Il linguaggio GNU MathProg
con esempi commentati e risolti

Roberto Cordone

6 agosto 2010

1 Linguaggi algebrici di modellizzazione

Gli indubbi vantaggi del solver di Excel nell'affrontare problemi decisionali (la disponibilità universale dello strumento, la sua facilità d'uso, l'immediatezza visiva con cui permette di organizzare i dati) non possono far dimenticare i suoi limiti.

In un foglio elettronico non si vede il modello, ma solo i numeri, ed è forte la tentazione di saltare la modellizzazione per passare direttamente alla risoluzione del problema. D'altra parte, se il problema non si limita a riguardare vettori e matrici, ma coinvolge dati più sofisticati (matrici a tre o più dimensioni, insiemi, collezioni di insiemi, ecc. . .), può diventare complesso rappresentarlo con Excel. E ancora, il risolutore di Excel è adatto a risolvere solo problemi di piccole dimensioni, dato che i suoi tempi di calcolo crescono rapidamente con la dimensione del problema proposto. Infine, se si affrontano problemi diversi, ma con la stessa struttura di fondo (per dirla tecnicamente, istanze diverse dello stesso problema), l'uso di Excel richiede di costruire ogni volta da zero un nuovo foglio. Lo stesso avviene se si vuole affrontare esattamente lo stesso problema, ma con modelli diversi, per confrontarne l'efficacia.

Per ovviare a tutte queste limitazioni si può ricorrere a

- *linguaggi algebrici di modellazione*, vale a dire linguaggi che consentono di descrivere un modello decisionale in modo comprensibile a un essere umano, ma anche formalmente strutturato, cioè accessibile a un risolutore
- *generatori algebrici di modelli*, cioè strumenti software che consentono di convertire modelli scritti in un linguaggio algebrico nelle strutture dati richieste da un risolutore

Usando un generatore di modelli, l'utente può concentrarsi sulla stesura del modello, usando un linguaggio di alto livello estremamente simile a quello matematico (in modo da renderne l'apprendimento semplice e immediato). La comunicazione con il risolutore non avviene scrivendo i dati nelle celle di un foglio elettronico (o addirittura riempiendo strutture dati in memoria, come deve fare chi usa direttamente un risolutore), ma scrivendo due file di testo:

uno contiene il modello vero e proprio (le variabili di decisione, la funzione obiettivo, i vincoli), l'altro contiene i valori specifici dei dati. La separazione tra la struttura logica del modello e i valori numerici permette di "riciclare" il modello usandolo su dati diversi e di "riciclare" i dati, applicandovi modelli alternativi dello stesso problema. Un altro vantaggio dei generatori algebrici di modelli è che molti consentono di applicare risolutori diversi.

Nel seguito introduciamo uno di questi linguaggi, *GNU MathProg*, che ha il vantaggio di disporre di un generatore algebrico scaricabile gratuitamente da Internet, alla pagina `http://gnuwin32.sourceforge.net/packages/glpk.htm`. Il generatore algebrico si chiama *GLPK* (GNU Linear Programming Kit) e comprende un risolutore gratuito di problemi di Programmazione Lineare e di Programmazione Lineare Intera. Si tratta di un sottoinsieme del linguaggio *AMPL*, la cui principale limitazione sta nel non prevedere la modellazione e risoluzione di problemi di Programmazione Non Lineare. La documentazione scaricabile insieme al programma descrive molto bene (in inglese) il linguaggio e l'uso del generatore algebrico. Ci limiteremo quindi a introdurre questo strumento con alcuni esempi (tratti dai capitoli 2 e 3), che saranno sufficienti a mettere in grado l'utente di sperimentare con modelli anche di una certa complessità.

Prima di far ciò, elenchiamo rapidamente alcune caratteristiche generali della sintassi di *GNU MathProg*:

- è un linguaggio *case sensitive*, cioè distingue le lettere maiuscole dalla minuscole (il buon senso suggerisce di non sfruttare questa proprietà per evitare di far confusione);
- le righe del programma hanno una lunghezza massima di 255 caratteri (quelli eccedenti vengono ignorati);
- i commenti sono introdotti dal simbolo `#`, che indica al generatore di ignorare il seguito della riga (è consigliabile fare largo uso di commenti);
- le diverse istruzioni del programma sono separate dal simbolo `;`
- "a capo" e tabulazioni vengono interpretati come semplici spazi bianchi (è bene sfruttare questa proprietà per dare al modello un formato più leggibile)

2 La sintassi di *GNU MathProg* attraverso un esempio

Consideriamo il problema n. 2 di Programmazione Lineare cioè il problema della dieta. Il suo modello matematico è:

$$\begin{aligned}
\min z = & \sum_{j=1}^{A} c_j x_j & \\
\text{s.t.} & \sum_{j=1}^{A} a_{ij} x_j \geq l_i & \forall i = 1, \ldots, S \\
& \sum_{j=1}^{A} a_{ij} x_j \leq u_i & \forall i = 1, \ldots, S \\
& x_j \geq 0 & \forall j = 1, \ldots, A.
\end{aligned}$$

e comprendeva

1. un insieme di $S = 3$ *sostanze* (corrispondente all'indice $i = 1, \ldots, S$) e un insieme di $A = 7$ *alimenti* (corrispondente all'indice $j = 1, \ldots, A$)
2. due vettori di dati numerici l_i ed u_i che indicano il *limite inferiore* e il *limite superiore* da assumere ogni giorno per ogni sostanza $i = 1, \ldots, S$ (si dice che questi vettori sono *indicizzati sull'insieme delle sostanze*), nonché un vettore di *prezzi* c_j indicizzato sull'insieme degli alimenti ($j = 1, \ldots, A$) e una matrice a due dimensioni a_{ij}, che fornisce il *contenuto* per ogni sostanza $i = 1, \ldots, S$ in ogni unità di alimento $j = 1, \ldots, A$ ed è quindi indicizzata su entrambi gli insiemi;
3. un vettore di variabili x_j (tutte continue e non negative) indicizzato sull'insieme degli alimenti;
4. una funzione obiettivo z, pari al *costo complessivo*;
5. due famiglie di vincoli indicizzate sulle sostanze ($i = 1, \ldots, S$), che impongono (rispettivamente) di rispettare il limite inferiore e quello superiore da assumere ogni giorno.

Tutti i modelli decisionali hanno la stessa struttura, cioè comprendono:

1. alcuni *insiemi*, sui quali scorrono degli indici;
2. dei *dati numerici* (di valore noto), che possono essere semplici numeri scalari, oppure vettori (indicizzati su un insieme) o matrici a due o più dimensioni (indicizzate su due o più insiemi);
3. delle *variabili* (il cui valore va determinato), che possono essere semplici numeri scalari, oppure vettori (indicizzati su un insieme) o matrici a due o più dimensioni (indicizzate su due o più insiemi);

4. una *funzione obiettivo* z, che dipende dai dati e dalle variabili e può essere minimizzata o massimizzata;

5. un certo numero di *vincoli*, di uguaglianza (equazioni) o disuguaglianza (disequazioni), singoli oppure raccolti in famiglie indicizzate su uno o più insiemi.

Una volta chiaro questo punto, stendere il modello *GNU MathProg* è cosa banale. Sebbene sia lecito raccogliere il modello e i dati in un solo file di testo, è concettualmente preferibile tenere separati questi due oggetti. Cominciamo con il file relativo al modello, per passare poi a quello relativo ai dati.

2.1 Il file modello

Il file modello ha di solito estensione `.mod` (è una convenzione, non un obbligo). Si apre con l'indicazione degli insiemi usati nel modello, ciascuno introdotto dalla parola chiave `set` e terminato dal punto e virgola:

```
set Sostanze;   # insieme delle sostanze rilevanti

set Alimenti;   # insieme degli alimenti disponibili
```

Per il momento, non andiamo a definire l'elenco dettagliato delle `Sostanze` e degli `Alimenti`, perché il file modello fornisce la struttura astratta del problema, mentre i valori specifici saranno forniti nell'altro file. Si noti la presenza dei commenti, introdotti dal simbolo `#`.

