

Fondamenti di Informatica (Elettronici)

THINK JULIA – Capitolo 11

20 novembre 2020

11. Dizionari¹

- 1 Un dizionario è un mapping (una funzione)
- 2 Dizionario come collezione di contatori
- 3 Ciclare sui dizionari
- 4 Ricerca inversa
- 5 Dizionari e array
- 6 Promemoria
- 7 Variabili globali
- 8 Debug
- 9 Glossario
- 10 Esercizi

¹Tratto da <https://benlauwens.github.io/ThinkJulia.jl/latest/book.html>, disponibile sotto Licenza 'Creative Commons Attribution-NonCommercial 3.0 Unported'.

Questo capitolo presenta un altro tipo predefinito chiamato dizionario.

Section 1

Un dizionario è un mapping (una funzione)

Mappatura dalle chiavi ai valori

Un **dizionario** è come un **array**, ma **più generale**.

- In un **array**, gli indici devono essere **numeri interi**; in un **dizionario** possono essere (quasi) **qualsiasi tipo**.

Mappatura dalle chiavi ai valori

Un **dizionario** è come un **array**, ma **più generale**.

- In un **array**, gli indici devono essere **numeri interi**; in un **dizionario** possono essere (quasi) **qualsiasi tipo**.
- Un **dizionario** contiene una **raccolta di indici**, denominati **chiavi**, e una raccolta di **valori**.

Mappatura dalle chiavi ai valori

Un **dizionario** è come un **array**, ma **più generale**.

- In un **array**, gli indici devono essere **numeri interi**; in un **dizionario** possono essere (quasi) **qualsiasi tipo**.
- Un **dizionario** contiene una **raccolta di indici**, denominati **chiavi**, e una raccolta di **valori**.
- Ogni **chiave** è associata a un **singolo valore**.

Mappatura dalle chiavi ai valori

Un **dizionario** è come un **array**, ma **più generale**.

- In un **array**, gli indici devono essere **numeri interi**; in un **dizionario** possono essere (quasi) **qualsiasi tipo**.
- Un **dizionario** contiene una **raccolta di indici**, denominati **chiavi**, e una raccolta di **valori**.
- Ogni **chiave** è associata a un **singolo valore**.
- L'associazione di una **chiave** e di un **valore** è chiamata **coppia chiave-valore** o talvolta **elemento**.

Mappatura dalle chiavi ai valori

Un **dizionario** è come un **array**, ma **più generale**.

- In un **array**, gli indici devono essere **numeri interi**; in un **dizionario** possono essere (quasi) **qualsiasi tipo**.
- Un **dizionario** contiene una **raccolta di indici**, denominati **chiavi**, e una raccolta di **valori**.
- Ogni **chiave** è associata a un **singolo valore**.
- L'associazione di una **chiave** e di un **valore** è chiamata **coppia chiave-valore** o talvolta **elemento**.

Mappatura dalle chiavi ai valori

Un **dizionario** è come un **array**, ma **più generale**.

- In un **array**, gli indici devono essere **numeri interi**; in un **dizionario** possono essere (quasi) **qualsiasi tipo**.
- Un **dizionario** contiene una **raccolta di indici**, denominati **chiavi**, e una raccolta di **valori**.
- Ogni **chiave** è associata a un **singolo valore**.
- L'associazione di una **chiave** e di un **valore** è chiamata **coppia chiave-valore** o talvolta **elemento**.

In linguaggio matematico, un dizionario rappresenta un **mapping dalle chiavi ai valori**, quindi si può anche dire che **ogni chiave "mappa" un valore**.

Ad esempio, creeremo un **dizionario** che mappa le **parole dall'inglese allo spagnolo**, quindi le **chiavi** e i **valori** sono tutte **stringhe**.

Built-in function Dict

La **funzione Dict** crea un **nuovo dizionario senza elementi** (vuoto).

Poiché Dict è il **nome** di una **funzione** predefinita, dovresti **evitare di usarlo** come nome di **variabile**.

```
julia> eng2sp = Dict()  
Dict{Any,Any} with 0 entries
```

- Il **tipo di dizionario** è racchiuso tra **parentesi graffe**: le “chiavi” sono di tipo “Any” e anche i “valori” sono di tipo “Any”.

Built-in function Dict

La **funzione Dict** crea un **nuovo dizionario senza elementi** (vuoto).

Poiché Dict è il **nome** di una **funzione** predefinita, dovresti **evitare di usarlo** come nome di **variabile**.

```
julia> eng2sp = Dict()  
Dict{Any,Any} with 0 entries
```

- Il **tipo di dizionario** è racchiuso tra **parentesi graffe**: le “chiavi” sono di tipo “Any” e anche i “valori” sono di tipo “Any”.
- Il dizionario è **vuoto**.

