UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Tutorato di programmazione

Piano Lauree Scientifiche - Progetto di Informatica

Goal, plan e ruoli delle variabili - appunti

Ci sono problemi o compiti di base (*goal*) molto comuni per cui esistono soluzioni algoritmiche paradigmatiche (*plan*). In questi appunti introduciamo innanzitutto un po' di terminologia utile, poi presentiamo una serie di compiti di base molto comuni per cui esistono soluzioni paradigmatiche basate sull'uso di cicli.

Autori:
Violetta Lonati (coordinatrice del gruppo), Anna Morpurgo e Umberto Costantini

Hanno contribuito alla revisione del materiale gli studenti tutor:
Alessandro Clerici Lorenzini, Alexandru David, Andrea Zubenko, Davide Cernuto, Francesco Bertolotti, Leonardo Albani, Luca Tansini, Margherita Pindaro, Umberto Costantini, Vasile Ciobanu, Vittorio Peccenati.

Si ringraziano per i numerosi spunti:
Carlo Bellettini, Paolo Boldi, Mattia Monga, Massimo Santini e Sebastiano Vigna.

1 I concetti di *goal* e *plan* (obiettivo e piano)

Ci sono **problemi o compiti di base** molto comuni per cui esistono **soluzioni algoritmiche paradigmatiche**. È molto utile imparare a riconoscere i sotto-compiti di base in un problema e conoscere le loro soluzioni paradigmatiche. **Non** si tratta di imparare un insieme di **"tecniche" da applicare a occhi chiusi**, ma di cogliere la logica che ci sta dietro e avere dei punti di partenza per poter progettare programmi per problemi via via più complessi.

Introduciamo un po' di terminologia che ci sarà utile. Chiameremo indifferentemente:

- ***goal*** o ***obiettivo*** o ***funzione*** o ***compito*** ciascun sottocompito che deve essere affrontato nella soluzione di un problema
- ***piano*** o ***soluzione tipica*** la soluzione paradigmatica con cui viene realizzato uno specifico compito di base

Ad esempio, per risolvere un problema potremmo avere a che fare con i seguenti obiettivi:

- calcolare la somma di una serie di numeri;
- verificare se un numero di tipo *float* può essere considerato "uguale" (abbastanza vicino) a un valore dato.

Esempi di soluzioni tipiche per questi due obiettivi sono:

- ```
  somma :=0
  for _,n := range serie {
    somma += n
  }
  ```
- `math.Abs(x - val) < EPSILON`

Per illustrare i concetti di *goal* e *plan* consideriamo come esempio il problema di calcolare la media di una sequenza di numeri terminata da 9999.

