sono i seguenti:
- nome della libreria;
- città;
- indirizzo;
- anno di fondazione;
- link al sito della libreria;
- codice identificativo univoco (CIU); Il codice di ogni singolo socio consiste di un intero positivo

Nella stessa città non esistono librerie aventi lo stesso nome.
La struttura dati dovrà essere organizzata nel seguente modo: le librerie il cui nome inizia con la stessa lettera (maiuscola o minuscola: si assume che il primo carattere del nome è costituito da una lettera) verranno inserite in una lista che dovrà essere mantenuta ordinata rispetto al nome, e in caso di nomi uguali, rispetto al codice identificativo.
Le liste (una per ogni lettera) dovranno poi essere inserite in un array (ogni posizione dell'array conterrà un riferimento (reference) ad una lista).

I dati da inserire dovranno essere letti dal file InputEs1.txt e avranno il seguente formato:
—— libreria città anno link codice ——
Un esempio:
atena Bologna 1920 xxx 00001
libertà Pisa 1915 zzz 00002
popolare Trento 1899 zzy 00003
aprile-25 Lecce 1945 yyy 00004
sitor Napoli 1919 kkk 00005
Beta Napoli 1925 qqq 00006
progresso Bologna 1921 sss 00007
Intorcia Benevento 1940 cvs 00008
belloni Pisa 1919 inc 00009
Kitor Catania 1918 nio 00010
belleri Udine 1899 ppp 00011

Il programma accetta come parametro il nome del file contenente i dati nel formato descritto in precedenza.
Una volta memorizzati i dati dei singoli iscritti, si richiede di progettare e implementare algoritmi per le seguenti operazioni:
1) ricerca (per nome) di un iscritto e, nel caso di presenza nella lista, stampa a video delle informazioni relative ad esso;
2) stampa a video di tutti gli iscritti dell’associazione;
3) stampa a video degli iscritti della stessa città;
4) cancellazione (dopo ricerca per verificarne la presenza) di un iscritto;
5) inserimento (dopo ricerca per verificarne la NON presenza) di un iscritto.

Le richieste relative ai punti precedenti vengo inserite da riga di comando.
I risultati delle richieste devono essere stampati a video.
Prevedere un comando per interrompere l’esecuzione del programma.
Discutere la complessità computazionale delle soluzioni proposte per le operazioni richieste (punti 1-5).

Librerie Storiche un anno dopo…

Alla società Binary è stata affidata la gestione del patrimonio librario della associazione Librerie Storiche. L'associazione ora gestisce N librerie con valori CIU pari a $\{i_1, i_2, \ldots i_N, i_1 < i_2 < i_3 < \ldots < i_N\}$. Per ogni libreria è stata creata una coppia <CIU, PR>: CIU è il codice identificativo unico della libreria, mentre PR è una reference (ricavata dal sito web della libreria) ad una lista che contiene tutti i libri della specifica libreria.

Facendo riferimento all'esempio precedente un possibile input è il seguente:

00001 xxx/PR
00002 zzz/PR
000013 zzzz/PR
000040 yyyy/PR
000053 kkks/PR
000061 qqqw/PR
000067 sssq/PR
000068 cvsc/PR
000091 incqw/PR
000110 nios/PR
000114 pppw/PR

Il programma accetta come parametro il nome del file contenente i dati nel formato descritto in precedenza. Si suppone che i dati in ingresso siano ordinati per valori di CIU crescenti.
Una volta memorizzati (opportunamente) i dati dei singoli iscritti, si richiede di progettare e implementare algoritmi per le seguenti operazioni:

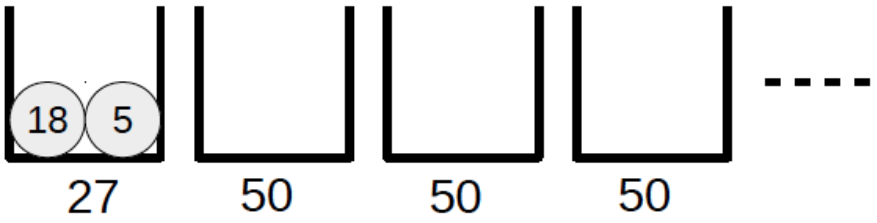
1) verificare che una libreria (utilizzando il codice CIU) sia presente nella struttura dati;
2) fornire (dopo aver verificato che il CIU è presente nella struttura) il valore corrispondente di PR.