Built-in function Dict

La **funzione Dict** crea un **nuovo dizionario senza elementi** (vuoto).

Poiché Dict è il **nome** di una **funzione** predefinita, dovresti **evitare di usarlo** come nome di **variabile**.

```
julia> eng2sp = Dict()  
Dict{Any,Any} with 0 entries
```

- Il **tipo di dizionario** è racchiuso tra **parentesi graffe**: le “chiavi” sono di tipo “Any” e anche i “valori” sono di tipo “Any”.
- Il dizionario è **vuoto**.
- Per **aggiungere elementi** al dizionario, puoi utilizzare le **parentesi quadre**:

Built-in function Dict

La **funzione Dict** crea un **nuovo dizionario senza elementi** (vuoto).

Poiché Dict è il **nome** di una **funzione** predefinita, dovresti **evitare di usarlo** come nome di **variabile**.

```
julia> eng2sp = Dict()  
Dict{Any,Any} with 0 entries
```

- Il **tipo di dizionario** è racchiuso tra **parentesi graffe**: le “chiavi” sono di tipo “Any” e anche i “valori” sono di tipo “Any”.
- Il dizionario è **vuoto**.
- Per **aggiungere elementi** al dizionario, puoi utilizzare le **parentesi quadre**:

Built-in function Dict

La **funzione Dict** crea un **nuovo dizionario senza elementi** (vuoto).

Poiché Dict è il **nome** di una **funzione** predefinita, dovresti **evitare di usarlo** come nome di **variabile**.

```
julia> eng2sp = Dict()
Dict{Any,Any} with 0 entries
```

- Il **tipo di dizionario** è racchiuso tra **parentesi graffe**: le “chiavi” sono di tipo “Any” e anche i “valori” sono di tipo “Any”.
- Il dizionario è **vuoto**.
- Per **aggiungere elementi** al dizionario, puoi utilizzare le **parentesi quadre**:

```
julia> eng2sp["one"] = "uno";
```

Questa riga **crea un elemento** che mappa **dalla chiave “one” al valore “uno”**.

Esempio di dizionario

Se **stampiamo** di nuovo il **dizionario**, noi vediamo una **coppia** “chiave-valore” con una **freccia** “=>” tra “chiave” e “valore”:

```
julia> eng2sp
Dict{Any,Any} with 1 entry:
  "one" => "uno"
```

Questo **formato di output** è anche un **formato di input**.

Ad esempio, puoi creare un **nuovo dizionario** con tre elementi:

```
julia> eng2sp = Dict("one" => "uno", "two" => "dos", "three" => "tres")
Dict{String,String} with 3 entries:
  "two"  => "dos"
  "one"  => "uno"
  "three" => "tres"
```

Tutte le **chiavi** e i **valori** iniziali sono **stringhe**, quindi viene creato un **Dict{String, String}**.

Accesso ai valori del dizionario

Ma non è un problema perché gli **elementi** di un **dizionario** **non** vengono (quasi) mai **indicizzati con indici interi**.

Invece, **usiamo le chiavi** per **cercare i valori** corrispondenti:

```
julia> eng2sp["two"] "dos"
```

La chiave `two` "è sempre associata al valore `dos` quindi **l'ordine degli elementi non ha importanza**. Se la **chiave non è nel dizionario**, si ottiene un'**eccezione**:

```
julia> eng2sp["four"]  
ERROR: KeyError: key "four" not found
```

AVVERTIMENTO

- **L'ordine delle coppie** chiave-valore potrebbe **non essere lo stesso**.

Accesso ai valori del dizionario

Ma non è un problema perché gli **elementi** di un **dizionario** **non** vengono (quasi) mai **indicizzati con indici interi**.

Invece, **usiamo le chiavi** per **cercare i valori** corrispondenti:

```
julia> eng2sp["two"] "dos"
```

La chiave `two` "è sempre associata al valore `dos` quindi **l'ordine degli elementi non ha importanza**. Se la **chiave non è nel dizionario**, si ottiene un'**eccezione**:

```
julia> eng2sp["four"]  
ERROR: KeyError: key "four" not found
```

AVVERTIMENTO

- L'**ordine delle coppie** chiave-valore potrebbe **non essere lo stesso**.
- Se digiti lo stesso esempio sul tuo computer, potresti ottenere un **risultato diverso**.

Accesso ai valori del dizionario

Ma non è un problema perché gli **elementi** di un **dizionario** **non** vengono (quasi) mai **indicizzati con indici interi**.