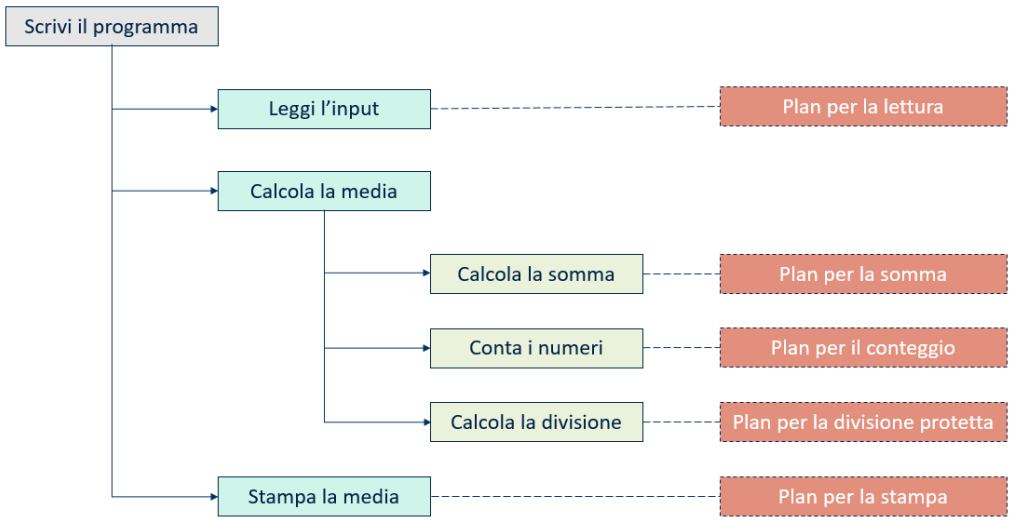

Figura 1: Calcolo della media: obiettivi, sotto-obiettivi e piani

Come mostrato in Figura 1, possiamo identificare 3 obiettivi principali:
  ```

- lettura dei numeri in input
- calcolo della media, che a sua volta può essere scomposto in tre sotto-obiettivi:
  - calcolo della somma dei numeri letti (escluso il valore di terminazione 9999)
  - conteggio dei numeri letti (escluso il valore di terminazione)
  - divisione tra la somma e il conteggio dei numeri letti, controllando che il divisore non sia zero
- stampa dell'output

A ciascun sotto-obiettivo corrisponderà un piano, e i piani per i sotto-obiettivi andranno poi combinati in un unico programma che risolva il problema dato.

Tratteremo più avanti il tema della composizione dei diversi piani in un programma.

2 Piani che fanno uso di cicli

Presentiamo ora i problemi di base la cui soluzione prevede un piano basato sull'iterazione e mostriamo i relativi piani.

Nota: nei prossimi esempi sono evidenziate in grassetto le variabili che fanno parte del piano in quanto hanno un ruolo cruciale. Riprenderemo nella sezione successiva questo aspetto.

2.1 Ripetizione

Alcuni compiti si risolvono ripetendo una data azione, o sequenza di azioni, un certo numero di volte.

Ad esempio, il compito di stampare *n* asterischi si risolve stampando 1 asterisco *n* volte:

```
 for i:=1; i<=n; i++ {
  fmt.Print("*")
}
```

In generale, nel *piano per la ripetizione*, un blocco di istruzioni che definisce l'azione da ripetere un certo numero di volte per risolvere il compito, viene eseguito a ogni iterazione. A volte nelle istruzioni è presente anche l'indice usato per gestire il ciclo, in modo da avere azioni parametriche. Ad esempio:

```
for i:=1; i<=n; i++ {
  fmt.Println(i*i)
}
```

2.2 Calcolo di un totale

Sommare o moltiplicare una serie di numeri da input o di valori già salvati in una stuttura di dati (array, slice, ...) è un compito molto comune.

Ad esempio questo è il piano per sommare *n* valori letti da standard input:

```
total :=0
for i:=1; i<=n; i++ {
  fmt.Scan(&newValue)
  total += newValue
}
```

Un compito che ha un piano analogo è quello di concatenare in un'unica stringa una serie di stringhe o caratteri. Si vedano anche i frammenti di codice 2, 4 e 6.
In particolare si noti che nel *piano per il calcolo di un totale* si usa per il risultato una variabile (nel Listing 2 `prodotto` e nei Listing 4 e 6 `somma`) che viene opportunamento inizializzata prima del ciclo e a cui viene sommato ciascun valore, uno a ogni iterazione.

2.3 Conteggio

Contare quanti valori soddisfano una data condizione o il numero iterazioni necessarie a raggiungere un obiettivo/risultato è un altro compito in cui ci si imbatte molto di frequente.

Questo è, ad esempio, il piano per contare le maiuscole in un riga di testo.