Seguono i dati, ciascuno introdotto dalla parola chiave `param` e terminato dal punto e virgola:

```
# fabbisogno minimo per ogni sostanza [g/giorno]
param LimiteInferiore {Sostanze} >= 0;

# fabbisogno massimo per ogni sostanza [g/giorno]
param LimiteSuperiore {Sostanze} >= 0;

# prezzo per ogni alimento [Euro/kg]
param Prezzo {Alimenti} >= 0;

# contenuto per ogni sostanza di ogni alimento [g/kg]
param Contenuto {Sostanze,Alimenti} >= 0;
```

Si noti che ogni vettore è seguito dall'indicazione (fra parentesi graffe) dell'insieme sul quale è indicizzato: esiste un limite inferiore e un limite superiore per ogni sostanza; esiste un prezzo per ogni alimento. Per le matrici, vale la stessa considerazione, salvo che fra parentesi graffe compaiono i due insiemi sui quali la matrice è indicizzata, separati da una virgola: esiste un contenuto per ogni sostanza e per ogni alimento; le righe corrispondono alle sostanze, le colonne agli alimenti. Eventuali matrici a più dimensioni avrebbero fra parentesi graffe più

di due insiemi, sempre separati da virgole. Un semplice dato numerico scalare, invece, non sarebbe seguito da alcun insieme fra parentesi graffe.

Notiamo anche le *istruzioni di controllo* `>= 0`. Queste indicano al generatore algebrico di verificare che i valori numerici dei dati, specificati nell'altro file, rispettino opportune condizioni (in questo caso, di non essere negativi). Nel caso non li rispettino, il generatore segnala un errore senza neppure chiamare il risolutore.

Segue la sezione dedicata alle variabili, ciascuna introdotta dalla parola chiave `var` e terminata dal punto e virgola:

```
# Porzioni giornaliere di ogni alimento [kg]
var x {Alimenti} >= 0;
```

Anche il vettore delle variabili è indicizzato su un insieme (quello degli `Alimenti`). La condizione `>= 0` qui ha un significato diverso dalla precedente: si tratta non di una precondizione da imporre su dati noti a priori, ma di un vero e proprio vincolo, che le variabili (inizialmente incognite) dovranno rispettare. Come tale, potremmo inserirlo nella sezione dei vincoli, più oltre; tuttavia, trattandosi di vincoli molto semplici e che limitano direttamente il valore di una singola variabile, è uso comune definirli in questa posizione. Non è necessario specificare che le variabili sono continue, mentre se fossero intere o binarie sarebbe necessario indicarlo qui (vedi gli esempi successivi).

Si passa poi alla funzione obiettivo, introdotta dalla parola chiave `minimize` (oppure `maximize`, secondo il tipo di problema), seguita da `:`, dall'espressione algebrica che la definisce e terminata dal solito punto e virgola:

```
minimize z : sum{j in Alimenti} Prezzo[j] * x[j];
```

Nell'espressione algebrica, il singolo elemento di un vettore viene rappresentato col nome del vettore (`Prezzo` o `x`), seguito da un indice fra parentesi quadre. La sommatoria viene resa con l'istruzione `sum`, seguita fra parentesi graffe dall'insieme sul quale si fa la somma; si noti che qui viene anche specificato l'indice `j` della sommatoria, dato che tale indice si deve ritrovare identico nell'espressione sommata (`Prezzo[j] * x[j]`).

L'ultima sezione del file modello è dedicata ai vincoli. Ogni famiglia di vincoli è introdotta dalla parola chiave `subject to`, seguita da un nome simbolico e dall'insieme sul quale è indicizzato il vincolo stesso, da un due punti (`:`) e dall'espressione del vincolo e terminata dal solito punto e virgola.

```
subject to VincoloInferiore {i in Sostanze} :
  sum{j in Alimenti} Contenuto[i,j] * x[j] >= LimiteInferiore[i];

subject to VincoloSuperiore {i in Sostanze} :
  sum{j in Alimenti} Contenuto[i,j] * x[j] >= LimiteSuperiore[i];
```

Si noti il fatto che, oltre all'insieme sul quale è indicizzato il vincolo (`Sostanze`), viene anche specificato l'indice `i`, dato che tale indice si deve ritrovare identico nell'espressione del vincolo e deve rimanere intatto dalla sommatoria (mentre l'indice `j` si semplifica nella sommatoria sull'insieme degli `Alimenti`).

Il file modello si chiude con la parola chiave `end` e col punto e virgola.

2.2 Il file dati

Il file dati ha di solito estensione `.dat` (è una convenzione, non un obbligo) e ripropone le prime due sezioni del file modello, cioè quelle degli insiemi e dei dati numerici.

Ciascun insieme è introdotto dalla parola chiave `set`, cui segue `:=` e l'elenco degli elementi dell'insieme, terminato dal punto e virgola:

```
set Sostanze :=   # insieme delle sostanze rilevanti
  Proteine
  Carboidrati
  Grassi
;

set Alimenti :=   # insieme degli alimenti disponibili
  Pasta Latte Formaggio Pesce Verdura Pane Polenta ;
```

Si noti il fatto che i commenti possono essere inseriti ovunque nel testo, pur di ricordare che dopo il simbolo `#` l'intera riga viene considerata un commento e ignorata, e il fatto che gli "a capo", le tabulazioni e gli spazi possono essere usati a piacere per dare la forma desiderata al testo e renderlo più leggibile.

Seguono i dati, ciascuno introdotto dalla parola chiave `param`, cui segue `:=` e l'elenco delle coppie *elemento-valore*, terminato dal punto e virgola:

```
param LimiteInferiore :=   # fabbisogno minimo
  Proteine     25          # per ogni sostanza [g/giorno]
  Carboidrati  15
  Grassi       10
;

param LimiteSuperiore :=   # fabbisogno massimo
  Proteine     35          # per ogni sostanza [g/giorno]
  Carboidrati  25
  Grassi       20
;

param Prezzo  := # prezzo per ogni alimento [Euro/kg]
  Pasta         4
  Latte         4
  Formaggio    15
  Pesce        22.5
  Verdura       3
  Pane          1
  Polenta       5
;
```

Si noti l'uso degli "a capo" e degli incolonnamenti per rendere più leggibile il testo: in realtà è sufficiente che elementi e valori si alternino a coppie (anche in ordine diverso da quello con cui sono stati definiti in precedenza gli elementi).

Una matrice a due o più dimensioni viene specificata ancora elencando le coppie *elemento-valore*, ma occorre tener conto che un elemento di una matrice è una casella, e quindi corrisponde a sua volta a una coppia (tripletta, o in generale n-upla) di elementi. Per separare l'elemento dal valore, si raccoglie la n-upla che descrive ogni elemento della matrice fra parentesi quadre, come segue:

```
# contenuto per ogni sostanza di ogni alimento [g/kg]
param Contenuto :=
  [Proteine,Pasta]         11.5  [Proteine,Latte]     3.15
  [Proteine,Formaggio]      8    [Proteine,Pesce]    18.5
  [Proteine,Verdura]        2.1  [Proteine,Pane]     12.0
  [Proteine,Polenta]        9
  [Carboidrati,Pasta]      72.7  [Carboidrati,Latte]  4.85
  [Carboidrati,Formaggio]   3.8  [Carboidrati,Pesce]  0.5
  [Carboidrati,Verdura]     0    [Carboidrati,Pane]  68
  [Carboidrati,Polenta]    74
  [Grassi,Pasta]            1.5  [Grassi,Latte]       1.55
  [Grassi,Formaggio]       11    [Grassi,Pesce]      19
  [Grassi,Verdura]          0.1  [Grassi,Pane]        6
  [Grassi,Polenta]          1
;
```

Infine, per un semplice dato numerico scalare il simbolo `:=` sarebbe ovviamente seguito solo dal valore e dal punto e virgola.

Anche il file dati si chiude con la parola chiave `end` e col punto e virgola.

Giunti a questo punto, dovrebbe risultare chiaro che:

1. scrivere un modello *GNU MathProg* una volta steso il modello matematico è quasi banale
2. conservare in due file distinti il modello e i dati consente di
   (a) risolvere istanze diverse dello stesso problema cambiando il file dei dati e conservando quello del modello
   (b) risolvere la stessa istanza di un problema con modelli diversi cambiando il file del modello e conservando quello dei dati

3 Uso di *GNU MathProg*

Una volta stesi i file per il modello e per i dati, è possibile richiamare il generatore algebrico (e automaticamente subito dopo il risolutore) con la seguente linea di comando:

[Directory]`glpsol -m` *[Modello]* `-d` *[Dati]*

dove *[Directory]* indica la directory in cui è presente il file `glpsol.exe`, mentre *[Modello]* e *[Dati]* sono, rispettivamente, il nome del file del modello e di quello dei dati. Ad esempio:

```
C:\Programmi\Glpk\bin\glpsol -m Dieta.mod -d Dieta.dat
```

Il generatore algebrico esegue un controllo di correttezza sulla sintassi dei due file, segnalando eventuali errori. Nel caso non vi siano errori, lancia il risolutore, che risolve il problema (in un tempo più o meno lungo, secondo la sua dimensione e difficoltà) e fornisce il valore ottimo della soluzione. Per conoscere la soluzione per esteso, occorre aggiungere l'ulteriore opzione `-o`:

[Directory]`glpsol -m` *[Modello]* `-d` *[Dati]* `-o` *[Soluzione]*

dove *[Soluzione]* indica il nome del file di testo in cui si vuole salvare la soluzione stessa.