Invece, **usiamo le chiavi** per **cercare i valori** corrispondenti:

```
julia> eng2sp["two"] "dos"
```

La chiave `two` "è sempre associata al valore `dos` quindi **l'ordine degli elementi non ha importanza**. Se la **chiave non è nel dizionario**, si ottiene un'eccezione:

```
julia> eng2sp["four"]
ERROR: KeyError: key "four" not found
```

AVVERTIMENTO

- L'**ordine delle coppie** chiave-valore potrebbe **non essere lo stesso**.
- Se digiti lo stesso esempio sul tuo computer, potresti ottenere un **risultato diverso**.
- In generale, **l'ordine degli "elementi" in un dizionario è imprevedibile**.

length and \in

La **funzione length** funziona anche sui **dizionari**; restituisce il **numero di coppie** chiave-valore:

```
julia> length(eng2sp)
3
```

La **funzione keys** restituisce una **collezione** con le **chiavi** del dizionario:

```
julia> ks = keys(eng2sp);
julia> print(ks)
["two", "one", "three"]
```

Ora puoi usare l'**operatore \in** per vedere se **qualcosa appare come una key** nel dizionario:

```
julia> "one" in ks
true
julia> "uno" in ks
false
```

Dizionario memorizzato come struttura dati “tabella hash”

Per vedere se qualcosa appare come un valore in un dizionario, puoi usare la funzione `values`, che restituisce una raccolta di valori, e quindi usare l'operatore `∈`:

```
julia> vs = values(eng2sp); julia> “uno” in vs true
```

L'operatore `∈` utilizza algoritmi diversi per array e dizionari.

- Per gli array, cerca gli elementi dell'array in ordine, come in Ricerca. Man mano che l'array si allunga, il tempo di ricerca aumenta in proporzione diretta.

Dizionario memorizzato come struttura dati “tabella hash”

Per vedere se qualcosa appare come un valore in un dizionario, puoi usare la funzione `values`, che restituisce una raccolta di valori, e quindi usare l'operatore `∈`:

```
julia> vs = values(eng2sp); julia> “uno” in vs true
```

L'operatore `∈` utilizza algoritmi diversi per array e dizionari.

- Per gli array, cerca gli elementi dell'array in ordine, come in Ricerca. Man mano che l'array si allunga, il tempo di ricerca aumenta in proporzione diretta.
- Per i dizionari, Julia usa un algoritmo chiamato tabella hash che ha una proprietà notevole: l'operatore `∈` impiega circa la stessa quantità di tempo, indipendentemente dal numero di elementi presenti nel dizionario.

Section 2

Dizionario come collezione di contatori

Dizionario con caratteri e contatori

Supponiamo che ti venga **assegnata una stringa** e che tu voglia **contare quante volte appare ogni lettera**. Ci sono diversi modi per farlo:

- **Puoi creare 26 variabili**, una per **ogni lettera** dell'alfabeto.

Quindi potresti **attraversare la stringa** e, per ogni carattere, incrementare il contatore corrispondente, probabilmente usando un **condizionale concatenato**.

Dizionario con caratteri e contatori

Supponiamo che ti venga **assegnata una stringa** e che tu voglia **contare quante volte appare ogni lettera**. Ci sono diversi modi per farlo:

- **Puoi creare 26 variabili**, una per **ogni lettera** dell'alfabeto.
Quindi potresti **attraversare la stringa** e, per ogni carattere, incrementare il contatore corrispondente, probabilmente usando un **condizionale concatenato**.
- Potresti **creare un array con 26 elementi**.
Quindi è possibile **convertire ogni carattere in un numero** (utilizzando la funzione predefinita `Int`), utilizzare il numero come indice nell'array e incrementare il contatore appropriato.

Dizionario con caratteri e contatori

Supponiamo che ti venga **assegnata una stringa** e che tu voglia **contare quante volte appare ogni lettera**. Ci sono diversi modi per farlo:

- **Puoi creare 26 variabili**, una per **ogni lettera** dell'alfabeto.
Quindi potresti **attraversare la stringa** e, per ogni carattere, incrementare il contatore corrispondente, probabilmente usando un **condizionale concatenato**.
- Potresti **creare un array con 26 elementi**.
Quindi è possibile **convertire ogni carattere in un numero** (utilizzando la funzione predefinita `Int`), utilizzare il numero come indice nell'array e incrementare il contatore appropriato.
- È possibile **creare un dizionario** con **caratteri come chiavi** e **contatori come valori** corrispondenti.