Le richieste relative ai punti precedenti vengo inserite da riga di comando.
I risultati delle richieste devono essere stampati a video.
Prevedere un comando per interrompere l'esecuzione del programma.
Discutere la complessità computazionale delle soluzioni proposte per le operazioni richieste.

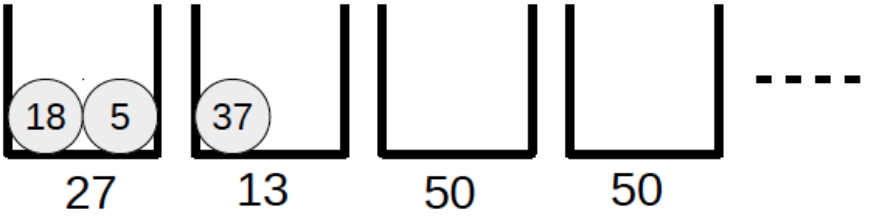
Esercizio 2.

Disponiamo di un numero potenzialmente illimitato di contenitori in grado di contenere **C** kg di oggetti ciascuno (**C** è un numero reale positivo). Vogliamo traslocare ***n*** oggetti aventi pesi **p0**, **p1**, …, **pn** utilizzando il numero minimo possibile di scatoloni; tutti i pesi sono reali positivi minori o uguali a C. In generale non esiste un algoritmo efficiente per determinare la soluzione ottimale di questo problema, per cui si richiede di progettare e realizzare un algoritmo che implementi la politica detta "First Fit". L'idea è la seguente: numeriamo gli infiniti contenitori con gli interi {0, 1, 2, … }; si considerano gli oggetti nell'ordine in cui compaiono nella lista (che pertanto non va riordinata in alcun modo), e ogni oggetto va inserito nel primo scatolone in grado di contenerlo.

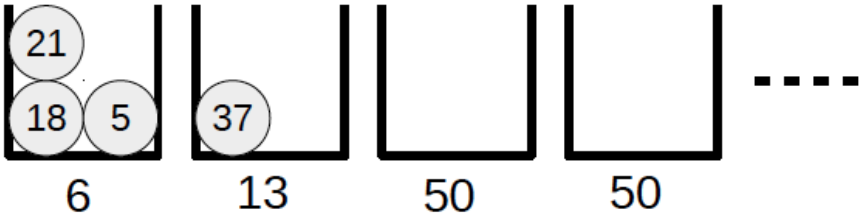
Ad esempio, consideriamo sette oggetti di pesi 18, 5, 37, 21, 27, 12, 19 da inscatolare in contenitori di capacità C = 50 kg. I primi due oggetti (pesi 18 e 5) vengono inseriti nel primo contenitore (i numeri in basso indicano la capacità residua di ciascun contenitore):



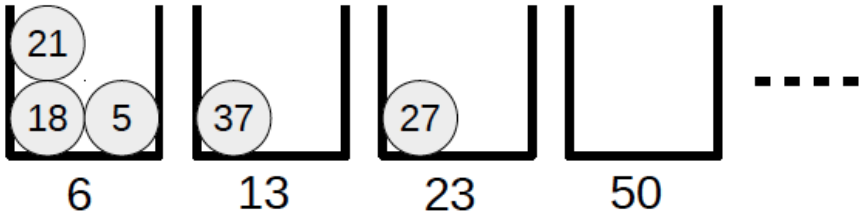
Il terzo oggetto, di peso 37, non trova spazio nel primo contenitore la cui capienza residua è 27 kg; di conseguenza, si userà il secondo:



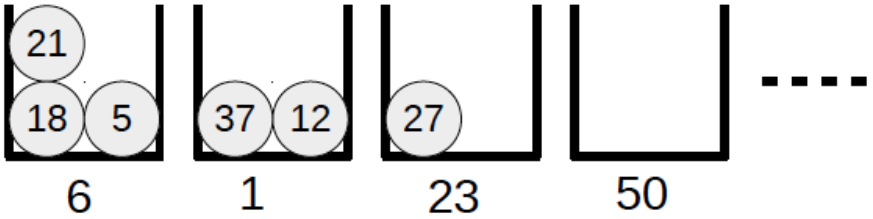
Il quarto oggetto di peso 21 trova posto nel primo contenitore:



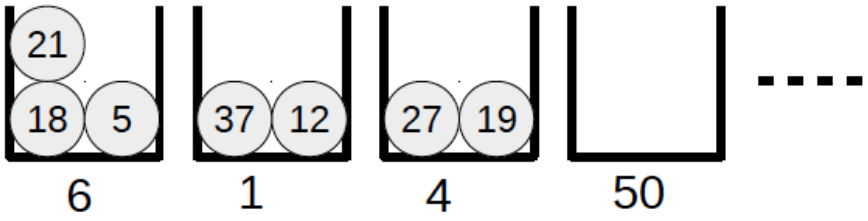
Il successivo oggetto, di peso 27, non può essere inserito né nel primo né nel secondo contenitore, quindi si inizia a riempire il terzo:



Il successivo oggetto di peso 12 trova posto nel secondo contenitore:



Infine, l'ultimo oggetto, di peso 19, trova posto nel terzo contenitore



Quindi, in questo caso dobbiamo utilizzare tre contenitori: il primo contiene oggetti per un peso complessivo di (21 + 18 + 5) = 44, il secondo contiene oggetti per un peso complessivo di 37 + 12 = 49, e il terzo contiene oggetti per un peso complessivo di 27 + 19 = 46.

Input:
Il programma accetta come unico parametro sulla riga di comando il nome di un file di input, che è in formato testo. Il file di input contiene i seguenti elementi:

- La prima riga contiene la capacità *C* dei contenitori (reale positivo);
- La seconda riga contiene il numero *n* di oggetti (intero positivo);
- Seguono *n* valori reali positivi, uno per riga, che rappresentano i pesi degli *n* oggetti.

Si utilizzi il tipo double per memorizzare i valori reali.

Output:
La prima riga dell'output deve contenere il numero *k* di contenitori utilizzati. Seguono *k* righe, rappresentanti il peso complessivo assieme agli identificatori/id degli oggetti in ciascun contenitore.

Esempio:

Input:	*Output:*
50.0 7 18.0 5.0 37.0 21.0 27.0 12.0 19.0	3 44.0, 3, 0, 1 49.0, 2, 5 46.0, 4, 6

Nota: *l'uso di operazioni in virgola mobile potrebbe portare a risultati leggermente diversi, per via degli inevitabili problemi di rappresentazione. In sede di valutazione tali differenze verranno ignorate.*
Questo esercizio ha una soluzione base piuttosto semplice che verrà accettata se realizzata correttamente, ma riceverà una valutazione bassa; esistono però soluzioni più efficienti che si basano su strutture dati ad hoc.

Esercizio 3.

Consideriamo una rete di telecomunicazione chiamata **Atlanta**, che è composta da **15 nodi** e **22 link bidirezionali/edges** come illustrato nella seguente figura.