Nel caso si volesse invece solo verificare la correttezza del modello, senza risolverlo, è sufficiente aggiungere l'opzione `--check`

[Directory]`glpsol -m` *[Modello]* `-d` *[Dati]* `--check`

Per le altre opzioni disponibili, rimandiamo alla documentazione e all'help consultabile con l'opzione `-h`

[Directory]`glpsol -h`

4 Interpretazione dell'output

Supponiamo di aver risolto con *GLPK* il problema della dieta con i due file costruiti più sopra e di aver salvato la soluzione su un file di testo, come descritto nella sezione precedente. Il file si compone di tre parti principali.

La prima parte descrive il problema (specificando il numero di righe, di colonne e di coefficienti non nulli), indica se sia ammissibile, illimitato oppure risolto all'ottimo e infine fornisce il valore ottimo della funzione obiettivo. Nel caso di problemi di Programmazione Lineare Intera, fornisce anche il valore del rilassamento lineare, vale a dire del corrispondente problema di Programmazione Lineare che si ottiene ignorando i vincoli di interezza sulle variabili. Nel caso del problema della dieta, il file della soluzione comincia come segue:

```
Problem:    Dieta
Rows:       7
Columns:    7
Non-zeros:  47
Status:     OPTIMAL
Objective:  z = 26.23620095 (MINimum)
```

Segue una sezione dedicata alla funzione obiettivo e ai vincoli, cioè alle righe del modello. Di ogni riga vengono fornite 7 informazioni:

1. `No.`: un numero progressivo;
2. `Row name`: il nome simbolico;
3. `St`: lo *stato* della variabile di *slack* associata al vincolo (`B` se in base, `NL` se fuori base e pari al suo valore minimo, `NU` se fuori base e pari al suo valore massimo);
4. `Activity`: il suo valore ottimo;
5. `Lower bound`: il suo valore minimo ammissibile;
6. `Upper bound`: il suo valore massimo ammissibile;
7. `Marginal`: il suo *prezzo ombra*, di cui diremo qualcosa in seguito.

Infine, si ha una sezione dedicata alle variabili, cioè alle colonne del modello. Di ogni colonna vengono fornite 7 informazioni:

1. `No.`: un numero progressivo;
2. `Column name`: il nome simbolico;
3. `St`: lo *stato* della variabile (`B` se in base, `NL` se fuori base e pari al suo valore minimo, `NU` se fuori base e pari al suo valore massimo);
4. `Activity`: il suo valore ottimo;
5. `Lower bound`: il suo valore minimo ammissibile;
6. `Upper bound`: il suo valore massimo ammissibile;
7. `Marginal`: il suo *costo ridotto*, di cui diremo qualcosa in seguito.

Quindi, la soluzione ottima del problema si ottiene consultando la colonna `Activity` nella sezione relativa alle variabili.

D'altra parte, anche le altre informazioni sono piuttosto interessanti e meritano alcune parole. Infatti esse consentono non solo di calcolare la soluzione ottima del problema, ma anche di valutarne la stabilità e la sensibilità a (piccole) variazioni dei dati. Questa è un'informazione utilissima, perché molto spesso i dati non sono davvero noti esattamente, ma sono semplici stime e scegliere una soluzione sulla base di una stima potrebbe esporre a gravi sorprese, anche nel caso di piccoli errori di stima. In altri casi, invece, la soluzione ottima potrebbe essere insensibile o quasi a errori nella stima dei dati. È quindi molto utile essere in grado di valutare quanto cambierebbe la soluzione ottima nel caso i dati dovessere cambiare rispetto alle stime usate per risolvere il problema.

4.1 Studio di sensitività

Queste informazioni si ottengono con uno studio di *sensitività*, che consiste nel consultare la colonna `Marginal` nelle sezioni relative ai vincoli e alle variabili nel file della soluzione.

Analisi dei vincoli (prezzi ombra)

```
No. Row name St Activity Lower bound Upper bound Marginal
--- -------- -- -------- ----------- ----------- ---------
  1 z        B   26.2362
  2 VincoloInferiore[Proteine]
             NL       25          25               1.43861
  3 VincoloInferiore[Carboidrati]
             B        25          15
  4 VincoloInferiore[Grassi]
             B        20          10
  5 VincoloSuperiore[Proteine]
             B        25                      35
  6 VincoloSuperiore[Carboidrati]
             NU       25                      25 -0.220572
  7 VincoloSuperiore[Grassi]
             NU       20                      20 -0.210733
```

Per quanto riguarda i vincoli, il valore del prezzo ombra per un vincolo è il *rapporto fra la variazione della funzione obiettivo e quella del termine noto* del vincolo stesso, purché si tratti di *variazioni "piccole"*, cioè inferiori a una soglia che definiremo in seguito. Nell'esempio considerato, il prezzo ombra del limite inferiore per i `Carboidrati` è nullo. Questo indica che, per piccole variazioni di tale limite, il valore ottimo della funzione obiettivo non cambia. Infatti, la soluzione ottima comporta un'assunzione di 25 g/giorno di carboidrati, che è lontana dal limite inferiore (15 g/giorno). Quindi, eventuali modifiche di tale limite non impattano sulla soluzione ottima, purché non siano molto intense (almeno 10 g/giorno in più). D'altra parte, il prezzo ombra del vincolo inferiore sulle proteine è pari a 1.43861. Questo indica che un piccolo aumento nel limite inferiore (per esempio da 25 g/giorno a 25.1 g/giorno) comporta un proporzionale aumento nella funzione obiettivo: da 26.2362 a $26.2362 + 0.1 \cdot 1.43861 = 26.3801$ Euro/giorno. In effetti, la soluzione ottima comporta un'assunzione di 25 g/giorno di proteine, che coincide con il limite inferiore e quindi è probabilmente dovuto alla presenza di tale limite, anziché all'obiettivo di ottimizzare la dieta. Basta costruire un file di dati modificato per verificare che questo è effettivamente quello che avviene.

D'altra parte, il prezzo ombra del vincolo superiore sui carboidrati è pari a -0.220572. Questo indica che un piccolo aumento nel limite superiore (per esempio da 25 g/giorno a 26 g/giorno) comporta un proporzionale calo nella funzione obiettivo, da 26.2362 a $26.2362+1\cdot(-0.220572) = 26.0156$ Euro/giorno. Basta costruire un file di dati modificato per verificare anche questo risultato. Si noti come i vincoli la cui variabile di *slack* risulta in base (`B` nella colonna `St`) hanno prezzo ombra nullo (perché sono soddisfatti con un ampio margine, e quindi insensibili alle variazioni del termine noto), mentre i vincoli con prezzo ombra non nullo hanno la variabile di *slack* fuori base (`NL` o `NU` nella colonna `St`), perché per essere sensibili alle variazioni del termine noto devono essere soddisfatti esattamente, con margine nullo.

Analisi delle variabili (costi ridotti)

```
No. Column name  St Activity Lower bound Upper bound Marginal
--- ------------ -- -------- ----------- ----------- --------
  1 x[Pasta]     NL        0           0              3.80769
  2 x[Latte]     NL        0           0             0.864799
  3 x[Formaggio] NL        0           0              6.64739
  4 x[Pesce]     B  0.930007           0
  5 x[Verdura]   B   1.65008           0
  6 x[Pane]      B  0.360809           0
  7 x[Polenta]   NL        0           0              8.58558
```

Il ragionamento è analogo per quanto riguarda le variabili. Il valore del costo ridotto per una variabile è il *rapporto fra la variazione della funzione obiettivo e quella del limite inferiore o superiore sulla variabile*, purché si tratti di *variazioni "piccole"*, cioè inferiori a una soglia che definiremo in seguito. Le variabili associate a `Pesce`, `Verdura` e `Pane` sono in base (`B` nella colonna `St`) e hanno costo ridotto nullo. Questo corrisponde al fatto che, se aumentassimo leggermente il loro limite inferiore (al momento nullo), il valore ottimo della funzione obiettivo non ne sarebbe influenzato. Infatti, tutte e tre le variabili hanno valore strettamente positivo, con un certo margine rispetto al loro limite inferiore. Invece, le altre variabili sono fuori base al loro valore minimo (`NL` nella colonna `St`) e hanno costo ridotto positivo. In particolare, il costo ridotto della `Pasta` è pari a 3.80769, cioè forzare la presenza di una piccola quantità di `Pasta` (ad esempio almeno 0.1 kg/giorno) nella dieta porterebbe a un aumento proporzionale nel valore ottimo della funzione obiettivo, che passerebbe da 26.2362 Euro/giorno a $26.2362 + 0.1 \cdot 3.80769 = 26.6170$ Euro/giorno. Basta costruire un file di modello modificato con l'aggiunta del vincolo

```
subject to VincoloPasta :
  x["Pasta"] >= 0.1;
```

per verificare questo risultato.