Dizionario con caratteri e contatori

Supponiamo che ti venga **assegnata una stringa** e che tu voglia **contare quante volte appare ogni lettera**. Ci sono diversi modi per farlo:

- **Puoi creare 26 variabili**, una per **ogni lettera** dell'alfabeto.
Quindi potresti **attraversare la stringa** e, per ogni carattere, incrementare il contatore corrispondente, probabilmente usando un **condizionale concatenato**.
- Potresti **creare un array con 26 elementi**.
Quindi è possibile **convertire ogni carattere in un numero** (utilizzando la funzione predefinita `int`), utilizzare il numero come indice nell'array e incrementare il contatore appropriato.
- È possibile **creare un dizionario** con **caratteri come chiavi** e **contatori come valori** corrispondenti.
- La **prima volta** che vedi un carattere, **aggiungi un elemento** al dizionario. Dopodiché aumenterai il valore degli elementi già in esso.

Dizionario con caratteri e contatori

Supponiamo che ti venga **assegnata una stringa** e che tu voglia **contare quante volte appare ogni lettera**. Ci sono diversi modi per farlo:

- **Puoi creare 26 variabili**, una per **ogni lettera** dell'alfabeto.
Quindi potresti **attraversare la stringa** e, per ogni carattere, incrementare il contatore corrispondente, probabilmente usando un **condizionale concatenato**.
- Potresti **creare un array con 26 elementi**.
Quindi è possibile **convertire ogni carattere in un numero** (utilizzando la funzione predefinita `Int`), utilizzare il numero come indice nell'array e incrementare il contatore appropriato.
- È possibile **creare un dizionario** con **caratteri come chiavi** e **contatori come valori** corrispondenti.
- La **prima volta** che vedi un carattere, **aggiungi un elemento** al dizionario. Dopodiché aumenterai il valore degli elementi già in esso.
- Ciascuna di queste **opzioni** esegue lo **stesso calcolo**, ma ciascuna di esse **implementa** tale calcolo in un **modo diverso**.

Implementazione di “istogramma”

Un'implementazione è un modo per eseguire un calcolo; alcune implementazioni sono migliori di altre.

Ad esempio, un vantaggio dell'implementazione col dizionario è che non dobbiamo sapere in anticipo quali lettere compaiono nella stringa e dobbiamo solo fare spazio per le lettere che compaiono. Ecco come potrebbe apparire il codice:

```
function histogram(s)
  d = Dict()
  for c in s
    if c !in keys(d)
      d[c] = 1
    else
      d[c] += 1
    end
  end
  d
end
```

Il nome della funzione è `histogram`, che è un termine statistico* per una raccolta di contatori (o frequenze).

Esempio d'uso della funzione “istogramma”

La **prima riga** della funzione crea un **dizionario vuoto**.

Il **ciclo for attraversa la stringa**. Ogni volta nel ciclo, se il carattere **c non è** nel dizionario, **creiamo un nuovo elemento** con la chiave **c** e il valore iniziale **1** (poiché abbiamo visto questa lettera una volta).

Se **c è già nel dizionario**, **incrementiamo d[c]**. Ecco come funziona:

```
julia> h = histogram("brontosaurus")
Dict{Any,Any} with 8 entries:
```

```
'n' => 1
's' => 2
'a' => 1
'r' => 2
't' => 1
'o' => 2
'u' => 2
'b' => 1
```

L'**istogramma** indica che le lettere **a** e **b** **appaiono una volta**; **o** appare **due volte** e così via.

Funzione get

I dizionari hanno una **funzione chiamata get** che accetta una **chiave** e un valore predefinito (**default**).

Se la chiave **appare nel dizionario**, get restituisce il **valore corrispondente**; **altrimenti** restituisce il **valore di default**.

Per esempio:

```
julia> h = histogram("a")
Dict{Any,Any} with 1 entry:
  'a' => 1
julia> get(h, 'a', 0)
1
julia> get(h, 'b', 0)
0
```

Section 3

Ciclare sui dizionari

Stampa delle chiavi

Puoi attraversare le “chiavi” del dizionario in un’istruzione `for`.

Ad esempio, `printhist` stampa ogni chiave e il valore corrispondente:

```
function printhist(h)
    for c in keys(h)
        println(c, " = ", h[c])
    end
end
```

Ecco come appare l’output:

```
julia> h = histogram("parrot");
julia> printhist(h)
a = 1
r = 2
p = 1
o = 1
t = 1
```

Di nuovo, le “chiavi” non sono in un ordine particolare.