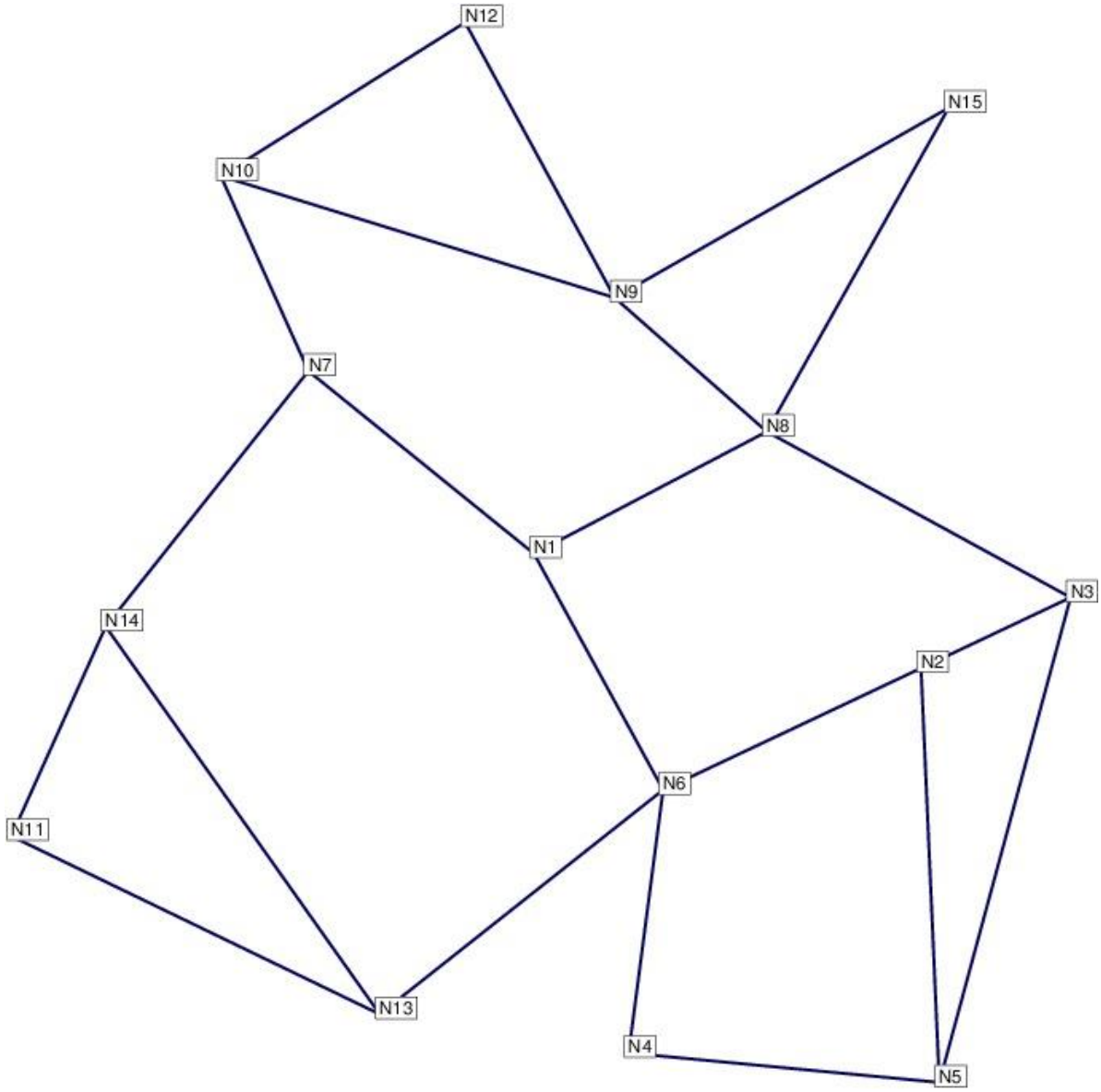


Figura 1: Atlanta network topology
(http://sndlib.zib.de/home.action?show=/problems.overview.action%3Fframeset)

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 atlanta

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

NODES (
  N1 ( 283.00 248.00 )
  N2 ( 451.00 201.00 )
  N3 ( 516.00 230.00 )
  N4 ( 324.00 43.00 )
  N5 ( 459.00 31.00 )
  N6 ( 339.00 151.00 )
  N7 ( 185.00 323.00 )
  N8 ( 384.00 298.00 )
  N9 ( 318.00 353.00 )
  N10 ( 147.00 403.00 )
  N11 ( 56.00 132.00 )
  N12 ( 253.00 466.00 )
  N13 ( 216.00 59.00 )
  N14 ( 97.00 218.00 )
  N15 ( 463.00 431.00 )
)