4.2 Studio di postottimalità

Rimane il punto oscuro di quanto possano essere ampie le variazioni affinché queste proprietà siano valide. Esiste una variazione massima (o minima) oltre la quale il comportamento del valore ottimo della funzione obiettivo non è più prevedibile conoscendo i prezzi ombra e i costi ridotti. È possibile conoscere tali limiti richiedendo al risolutore anche uno studio di *postottimalità*. Tale studio richiede di aggiungere l'ulteriore opzione `--bounds` alla chiamata del risolutore:

[Directory]`glpsol -m` *[Modello]* `-d` *[Dati]* `-o` *[Soluzione]* `--bounds` *[Bounds]*

dove *[Bounds]* indica il nome del file di testo in cui si vuole salvare lo studio stesso. Tale file si compone di tre parti principali, che indicano *entro quali*

intervalli di variazione dei coefficienti, termini noti e limiti sulle variabili la struttura della soluzione rimane invariata. Sottolineiamo la parola "struttura": facendo variare i coefficienti negli intervalli indicati da questa analisi, la soluzione ottima e il suo valore cambiano, ma rimane identica la *base* ottima, cioè nessuna variabile entra o esce dalla base stessa. Si noti anche che l'analisi considera un coefficiente alla volta, cioè suppone che tutti gli altri coefficienti rimangano invariati: è chiaro che se cambiano insieme due o più coefficienti, anche se rimangono negli intervalli forniti dallo studio, l'effetto combinato delle variazioni può modificare la struttura della base ottima.

Analisi dei coefficienti della funzione obiettivo

```
Objective Coefficient Analysis
   No.  Column name St    Value        Max increase  Max decrease
------ ------------ -- ------------- ------------- -------------
     1 x[Pasta]     NL             4      infinity       3.80769
     2 x[Latte]     NL             4      infinity      0.864799
     3 x[Formaggio] NL            15      infinity       6.64739
     4 x[Pesce]     B           22.5       3.80987       14.1897
     5 x[Verdura]   B             3       1.43829      0.434545
     6 x[Pane]      B             1       3.55497       518.143
     7 x[Polenta]   NL            5      infinity       8.58558
```

La prima parte riguarda i coefficienti della funzione obiettivo. Per ogni coefficiente si riportano sei informazioni:

1. `No.`: un numero progressivo;
2. `Column name`: il nome simbolico della variabile associata;
3. `St`: lo *stato* della variabile associata (`B` se in base, `NL` se fuori base e pari al suo valore minimo, `NU` se fuori base e pari al suo valore massimo);
4. `Value`: il valore iniziale del coefficiente;
5. `Max increase`: il massimo aumento che non porta a variazioni della base ottima;
6. `Max decrease`: il massimo calo che non porta a variazioni della base ottima.

Nel caso della dieta, ad esempio, il costo della `Pasta` può aumentare indefinitamente senza modificare la base ottima. Infatti, la `Pasta` non fa parte della dieta ottima, e aumentarne il prezzo non fa che confermare questa situazione. D'altra parte, diminuirne il prezzo non cambia le cose finché il calo non diventa > 3.80769 Euro/kg, cioè finché il prezzo non diventa $< 4 - 3.80769 = 0.19231$ Euro/kg. Basta costruire un file di dati con prezzi di poco inferiori o superiori a questa soglia per verificare che il risultato è corretto. Analogamente, il prezzo del `Pesce` può crescere di 3.80987 Euro/kg oppure calare di 14.1897 Euro/kg

senza modificare la base ottima, mentre variazioni più forti modificano la base stessa, facendo entrare e uscire variabili dalla base. Questo non significa che sia necessariamente la variabile associata al `Pesce` a uscire di base.

Analisi dei termini noti dei vincoli

No.	Row name	St	Value	Max increase	Max decrease
1	z	B			
2	VincoloInferiore[Proteine]	NL			
	LOWER		25	10	3.30489
3	VincoloInferiore[Carboidrati]	B			
	LOWER		15	10	infinity
4	VincoloInferiore[Grassi]	B			
	LOWER		10	10	infinity
5	VincoloSuperiore[Proteine]	B			
	UPPER		35	infinity	10
6	VincoloSuperiore[Carboidrati]	NU			
	UPPER		25	36.4103	10
7	VincoloSuperiore[Grassi]	NU			
	UPPER		20	3.40256	10

La seconda parte del file riguarda i termini noti dei vincoli. Per ogni vincolo si riportano le sei informazioni prima elencate. Nel problema della dieta, si può osservare che il limite inferiore sulla quantità di carboidrati può calare a piacere e crescere al massimo di 10 g/giorno senza modificare la base ottima. Questo è coerente con il fatto che la quantità di carboidrati assunta è pari a 25 g/giorno e il limite inferiore è pari a 15 g/giorno (colonne `Activity` e `Lower bound` nella sezione relativa alle variabili del file della soluzione).

Analisi dei limiti sulle variabili

Variable Bounds Analysis

No.	Column name	St	Value	Max increase	Max decrease
1	x[Pasta]	NL			
	LOWER		0	0.336861	3.46575
2	x[Latte]	NL			
	LOWER		0	2.74431	infinity
3	x[Formaggio]	NL			
	LOWER		0	1.62992	1.09102

```
4 x[Pesce]     B
    LOWER                  0      0.930007      infinity
5 x[Verdura]   B
    LOWER                  0       1.65008      infinity
6 x[Pane]      B
    LOWER                  0      0.360809      infinity
7 x[Polenta]   NL
    LOWER                  0      0.330894       3.15092
```

Infine, la terza parte del file riguarda i limiti inferiori sulle variabili, che nel caso della dieta sono tutti nulli. Per ogni variabile si riportano le solite sei informazioni. È ovvio che nel caso della `Verdura`, che compare con 1.65008 kg/giorno nella soluzione ottima, imporre un limite inferiore più basso non ha alcun effetto sulla base ottima, e neppure ne ha imporre un limite più alto, fino a che il limite stesso non diventa > 1.65008 kg/giorno. Da quel punto in poi, infatti, la `Verdura` comparirà nella dieta con il minimo valore ammissibile a causa del limite inferiore, e non perché sia conveniente includerla per ottimizzare il costo, dato che il valore ottimo sarebbe più basso. D'altra parte, nel caso del `Formaggio` (che non compare nella soluzione ottima), la base ottima rimane invariata finché il limite inferiore non diminuisce di un valore > 1.09102 kg/giorno (cosa che non ha alcun significato fisico, dato che corrisponderebbe a fornire dosi negative di `Formaggio`, ovvero prelevare dal corpo dei clienti le quantità corrispondenti alla dose negativa desiderata). Ma la base ottima rimane invariata anche aumentando il limite inferiore fino al valore di soglia 1.62992 kg/giorno. Infatti, è vero che imponendo un limite inferiore pari (ad esempio) a 1.6 kg/giorno, la dieta includerebbe una quantità positiva di `Formaggio`; tuttavia, la variabile corrispondente sarebbe pur sempre fuori base al suo valore minimo, cioè comparirebbe nella dieta solo a causa del vincolo. D'altra parte, per limiti più alti, la `Pasta` sostituisce il `Pesce` nella base ottima, che quindi cambia. Per verificarlo, è sufficiente costruire un file di modello modificato con l'aggiunta del vincolo:

```
subject to VincoloFormaggio :
  x["Formaggio"] >= 1.63;
```

5 Problemi commentati e risolti

Consideriamo ora brevemente alcuni altri problemi di Programmazione Lineare e Lineare Intera, scelti fra quelli descritti nelle raccolte di esercizi che trovate su questo sito.

5.1 Problema n.1: Il mix produttivo ottimale

Partendo dal modello matematico:

$$\begin{aligned} \max z &= \sum_{j=A}^{C} c_j x_j \\ \text{s.t.} \sum_{j=A}^{C} a_{ij} x_j &\leq b_i \qquad \forall i = 1, \ldots, 5 \\ x_j &\geq 0 \qquad \forall j = A, \ldots, C \end{aligned}$$

procediamo come sopra, indicando per prima cosa gli insiemi usati nel modello, che sono i `Reparti` e i `Modelli` di veicoli.