```
count := 0
for _, r := range line {
  if unicode.IsUpper(r) {
    count++
  }
}
```

Il *piano per il conteggio* è molto simile a quello per il calcolo di un totale. Anche qui è sempre presente una variabile che viene opportunamento inizializzata prima del ciclo e poi aggiornata a ogni iterazione, con la differenza che in questo caso si tratta di una variabile con il ruolo di *contatore* che, ad ogni iterazione, viene incrementata se la condizione è soddisfatta.

2.4 Identificazione della prima (eventuale) occorrenza/corrispondenza

Ci sono problemi su serie di valori per cui serve un ciclo ma non occorre esaminare tutti gli elementi della serie. In particolare, quando l'obiettivo è trovare in una serie un valore che soddisfa una condizione data, ci si può fermare non appena se ne trova uno.

Nel risolvere problemi di questo tipo occorre generalmente gestire anche l'eventualità che il valore cercato non sia presente nella serie analizzata. Con una variabile booleana si può gestire questo aspetto: la si inizializza a `false` e la si porta a `true` se e quando il valore cercato viene incontrato.

In alternativa, e soprattutto nel caso si sia interessati a sapere anche la posizione in cui si trova il valore cercato, si usa una variabile per la posizione inizializzandola a un valore non valido per una posizione (spesso il valore -1) e aggiornandola se e quando il valore cercato viene incontrato.

Un piano per trovare il primo numero pari in un array può essere questo:

```
found := false //o, in alternativa, pos := - 1
for i:=1; i<=n; i++{
  if array[i] %2 == 0 {
    found = true //o, in alternativa, pos = i
    break
  }
}
```

Parliamo in questo caso di *piano per la identificazione della prima occorrenza* o corrispondenza (o *piano per la ricerca lineare*).

Un ciclo for unario permette di gestire la situazione senza ricorrere a un `break`:

```
found := false
pos := 0
for !found && pos < n
  if array[pos] %2 == 0 {
    found = true
  } else {
    pos++
  }
}
```

2.5 Ricerca del massimo o del minimo

Altro problema molto comune è quello di determinare il massimo o il minimo di una serie di valori.

La soluzione di questo problema consiste nel confrontare il candidato come valore massimo/minimo con ogni valore della serie e aggiornarlo ogni volta che si incontra un valore maggiore/minore. Una variabile apposta conterrà il candidato. Il valore di questa variabile una volta raggiunta le fine della serie è il valore (massimo/minimo) cercato. Occorre però avere un primo candidato con cui confrontare i valori della serie. Spesso la soluzione più semplice è usare il primo elemento della serie come primo candidato.

Ecco ad esempio il piano per trovare il valore minimo in un array:

```
min := array[0]
for i:=1; i< n; i++ {
  if array[i] < min {
    min = array[i]
  }
}
```

Altri esempi sono i frammenti 1, 3 e 7.

Nel *piano per ricerca del valore estremo* è sempre presente una variabile (`maxM` nel Listing 1, `min` nel Listing 7, `ultimoGiornoPioggia` nel Listing 3) che viene inizializzata prima del ciclo al primo elemento della sequenza o a un valore estremo per il dominio di valori considerato (es. 0 o `math.MinInt64` o `math.MaxInt64`) e poi aggiornata nel ciclo ogni qual volta viene trovato un valore più estremo (maggiore o minore) di quello memorizzato precedentemente.

2.6 Elaborazione su valori adiacenti

A volte il problema da risolvere prevede che gli elementi della serie non vengano considerati uno alla volta, ma a coppie di valori (o a terne, ...). Un esempio di problema di questo tipo è stampare le differenze tra numeri adiacenti, cioè tra il secondo e il primo, tra il terzo e il secondo e così via.

Per risolvere questo problema, partiamo dal piano per leggere una serie di valori:

```
var current int
for i:= 0; i <n; i++ {
  fmt.Scan(&current)
}
```

Ad ogni iterazione, il valore appena letto viene salvato in `current`, sovrascrivendo il precedente. Per il nostro problema occorre quindi salvare il valore precedente prima che venga sovrascritto:

```
  previous = current
  fmt.Scan(&current)
```

Inoltre alla prima iterazione non si dispone ancora di un valore precedentemente memorizzato, cosa che si può risolvere facendo la lettura iniziale prima del ciclo.

Ecco il piano completo:

```
var previous, current int
fmt.Scan(&current)
for i:= 1; i <n; i++ {
  previous = current
  fmt.Scan(&current)
  fmt.Println(current - previous)
}
```

Ricadono in questo tipo i problemi in cui occorre analizzare l'andamento di valori numerici in una sequenza / elaborare valori contigui. Vedi anche i frammenti 5, 8 e 9.

Si noti che nel *piano per l'uso di valori adiacenti* due (o più, a seconda di quanti valori adiacenti occorre considerare) variabili (`numeri[i]` e `numeri[i-1]` nei Listing 5 e 8, `corrente` e `precedente` nel Listing 9) scorrono insieme "a braccetto" la sequenza, assumendo a ogni iterazione la seconda il valore della prima e la prima il nuovo valore da considerare.

3 Ruoli delle variabili

Per quanto riguarda le variabili, si può fare un discorso analogo. È infatti importante imparare, oltre al concetto di variabile in sé come contenitore di un valore modificabile di un dato tipo, i casi d'uso specifici delle variabili nel risolvere problemi. Si parla in questo caso di *ruolo delle variabili*. Si pensi per esempio al problema di identificare il valore massimo in una sequenza di valori numerici o di calcolarne la somma e alle variabili che hanno un ruolo essenziale nella soluzione del problema.

Nel presentare i piani per l'iterazione abbiamo già evidenziato in grassetto le variabili che hanno un ruolo cruciale e che quindi fanno parte del piano stesso. Le riprendiamo qui, associandole ai relativi piani:

Ruolo	**Plan**	**Esempi**	**Definizione informale**
Accumulatore (*Gatherer*)	Calcolo di un totale	`sum`, `total`, `new_string`	Accumula i contributi di valori individuali
Passo-passo (*Stepper*)	Conteggio / posizione nella Ricerca lineare	`count`, `index`	Assume una successione di valori che variano in modo sistematico e predicibile
One-way flag	Ricerca lineare	`found`, `errorsOccurred`	Una variabile a due valori (spesso booleana) che può cambiare valore una volta sola
Ricercato (*Most-wanted holder)*	Ricerca del minimo o del massimo	`max`, `min`	Contiene il miglior risultato ottenuto finora
Inseguitore (*Follower*) e inseguito (detti anche osso e segugio)	Elaborazione su valori adiacenti	`previousValue` e `currentValue`	Variabile (l'inseguitore) che viene sempre aggiornata con il vecchio valore di un'altra variabile (l'inseguito)

Come già fatto notare nelle descrizioni degli obiettivi e relativi piani che usano cicli:

- Il ***piano per il calcolo di un totale*** dovrà avere una ***variabile accumulatore***, inizializzata fuori dal ciclo al valore di partenza (es. 0 per una somma, 1 per un prodotto, "" per una stringa, ma anche valori diversi da questi determinati dal problema specifico) e che viene aggiornata all'interno del ciclo.

- Il ***piano per il conteggio*** dovrà avere una ***variabile passo-passo***, inizializzata fuori dal ciclo a un valore opportuno (spesso 0 o 1, ma anche qui il problema specifico può richiedere un valore diverso) e che viene incrementata all'interno del ciclo.
- Il ***piano per l'identificazione della prima occorrenza***, detta anche ricerca lineare, dovrà avere una ***variabile one-way flag***, inizializzata fuori dal ciclo a un valore opportuno (tendenzialmente a `false`, ma anche qui il problema specifico può richiedere `true` invece) e che viene modificata una sola volta all'interno del ciclo, spesso in un `if` con un `break`. Nel caso si voglia sapere (anche) la posizione, spesso si fa a meno della *variabile one-way flag*, utilizzando una ***variabile passo-passo*** che rileva la posizione e inizializzandola a una posizione non valida (tipicamente -1, anche qui il problema specifico può richiedere un valore diverso), affidando così alla variabile per la posizione anche il compito di segnalare la non presenza del valore cercato.