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

LINKS (
  L1 ( N1 N6 ) 11000.00 0.00 67.80 0.00 ( 1000.00 950000.00 4000.00 1090000.00 )
  L2 ( N1 N7 ) 7000.00 0.00 67.80 0.00 ( 1000.00 875000.00 4000.00 1015000.00 )
  L3 ( N1 N8 ) 4000.00 0.00 67.80 0.00 ( 1000.00 1010000.00 4000.00 1150000.00 )
  L4 ( N2 N3 ) 8000.00 0.00 67.80 0.00 ( 1000.00 980000.00 4000.00 1120000.00 )
  L5 ( N2 N5 ) 1000.00 0.00 67.80 0.00 ( 1000.00 1790000.00 4000.00 1930000.00 )
  L6 ( N2 N6 ) 15000.00 0.00 67.80 0.00 ( 1000.00 575000.00 4000.00 715000.00 )
  L7 ( N3 N5 ) 5.00 0.00 67.80 0.00 ( 1000.00 2060000.00 4000.00 2200000.00 )
  L8 ( N3 N8 ) 8000.00 0.00 67.80 0.00 ( 1000.00 875000.00 4000.00 1015000.00 )
  L9 ( N4 N5 ) 5.00 0.00 67.80 0.00 ( 1000.00 1295000.00 4000.00 1435000.00 )
  L10 ( N4 N6 ) 4000.00 0.00 67.80 0.00 ( 1000.00 1925000.00 4000.00 2065000.00 )
  L11 ( N6 N13 ) 2000.00 0.00 67.80 0.00 ( 1000.00 3200000.00 4000.00 3340000.00 )
  L12 ( N7 N10 ) 5000.00 0.00 67.80 0.00 ( 1000.00 1145000.00 4000.00 1285000.00 )
  L13 ( N7 N14 ) 5000.00 0.00 67.80 0.00 ( 1000.00 1445000.00 4000.00 1585000.00 )
  L14 ( N8 N9 ) 3000.00 0.00 67.80 0.00 ( 1000.00 1115000.00 4000.00 1255000.00 )
  L15 ( N8 N15 ) 3000.00 0.00 67.80 0.00 ( 1000.00 1190000.00 4000.00 1330000.00 )
  L16 ( N9 N10 ) 2000.00 0.00 67.80 0.00 ( 1000.00 935000.00 4000.00 1075000.00 )
  L17 ( N9 N12 ) 3000.00 0.00 67.80 0.00 ( 1000.00 1130000.00 4000.00 1270000.00 )
  L18 ( N9 N15 ) 1000.00 0.00 67.80 0.00 ( 1000.00 1070000.00 4000.00 1210000.00 )
  L19 ( N10 N12 ) 5.00 0.00 67.80 0.00 ( 1000.00 1505000.00 4000.00 1645000.00 )
  L20 ( N11 N13 ) 1000.00 0.00 67.80 0.00 ( 1000.00 1085000.00 4000.00 1225000.00 )
  L21 ( N11 N14 ) 1000.00 0.00 67.80 0.00 ( 1000.00 1700000.00 4000.00 1840000.00 )
  L22 ( N13 N14 ) 1000.00 0.00 67.80 0.00 ( 1000.00 830000.00 4000.00 970000.00 )
)

…

)
```

Per semplicità andiamo ad associare ai nodi un *id* intero che va da 0 a 14 al posto degli id N1, N2, …, N15.

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 **tutti i cammini di costo minimo** fra il nodo **N4** ed il nodo **N12**.
Il ***peso*** associato ad un link/edge *(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 Atlanta, w_ij = max_(pre_installed_capacity di tutti i link)/pre_installed_capacity sul link *(i,j)*.
Se il peso ***w_ij*** di un link risulta infinito, l'algoritmo di calcolo dei cammini minimi dovrà provvedere a scartare quel link perché il suo costo è eccessivamente alto/infinito.

1) Si richiede di usare in questo esercizio una implementazione dell'algoritmo di Dijkstra che si basa sull'utilizzo delle LISTE durante il calcolo dei cammini minimi. Il programma deve stampare a video:
   - TUTTI i cammini di costo minimo fra il nodo sorgente N4 ed il nodo destinazione N12, ed il costo totale del cammino minimo (*si nota che possono esistere <u>più di un</u> cammino di costo minimo fra una coppia di nodi*).
   - il tempo totale in secondi per trovare la soluzione.

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

Esercizio 4.

Riprendiamo la stessa rete (lo stesso grafo) dell'esercizio precedente (Esercizio 3) con le stesse specifiche. Si richiede in questo esercizio di calcolare tutti i cammini di costo minimo fra il nodo N4 ed il nodo N12 con una implementazione dell'algoritmo di Dijkstra che utilizza una *coda di priorità* per il calcolo dei cammini minimi. Anche in questo esercizio, il programma deve stampare a video le seguenti informazioni: **i)** tutti i cammini di costo minimo fra il nodo N4 ed il nodo N12, ed il costo totale del cammino minimo. **ii)** Il tempo totale in secondi per calcolare la soluzione.
Inoltre, si richiede di calcolare il costo computazionale totale del programma in funzione del numero di nodi e di link e di discutere sulla differenza/efficienza tra questa versione dell'algoritmo e quella determinata nell'esercizio precedente.