Quindi, passiamo ai dati: le `Capacità` costituiscono un vettore indicizzato sui `Reparti`, i `Profitti` un vettore indicizzato sui `Modelli`. I tempi di lavorazione pongono un piccolo problema: i reparti per le finiture sono specializzati sui singoli modelli, per cui non ha senso, a rigore, parlare di tempo di lavorazione per un modello da parte di un reparto dedicato alla finitura di un altro modello. Si può risolvere il problema in modo semplice, attribuendo ai reparti che non sono specializzati su un modello tempi particolarmente lunghi, in modo che le soluzioni che li impiegano (e che non sono ammissibili) risultino ammissibili, ma non ottime.

Un altro modo è più complesso, ma elegante, e consiste nel descrivere correttamente l'insieme delle combinazioni ammissibili. Introduciamo quindi un terzo insieme, di `Combinazioni`, che contiene le sole combinazioni ammissibili fra `Reparti` e `Modelli`. Questo insieme è un sottoinsieme delle coppie di elementi dell'insieme dei `Reparti` e dell'insieme dei `Modelli` e andrà specificato come tale nel file del modello attraverso la parola chiave `within` seguita dall'insieme dal quale si va ad estrarlo. Quest'ultimo è il prodotto cartesiano dell'insieme dei `Reparti` e di quello dei `Modelli` e viene indicato indicando fra parentesi graffe gli insiemi componenti, separati da virgole:

```
set Combinazioni within {Reparti,Modelli};
```

Ovviamente, questa riga dovrà seguire quelle che definiscono i due insiemi componenti.

D'altra parte, l'insieme stesso andrà esplicitamente definito nel file dei dati, elencandone gli elementi, come per tutti gli altri insiemi del problema. Gli elementi sono coppie e vengono riportate elencando i membri di ogni coppia fra parentesi tonde e separati da virgole.

```
set Combinazioni :=
(Motori,A)       (Motori,B)       (Motori,C)
(Carrozzeria,A)  (Carrozzeria,B)  (Carrozzeria,C)
(FinituraA,A)    (FinituraB,B)    (FinituraC,C)
;
```

A questo punto, siamo in grado di definire nel file del modello una matrice parziale di `Tempi`, indicizzata sull'insieme delle `Combinazioni`.

```
param Tempi {Combinazioni} ;
```

e di indicarne i valori numerici nel file dei dati, esattamente come si fa per le matrici complete:

```
param Tempi :=
[Motori,A]       3  [Motori,B]       2  [Motori,C]       1
[Carrozzeria,A]  1  [Carrozzeria,B]  2  [Carrozzeria,C]  3
[FinituraA,A]    2  [FinituraB,B]    3  [FinituraC,C]    2
;
```

L'ultimo aspetto degno di nota è il fatto che, a rigore, il vincolo $\sum_{j=A}^{C} a_{ij}x_j \leq b_i$ è corretto solo se si assegnano tempi di lavorazione molto elevati alle combinazioni inammissibili. Se invece si adotta l'approccio prima descritto, non è vero che la somma sull'indice `j` si estende su tutti i `Modelli`, perché quando `i` corrisponde a uno dei reparti di finitura, solo per uno dei tre modelli è definito un tempo di lavorazione. Quindi la sommatoria va limitata a quegli indici `j` per i quali la coppia `(i,j)` cade nelle `Combinazioni` ammissibili. Il vincolo va quindi scritto come segue:

```
subject to VincoloTempi {i in Reparti} :
  sum{j in Modelli: (i,j) in Combinazioni} Tempi[i,j] * x[j] <= Capacita[i];
```

restringendo l'insieme `Modelli` su cui scorre l'indice `j` con la condizione logica `: (i,j) in Combinazioni`.

Il resto del modello non presenta differenze rilevanti rispetto a quello della dieta. Concludendo, il file del modello è

```
# Mix produttivo ottimale

set Reparti;
set Modelli;
set Combinazioni within {Reparti,Modelli};

param Capacita {Reparti};
param Profitti {Modelli};
param Tempi {Combinazioni};

# Porzioni giornaliere di ogni alimento [kg]
var x {Modelli} >= 0;

maximize z : sum{j in Modelli} Profitti[j] * x[j];

subject to VincoloTempi {i in Reparti} :
  sum{j in Modelli: (i,j) in Combinazioni} Tempi[i,j] * x[j] <= Capacita[i];

end;
```

Mentre il file dei dati è

```
# Mix produttivo ottimale

set Reparti := Motori  Carrozzeria  FinituraA  FinituraB  FinituraC;
set Modelli := A B C;
set Combinazioni :=
(Motori,A)       (Motori,B)       (Motori,C)
(Carrozzeria,A)  (Carrozzeria,B)  (Carrozzeria,C)
(FinituraA,A)    (FinituraB,B)    (FinituraC,C)
;

param Capacita :=
Motori      120
Carrozzeria  80
FinituraA    96
FinituraB   102
FinituraC    40
;

param Profitti :=
A   840
B  1120
C  1200
;

param Tempi :=
[Motori,A]       3  [Motori,B]       2  [Motori,C]       1
[Carrozzeria,A]  1  [Carrozzeria,B]  2  [Carrozzeria,C]  3
[FinituraA,A]    2  [FinituraB,B]    3  [FinituraC,C]    2
;

end;
```

5.2 Problema n.2: Pianificazione multi-periodo

Il modello matematico era:

$$\begin{aligned}
\min z = & \sum_{i=1}^{M} (cx_i + c^+ x_i^+ + sy_i) \\
\text{s.t.} & y_i = y_{i-1} + x_i + x_i^+ - d_i && \forall i = 1, \ldots, M \\
& x_i \leq Q && \forall i = 1, \ldots, M \\
& x_i^+ \leq Q^+ && \forall i = 1, \ldots, M \\
& x_i \geq 0 && \forall i = 1, \ldots, M \\
& x_i^+ \geq 0 && \forall i = 1, \ldots, M \\
& y_i \geq 0 && \forall i = 1, \ldots, M
\end{aligned}$$

Si basa su un insieme di periodi consecutivi, ordinati da 1 a M. Quando gli elementi di un insieme non sono nomi simbolici, ma valori numerici, è possibile definire l'insieme stesso nel file dei dati senza elencare uno per uno tutti gli elementi: basta indicare gli estremi, separati da due punti consecutivi (`..`). Accade spesso che i valori estremi non siano numeri assegnati a priori, uguali per tutte le istanze del problema, ma siano a loro volta dei dati numerici (in questo caso, il primo estremo è 1, ma il secondo è un parametro M). Tali dati vengono definiti come tutti gli altri nel file del modello e il loro valore effettivo viene indicato nel file dei dati. L'unica avvertenza è che *la definizione del parametro deve precedere quella dell'insieme che ne dipende.*

```
param M := 3;
set Periodi := 1..M;
```

Come già discusso in precedenza, l'equazione di bilancio del prodotto $y_i = y_{i-1} + x_i + x_i^+ - d_i$ per $i = 1$ pone un piccolo problema, dato che richiede l'esistenza di una grandezza y_0, che non è una variabile (del resto, le variabili `y` sono indicizzate sui `Periodi`, e quindi da 1 a `M`), ma un dato costante e noto a priori, cioè l'eventuale giacenza iniziale in magazzino. Questo problema si può risolvere in diversi modi:

1. si può definire un insieme di periodi esteso `Periodi0`, che comprende `Periodi0`, ma anche 0, indicizzare la variabile `y` su di esso e aggiungere un vincolo che le imponga il valore desiderato

2. si può estrarre l'equazione di bilancio per $i = 1$ dalla famiglia cui appartiene, definendola a parte come $y_1 = g + x_1 + x_1^+ - d_1$, dove g è la giacenza iniziale ed è un dato

Prima alternativa Cominciamo a descrivere la prima alternativa. Il set `Periodi0` può essere definito a partire da `Periodi` e dall'insieme che contiene solo lo zero con un'operazione di unione fra insiemi, descritta dalla parola chiave `union`:

```
set Periodi0 := {0} union Periodi;
```

Il linguaggio *GNU MathProg* comprende le principali operazioni fra insiemi: oltre all'unione, l'intersezione (`inter`) e la differenza (`diff`).

Questa definizione di `Periodi0` può essere messa nel file dei dati, ma anche in quello del modello, dato che in essa non si precisano uno per uno gli elementi (che potrebbero cambiare affrontando un'altra istanza del problema), ma la regola con cui si costruisce l'insieme a partire da un altro. Quindi, questa definizione rimane la stessa al variare delle istanze, e si può anche inserire nel file del modello (senza scrivere nulla nel file dei dati). La stessa cosa, in effetti, vale anche per la definizione di `Periodi`, che è sempre `1..M` e non cambia volta per volta. Il valore di `M`, invece, va assegnato nel file dei dati, perché può cambiare ogni volta.