- Il ***piano per la ricerca del massimo o del minimo*** dovrà avere una ***variabile ricercato***, inizializzata fuori dal ciclo a un valore opportuno (a volte al primo valore della sequenza, altre a un valore limite per il dominio considerato, come 0 per valori non negativi, ma anche qui il problema specifico può richiedere un valore diverso) e che viene aggiornata all'interno del ciclo.
- Il ***piano per l'elaborazione su valori adiacenti*** dovrà avere una ***variabile inseguito***, inizializzata fuori dal ciclo e che viene aggiornata all'interno del ciclo, e una ***variabile inseguitore*** il cui aggiornamento avviene all'interno del ciclo appena prima dell'aggiornamento di *inseguito*.

Test di verifica della lettura di "Goal, plan e ruoli delle variabili"

Dopo aver letto con attenzione gli appunti, rispondete alle domande che seguono.

Per ciascuna domanda, scegliere la risposta corretta.

1. Che cosa è un *goal*?
   a) una soluzione tipica per un compito dato
   b) un sotto-compito che deve essere affrontato nella soluzione di un problema
   c) un programma realizzato per uno scopo dichiarato
   d) l'obiettivo a cui deve tendere un programma
2. Che cosa è un *plan*?
   a) uno dei sottoproblemi in cui va scomposto il problema da risolvere
   b) una funzione di libreria
   c) la soluzione tipica con cui viene realizzato uno specifico goal
   d) il progetto di cui fa parte il programma da scrivere
3. A cosa ci si riferisce quando si parla di goal/plan per la ripetizione?
   a) al fare una data azione un certo numero di volte
   b) alla ripetizione dello stesso blocco di codice in punti diversi del programma
   c) al ripetere in output l'input letto
   d) a ripetere negli esercizi proposti quanto si è imparato a lezione
4. A cosa ci si riferisce quando si parla di goal/plan per il calcolo di un totale?
   a) all'inizializzazione di una variabile di nome `totale`
   b) al programma `totale` che bisogna implementare
   c) alla memorizzazione di tutto l'input in un'unica variabile
   d) a una computazione, come la somma o il prodotto, che accumula diversi valori in un unico risultato
5. A cosa ci si riferisce quando si parla di goal/plan per il conteggio?
   a) a una computazione del numero di valori in un insieme
   b) all'utilizzo di un ciclo ternario la cui terza espressione è `i++`
   c) alla dichiarazione di una variabile per un conteggio
   d) all'uso di una dell'espressione `+=1`
6. A cosa ci si riferisce quando si parla di goal/plan per l'identificazione della prima occorrenza?
   a) al test per verificare che ci sia almeno un primo elemento in una sequenza
   b) all'identificazione della prima variabile che serve all'interno del programma
   c) alla ricerca del valore che compare più spesso in una sequenza
   d) alla ricerca lineare di un dato valore in una sequenza
7. A cosa ci si riferisce quando si parla di goal/plan per la ricerca di un massimo o di un minimo?
   a) all'identificazione dell'ultimo valore letto
   b) all'identificazione in una sequenza dell'elemento minimo o massimo rispetto a un criterio dato (valore, posizione, ecc.)
   c) alla ricerca del primo/ultimo elemento di un struttura (array, slice, ...)
   d) alle costanti che rappresentano i valori minimi e massimi per un tipo (`int`, `float64`, `...`) messe a disposizione nelle librerie del linguaggio di programmazione
8. A cosa ci si riferisce quando si parla di goal/plan per l'elaborazione su valori adiacenti?
   a) all'incremento di un indice in un ciclo in modo che assuma il valore successivo
   b) alla lettura dell'input due elementi alla volta
   c) a una computazione che si basa sull'elaborazione di un certo numero di elementi consecutivi in una sequenza
   d) al sommare ogni valore letto a quello successivo