La prima versione del file del modello è quindi

```
# Pianificazione multi-periodo

param M;
set Periodi := 1..M;
set Periodi0 := {0} union Periodi;

param c;     # costo di produzione normale [Euro/pezzo]
param cp;    # costo di produzione extra   [Euro/pezzo]
param s;     # costo di stoccaggio         [Euro/pezzo]
param g;     # giacenza iniziale [pezzi]

param d {Periodi};     # domanda [pezzi/mese]

param Q;               # capacita' produttiva normale [pezzi/mese]
param Qp;              # capacita' produttiva extra   [pezzi/mese]

var x  {Periodi} >= 0; # produzione normale
var xp {Periodi} >= 0; # produzione extra
var y {Periodi0} >= 0; # scorta al termine

maximize z : sum{i in Periodi} (c * x[i] + cp * xp[i] + s * y[i]);

subject to VincoloGiacenzaIniziale :
  y[0] = g;

subject to VincoloBilancio {i in Periodi} :
  y[i] = y[i-1] + x[i] + xp[i] - d[i];

subject to VincoloCapacita {i in Periodi} :
  x[i] <= Q;

subject to VincoloCapacitaExtra {i in Periodi} :
  xp[i] <= Qp;
```

```
end;
```

Seconda alternativa Passiamo ora alla seconda alternativa. Questa non richiede alcun insieme ausiliario, per cui le variabili `y` tornano ad essere semplicemente indicizzate sui `Periodi`. Si richiede però di specificare che l'equazione di bilancio classica non va imposta su tutti i `Periodi`, ma solo su quelli successivi al primo. Abbiamo visto qualcosa del genere in una sommatoria del problema precedente: basta aggiungere alla definizione dell'insieme su cui sono indicizzati i vincoli la condizione logica `: i > 1`, come segue:

```
subject to VincoloBilancio {i in Periodi: i > 1} :
  y[i] = y[i-1] + x[i] + xp[i] - d[i];
```

Ovviamente, bisogna aggiungere un vincolo di bilancio specifico per il periodo iniziale

```
subject to VincoloBilancioIniziale :
  y[1] = g + x[1] + xp[1] - d[1];
```

ma non è più necessario il vincolo di giacenza iniziale.

In conclusione, la seconda versione del file del modello è quindi

```
# Pianificazione multi-periodo

param M;
set Periodi := 1..M;

param c;     # costo di produzione normale [Euro/pezzo]
param cp;    # costo di produzione extra   [Euro/pezzo]
param s;     # costo di stoccaggio         [Euro/pezzo]
param g;     # giacenza iniziale [pezzi]

param d {Periodi};     # domanda [pezzi/mese]

param Q;               # capacita' produttiva normale [pezzi/mese]
param Qp;              # capacita' produttiva extra   [pezzi/mese]

var x  {Periodi} >= 0; # produzione normale
var xp {Periodi} >= 0; # produzione extra
var y  {Periodi} >= 0; # scorta al termine

maximize z : sum{i in Periodi} (c * x[i] + cp * xp[i] + s * y[i]);

subject to VincoloBilancio {i in Periodi: i > 1} :
  y[i] = y[i-1] + x[i] + xp[i] - d[i];
```

```
subject to VincoloBilancioIniziale :
  y[1] = g + x[1] + xp[1] - d[1];

subject to VincoloCapacita {i in Periodi} :
  x[i] <= Q;

subject to VincoloCapacitaExtra {i in Periodi} :
  xp[i] <= Qp;

end;
```

Dati Il file dei dati è rigorosamente identico nei due casi, a conferma del fatto che il problema è lo stesso, ed è solo cambiato il modello con il quale lo abbiamo rappresentato.

```
# Pianificazione multi-periodo

param M := 3;

param c  := 300;     # costo di produzione normale [Euro/pezzo]
param cp := 330;     # costo di produzione extra [Euro/pezzo]
param s  :=  10;     # costo di stoccaggio [Euro/pezzo]
param g  :=   0;     # giacenza iniziale [pezzi]

param d :=           # domanda  [pezzi/mese]
1  100
2  130
3  150
;

param Q  := 110;     # capacita' produttiva normale [pezzi/mese]
param Qp := 60;      # capacita' produttiva extra   [pezzi/mese]

end;
```

5.3 Problema n.3: Posologia

Il modello matematico completo era:

$$\begin{aligned}
\min z' &= \sum_{i=1}^{N}\sum_{j=1}^{F} x_{ij} \\
\min z'' &= \sum_{i=1}^{N}\sum_{j=1}^{F} c_j x_{ij} \\
\text{s.t.} \sum_{j=1}^{F}\sum_{k=0}^{12} a_{kj} x_{i-k\ j} &\geq q_i \qquad \forall i = 1,\ldots,N \\
\sum_{j=1}^{F}\sum_{k=0}^{12} a_{kj} x_{i-k\ j} &\leq Q \qquad \forall i = 1,\ldots,N \\
x_{ij} &\geq 0 \qquad \forall i = 1,\ldots,N\ \forall j = 1,\ldots,F
\end{aligned}$$

che presenta un solo problema nella traduzione in *GNU MathProg*: nelle espressioni $x_{i-k,j}$, l'indice $i-k$ va inteso come indice circolare, cioè calcolato modulo 24. Alcuni linguaggi algebrici consentono di farlo direttamente, definendo l'insieme `FasceOrarie` come `1..N` e specificando che va inteso come insieme circolare, cioè che l'ultimo termine ha come successivo il primo. *GNU MathProg* non lo consente, ma vi sono altri modi di aggirare l'ostacolo: ne descriviamo uno.

Rigorosamente parlando, con $x_{i-k,j}$ intendiamo la variabile $x_{i-k,j}$ quando $i > k$, mentre intendiamo $x_{N+i-k,j}$ quando $i < k$ (dato che comunque $k \leq 12 < N$). Basterebbe quindi disporre di un modo per scrivere due espressioni alternative e una regola per scegliere quella desiderata. Questo modo esiste, ed è il costrutto `(if` *condizione* `then` *espressione 1* `else` *espressione 2*`)`, dove *espressione 1* e *espressione 2* sono le due espressioni alternative e *condizione* è la condizione logica che definisce quando usare la prima. Nel nostro caso, all'interno dei vincoli potremmo sostituire `x[i-k,j]`, che sarebbe errato perché `i-k` non ricade in `FasceOrarie`, con l'espressione (if (i ¿ k) then x[i-k,j] else x[N+i-k,j]). Il primo vincolo, quindi, diventerebbe

```
sum{j in Farmaci, k in Distanze} a[k,j] *
  (if (i > k) then x[i-k,j] else x[N+i-k,j])*1000 >= q[i];
```

Si noti anche come è possibile raccogliere più sommatorie in una sola e il coefficiente 1000, che trasforma i grammi di farmaco assunti in milligrammi.

Quindi il file del modello (in cui consideriamo solo la seconda funzione obiettivo) è

```
# Posologia

param N;  # numero delle fasce orarie
param F;  # numero dei farmaci
```

```
set FasceOrarie := 1..N;
set Farmaci := 1..F;
set Distanze := 0..12;

param c {Farmaci};           # costo del farmaco [Euro/g]
param q {FasceOrarie};       # fabbisogno nelle diverse fasce orarie [mg/cc]
param Q;                     # quantita' massima di proteina ammissibile [mg/cc]
param a {Distanze,Farmaci};  # quantita' generata a una certa distanza
                             # di tempo da un certo farmaco [mg/cc]

var x  {FasceOrarie,Farmaci} >= 0; # quantita' di farmaco assunta [g]

minimize z : sum{i in FasceOrarie, j in Farmaci} c[j] * x[i,j];

subject to VincoloFabbisogno {i in FasceOrarie} :
  sum{j in Farmaci, k in Distanze} a[k,j] *
    (if (i > k) then x[i-k,j] else x[N+i-k,j])/1000 >= q[i];

subject to VincoloMassimo {i in FasceOrarie} :
  sum{j in Farmaci, k in Distanze} a[k,j] *
    (if (i > k) then x[i-k,j] else x[N+i-k,j])/1000 <= Q;

end;
```

E il file dei dati è

```
# Posologia

param N := 24;  # numero delle fasce orarie
param F :=  2;  # numero dei farmaci

param c :=      # costo del farmaco [Euro/g]
1  0.70
2  0.95
;

param q :=      # fabbisogno nelle diverse fasce orarie [mg/cc]
 1   5.0
 2   1.0
 3   0.0
 4   0.0
 5   0.0
 6   0.0
 7   4.0
 8  15.0
 9  12.0
10   5.0
```

```
11   4.0
12   3.0
13  25.0
14  30.0
15  25.0
16  15.0
17   5.0
18   4.0
19   3.0
20  25.0
21  30.0
22  25.0
23  20.0
24  10.0
;

param Q := 45;  # quantita' massima di proteina ammissibile [mg/cc]

param a :=      # quantita' generata a una certa distanza
                # di tempo da un certo farmaco [mg/cc]
[ 0,1]   1.5  [ 0,2]  2.5
[ 1,1]   3.0  [ 1,2]  4.0
[ 2,1]   4.0  [ 2,2]  5.5
[ 3,1]   2.5  [ 3,2]  4.0
[ 4,1]   1.9  [ 4,2]  3.0
[ 5,1]   1.4  [ 5,2]  1.5
[ 6,1]   1.0  [ 6,2]  0.7
[ 7,1]   0.7  [ 7,2]  0.4
[ 8,1]   0.5  [ 8,2]  0.2
[ 9,1]   0.3  [ 9,2]  0.0
[10,1]   0.2  [10,2]  0.0
[11,1]   0.1  [11,2]  0.0
[12,1]   0.0  [12,2]  0.0
;

end;
```

5.4 Problema n.4: Trasporto a costo minimo

Il modello matematico completo era:

$$
\begin{aligned}
\text{minimize} \quad z = & \sum_{i=1}^{O} \sum_{j=1}^{D} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{D} x_{ij} = o_i \qquad \forall i = 1, \ldots, O \\
& \sum_{i=1}^{O} x_{ij} = d_j \qquad \forall j = 1, \ldots, D \\
& x_{ij} \geq 0 \qquad \forall i = 1, \ldots, O \;\; \forall j = 1, \ldots, D.
\end{aligned}
$$

da cui non è difficile ricavare il file del modello

```
# Trasporto

param O;  # numero delle origini
param D;  # numero delle destinazioni

set Origini := 1..O;        # insieme delle origini
set Destinazioni := 1..D;   # insieme delle origini

# quantita' offerte
param o {Origini} >= 0;

# quantita' richieste
param d {Destinazioni} >= 0;

# costo unitario di trasporto
param c {Origini,Destinazioni} >= 0;

# quantita' trasportate
var x {Origini,Destinazioni} >= 0;

minimize z : sum{i in Origini, j in Destinazioni} c[i,j] * x[i,j];

subject to VincoloOrigini {i in Origini} :
  sum{j in Destinazioni} x[i,j] = o[i];

subject to VincoloDestinazioni {j in Destinazioni} :
  sum{i in Origini} x[i,j] = d[j];

end;
```

e il file dei dati

```
# Trasporto
```

```
param O := 8;  # numero delle origini
param D := 4;  # numero delle destinazioni

# quantita' offerte
param o :=
1  30
2  40
3  20
4  35
5  40
6  30
7  25
8  50
;

# quantita' richieste
param d :=
1  70
2  70
3  50
4  80
;

# costo unitario di trasporto
param c :=
[1,1] 20  [1,2] 25  [1,3] 30  [1,4] 28
[2,1] 15  [2,2] 12  [2,3] 32  [2,4] 26
[3,1] 18  [3,2] 41  [3,3] 36  [3,4] 37
[4,1] 32  [4,2] 23  [4,3] 35  [4,4] 20
[5,1] 31  [5,2] 40  [5,3] 19  [5,4] 38
[6,1] 33  [6,2] 22  [6,3] 34  [6,4] 21
[7,1] 25  [7,2] 29  [7,3] 26  [7,4] 27
[8,1] 30  [8,2] 24  [8,3] 39  [8,4] 28
;

end;
```

Come si potrebbe procedere qualora alcune delle tratte origine-destinazione fossero proibite?

5.5 Problema n.5: Lo zaino

Il modello matematico completo era:

$$\begin{aligned} \text{maximize} \quad & z = \sum_{i=1}^{N} c_i x_i \\ \text{subject to} & \sum_{i=1}^{N} a_i x_i \leq b \\ & x_i \in \{0,1\} \qquad\qquad \forall i = 1, \ldots, N \end{aligned}$$

e non presenta difficoltà aggiuntive rispetto a quelli già affrontati, salvo il fatto che le variabili `x` non sono continue (e non negative), ma sono binarie. Questo comporta una semplice modifica nella loro definizione nel file del modello, che richiede l'uso della parola chiave `binary`, come segue:

```
var x {Oggetti} binary;
```

Se ci trovassimo di fronte a generiche variabili intere, si userebbe invece la parola chiave `integer`.

Il file del modello diventa quindi:

```
# Zaino

param N;              # numero degli oggetti
set Oggetti := 1..N; # insieme degli oggetti

param c {Oggetti} >= 0; # valore di ogni oggetto
param a {Oggetti} >= 0; # volume di ogni oggetto
param b >= 0;           # capacita' dello zaino

# scelta degli oggetti (1 se inserire, 0 se no)
var x {Oggetti} binary;

maximize z : sum{i in Oggetti} c[i] * x[i];

subject to VincoloCapacita :
  sum{i in Oggetti} a[i] * x[i] <= b;

end;
```

e quello dei dati:

```
# Zaino

param N := 10;          # numero degli oggetti

param c :=              # valore di ogni oggetto
```

```
 1   4
 2   6
 3  40
 4  15
 5  20
 6  20
 7  21
 8  38
 9  46
10  56
;

param a :=              # volume di ogni oggetto
 1   8
 2   9
 3  13
 4  24
 5  28
 6  36
 7  41
 8  57
 9  68
10  70
;

param b := 100;         # capacita' dello zaino

end;
```

5.6 Problema n.6: Set covering

Il modello matematico completo era:

$$\begin{aligned} \text{minimize} \quad & z = \sum_{j=1}^{M} c_j x_j \\ \text{subject to} & \sum_{j=1}^{M} a_{ij} x_j \geq 1 && \forall i = 1, \ldots, N \\ & x_j \in \{0, 1\} && \forall j = 1, \ldots, M. \end{aligned}$$

Questo modello può essere realizzato almeno in due forme alternative, la prima più semplice, la seconda più compatta e matematicamente più elegante. Entrambe consentono di introdurre nuovi elementi di *GNU MathProg*.

Prima alternativa Si tratta ovviamente di definire l'insieme degli utenti e quello dei luoghi, il vettore dei costi e soprattutto la matrice binaria che indica,

per ogni utente i e luogo j se il luogo copre l'utente ($a_{ij} = 1$) oppure no ($a_{ij} = 0$). La definizione standard nel file dei dati comporterebbe un lungo elenco di valori tutti uguali a 1 o a 0:

```
param a :=
[1,1] 1  [1,2] 0 [1,3] 0 [1,4] 0  [1,5] 1...
```

che è evidentemente ridondante: basterebbe indicare le caselle di valore 1 e suggerire che tutte le altre hanno valore nullo. Si può ottenere questo risultato usando la parola chiave `default` nel file del modello e riportando nel file dei dati solo i valori che non coincidono con il valore di *default*. Più precisamente, nel file del modello si scrive:

```
param a {Utenti,Luoghi} >= 0, <= 1, default 0;
```

(dove si possono notare anche le condizioni che impongono ai dati numerici di non essere negativi né maggiori di 1) e nel file dei dati

```
param a :=
[1,1] 1  [1,5] 1...
```

Le caselle per le quali non viene specificato un valore assumono quello di di *default*.

Concludendo, il file del modello risulta

```
# Set Covering

param M;             # numero degli utenti
param N;             # numero dei luoghi

set Utenti := 1..M;  # insieme degli utenti
set Luoghi := 1..N;  # insieme dei luoghi

param c {Luoghi} >= 0;                          # costo di ogni luogo
param a {Utenti,Luoghi} >= 0, <= 1, default 0;  # matrice di copertura

# scelta dei luoghi (1 se usare, 0 se no)
var x {Luoghi} binary;

minimize z : sum{j in Luoghi} c[j] * x[j];

subject to VincoloCopertura {i in Utenti} :
  sum{j in Luoghi} a[i,j] * x[j] >= 1;

end;
```

e il file dei dati

```
# Set Covering

param M :=  5;       # numero degli utenti
param N := 10;       # numero dei luoghi

param c :=           # costo di ogni luogo
 1  205
 2  311
 3  450
 4  274
 5  321
 6  469
 7  327
 8  631
 9  750
10  400
;

param a :=            # matrice di copertura
[1,1] 1  [1,5] 1  [1,6] 1  [1,7] 1  [1,9] 1  [1,10] 1
[2,4] 1  [2,5] 1  [2,9] 1
[3,5] 1  [3,6] 1
[4,7] 1  [4,8] 1  [4,9] 1
[5,1] 1  [5,2] 1  [5,3] 1  [5,7] 1  [5,8] 1  [5,9] 1
;

end;
```

Seconda alternativa Una seconda alternativa consiste nel modellare il problema associando ad ogni utente i il sottoinsieme L_i dei luoghi che lo coprono. A questo punto, non c'è più bisogno della matrice a_{ij}, perché i vincoli di copertura si possono scrivere nella forma

$$\sum_{j \in L_i} x_j \geq 1 \qquad \forall i = 1, \ldots, N$$

In realtà, questa scrittura è assolutamente equivalente all'altra, ma si traduce in un programma diverso in linguaggio *GNU MathProg*. Bisogna infatti definire una *collezione di insiemi* (ovvero insieme di insiemi), indicizzato sull'insieme degli `Utenti`, cioè associare ad ogni utente non un numero (come si è fatto sinora), ma un insieme. La sintassi è semplice. Nel file del modello, occorre definire la collezione L come:

```
set L {Utenti} within Luoghi;
```

Questa scrittura assomiglia sia a quella degli insiemi sia a quella dei vettori numerici. Differisce da quella degli insiemi semplici perché `L` è seguita dall'insieme

sul quale è indicizzata, cioè l'insieme degli `Utenti`: non è un insieme solo, ma sono tanti, uno per ogni utente. Differisce da quella dei vettori perché è preceduta dalla parola chiave `set` anziché `param`. Si noti anche il fatto che ciascun insieme della collezione è definito come sottoinsieme dell'insieme dei `Luoghi`.

Nel file dei dati, occorre definire per ogni elemento dell'insieme degli `Utenti` il corrispondente sottoinsieme dei `Luoghi` che lo coprono.

```
set L[1] := 1 5 6 7 9 10;
set L[2] := 4 5 9;
...
```

Concludendo, il file del modello diviene:

```
# Set Covering

param M;             # numero degli utenti
param N;             # numero dei luoghi

set Utenti := 1..M;  # insieme degli utenti
set Luoghi := 1..N;  # insieme dei luoghi

param c {Luoghi} >= 0;                          # costo di ogni luogo
param a {Utenti,Luoghi} >= 0, <= 1, default 0;  # matrice di copertura

# scelta dei luoghi (1 se usare, 0 se no)
var x {Luoghi} binary;

minimize z : sum{j in Luoghi} c[j] * x[j];

subject to VincoloCopertura {i in Utenti} :
  sum{j in Luoghi} a[i,j] * x[j] >= 1;

end;
```

e il file dei dati diviene:

```
# Set Covering

param M :=  5;       # numero degli utenti
param N := 10;       # numero dei luoghi

# collezione dei luoghi che coprono ogni utente
set L[1] := 1 5 6 7 9 10;
set L[2] := 4 5 9;
set L[3] := 5 6;
set L[4] := 7 8 9;
set L[5] := 1 2 3 7 8 9;
```

```
param c :=           # costo di ogni luogo
 1  205
 2  311
 3  450
 4  274
 5  321
 6  469
 7  327
 8  631
 9  750
10  400
;

end;
```

5.7 Problema n.7: Containers

Per brevità, consideriamo solo l'ultimo dei tre modelli precedentemente formulati:

$$
\begin{aligned}
\text{minimize} \quad & z = \sum_{j=1}^{M} c_j y_j \\
\text{subject to} \quad & \sum_{j=1}^{M} x_{ij} = 1 && \forall i = 1, \ldots, N \\
& \sum_{i=1}^{N} p_i x_{ij} \leq P_j y_j && \forall j = 1, \ldots, M \\
& y_j \in \{0, 1\} && \forall j = 1, \ldots, M \\
& x_{ij} \in \{0, 1\} && \forall i = 1, \ldots, N \quad \forall j = 1, \ldots, M.
\end{aligned}
$$

La traduzione non presenta particolari difficoltà. Il file del modello è:

```
# Containers

param N;                # numero degli oggetti
param M;                # numero dei container

set Oggetti   := 1..N;  # insieme degli oggetti
set Container := 1..M;  # insieme dei container

param p {Oggetti} >= 0;    # peso di ogni oggetto
param P >= 0;              # capacita' dei container
param c {Container} >= 0;  # costo di ogni container

# assegnamento degli oggetti ai container
```

```
var x {Oggetti,Container} binary;

# scelta dei container
var y {Container} binary;

minimize z : sum{j in Container} c[j] * y[j];

subject to VincoloAssegnamento {i in Oggetti} :
  sum{j in Container} x[i,j] = 1;

subject to VincoloCapacita {j in Container} :
  sum{i in Oggetti} p[i] * x[i,j] <= P * y[j];

end;
```

E il file dei dati è:

```
# Containers

param N := 15;  # numero degli oggetti
param M :=  4;  # numero dei container

param p :=  # peso di ogni oggetto
 1 10
 2 24
 3 18
 4  7
 5  7
 6  6
 7 16
 8 11
 9  8
10 11
11  1
12  6
13 15
14  8
15  2
;

param P :=  # capacita' dei container
 1 45
 2 50
 3 60
 4 80
;
```

```
param c :=  # costo di ogni container
 1  900
 2 1000
 3 1200
 4 1300
;

end;
```

5.8 Problema n.8: Sentinelle

Il modello matematico completo era:

$$\begin{aligned} \text{minimize} \quad & z = \sum_{i=1}^{N} x_i \\ \text{subject to} \quad & \sum_{j=1}^{N} a_{ij} x_j \geq 1 \qquad \forall i = 1, \ldots, N \\ & x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, N \end{aligned}$$

La traduzione diretta di questo modello in linguaggio *GNU MathProg* è la seguente:

```
# Sentinelle

param N;              # numero degli incroci
set Incroci:= 1..N;  # insieme degli incroci

# matrice della connessione fra incroci
param a {Incroci,Incroci} >= 0, <= 1, default 0;

# assegnamento degli oggetti ai container
var x {Incroci} binary;

minimize z : sum{i in Incroci} x[i];

subject to VincoloSorveglianza {i in Incroci} :
  sum{j in Incroci} a[i,j] * x[j] >= 1;

end;
```

con il seguente file dei dati:

```
# Sentinelle

param N := 30;  # numero degli incroci
```

```
# matrice della connessione fra incroci
param a :=
 [1, 2] 1   [1, 3] 1   [1, 4] 1
 [2, 1] 1   [2,30] 1
 [3, 1] 1   [3,13] 1   [3,16] 1
 [4, 1] 1   [4, 5] 1   [4, 6] 1   [4,24] 1
 [5, 4] 1   [5, 6] 1   [5, 8] 1   [5,13] 1
 [6, 4] 1   [6, 5] 1   [6, 7] 1
 [7, 6] 1   [7, 9] 1   [7,10] 1
 [8, 5] 1   [8, 9] 1   [8,12] 1   [8,13] 1   [8,27] 1
 [9, 7] 1   [9, 8] 1   [9,10] 1   [9,27] 1
[10, 7] 1  [10, 9] 1  [10,11] 1
[11,10] 1  [11,23] 1  [11,29] 1
[12, 8] 1  [12,13] 1  [12,18] 1  [12,19] 1  [12,27] 1
[13, 3] 1  [13, 5] 1  [13, 8] 1  [13,12] 1  [13,14] 1
[14,13] 1  [14,15] 1  [14,18] 1
[15,14] 1  [15,16] 1  [15,17] 1
[16, 3] 1  [16,15] 1
[17,15] 1
[18,12] 1  [18,14] 1  [18,19] 1  [18,28] 1
[19,12] 1  [19,18] 1  [19,20] 1  [19,28] 1
[20,19] 1  [20,21] 1  [20,22] 1  [20,29] 1
[21,20] 1
[22,20] 1
[23,11] 1
[24, 4] 1  [24,25] 1  [24,26] 1
[25,24] 1
[26,24] 1
[27, 8] 1  [27, 9] 1  [27,12] 1  [27,29] 1
[28,18] 1  [28,19] 1
[29,11] 1  [29,20] 1  [29,27] 1
[30, 2] 1
;

end;
```

Si noti in particolare che le strade di collegamento consentono una relazione simmetrica di sorveglianza fra incroci, cioè se una strada va dall'incrocio i_1 all'incrocio i_2. sorvegliando ciascuno dei due si sorveglia automaticamente l'altro.

È anche possibile proporre una versione alternativa, nella quale anziché una matrice numerica si usa l'insieme delle vie per esprimere il vincolo di connessione. In che modo?