Attraversa le chiavi in “sorted” ordine

Per attraversare le chiavi in “sorted order”, puoi combinare `sort` e `collect`:

```
julia> for c in sort(collect(keys(h)))  
           println(c, " = ", h[c])  
       end
```

```
a = 1
```

```
o = 1
```

```
p = 1
```

```
r = 2
```

```
t = 1
```

Section 4

Ricerca inversa

Ricerca e ricerca inversa

Dato un dizionario d e una chiave k , è facile trovare il valore corrispondente $v = d[k]$. Questa operazione è chiamata *lookup*.

Ma cosa succede se hai v e vuoi trovare k ? Hai due problemi:

- in primo luogo, potrebbe esserci più di una chiave associata al valore v .

A seconda dell'applicazione, potresti essere in grado di sceglierne uno o potresti dover creare un array che li contenga tutti.

Ricerca e ricerca inversa

Dato un dizionario d e una chiave k , è facile trovare il valore corrispondente $v = d[k]$. Questa operazione è chiamata *lookup*.

Ma cosa succede se hai v e vuoi trovare k ? Hai due problemi:

- in primo luogo, potrebbe esserci più di una chiave associata al valore v .

A seconda dell'applicazione, potresti essere in grado di sceglierne uno o potresti dover creare un array che li contenga tutti.

- Secondo, non esiste una sintassi semplice per eseguire una ricerca inversa; devi cercare.

Ecco una funzione che prende un valore e restituisce la prima chiave associata a quel valore:

Ricerca e ricerca inversa

Dato un dizionario d e una chiave k , è facile trovare il valore corrispondente $v = d[k]$. Questa operazione è chiamata *lookup*.

Ma cosa succede se hai v e vuoi trovare k ? Hai due problemi:

- in primo luogo, potrebbe esserci più di una chiave associata al valore v .

A seconda dell'applicazione, potresti essere in grado di sceglierne uno o potresti dover creare un array che li contenga tutti.

- Secondo, non esiste una sintassi semplice per eseguire una ricerca inversa; devi cercare.

Ecco una funzione che prende un valore e restituisce la prima chiave associata a quel valore:

Ricerca e ricerca inversa

Dato un dizionario `d` e una chiave `k`, è facile trovare il valore corrispondente $v = d[k]$. Questa operazione è chiamata `lookup`.

Ma cosa succede se hai `v` e vuoi trovare `k`? Hai due problemi:

- in primo luogo, potrebbe esserci più di una chiave associata al valore `v`.

A seconda dell'applicazione, potresti essere in grado di sceglierne uno o potresti dover creare un array che li contenga tutti.

- Secondo, non esiste una sintassi semplice per eseguire una ricerca inversa; devi cercare.

Ecco una funzione che prende un valore e restituisce la prima chiave associata a quel valore:

```
function reverselookup(d, v)
  for k in keys(d)
    if d[k] == v
      return k
    end
  end
  error("LookupError")
end
```

Funzione error

La **funzione precedente** è ancora un altro **esempio** del **pattern di ricerca**, ma utilizza una funzione che non abbiamo **mai visto prima**, **error**.

La funzione **error** è usata **per produrre una `ErrorException`** che **interrompe** il normale **flusso del controllo**.

In questo caso **fornisce il messaggio** `LookupError`, che indica che **una chiave non esiste**.

Se arriviamo alla **fine del ciclo**, significa che **v non appare** nel dizionario come valore, quindi **lanciamo una “eccezione”**.

Ricerche inverse

Ecco un esempio di una **ricerca inversa riuscita**:

```
julia> h = histogram("parrot");
julia> key = reverselookup(h, 2)
'r': ASCII/Unicode U+0072 (category Ll: Letter, lowercase)
```

E una **senza successo**:

```
julia> key = reverselookup(h, 3)
ERROR: LookupError
```

L'effetto quando si **genera un'eccezione** è lo **stesso di quando Julia** ne lancia una: **stampa uno stacktrace** e un **messaggio di errore**.

Julia fornisce un **modo ottimizzato** per eseguire una **ricerca inversa**: `findall(isequal(3), h)`.

AVVERTIMENTO

Una **ricerca inversa** è **molto più lenta** di una **ricerca diretta**; se devi farlo spesso, o se il dizionario diventa grande, le **prestazioni** del tuo programma **ne risentiranno**.

Section 5

Dizionari e array

invertdict per invertire un dizionario

Gli `array` possono apparire come `valori` in un `dizionario`.

Ad esempio, se ti viene fornito un `dizionario` che `mappa dalle lettere alle frequenze`, potresti `voloerlo invertire`; ovvero, creare un dizionario che mappa dalle `frequenze alle lettere`.

Poiché potrebbero esserci più lettere con la stessa frequenza, `ogni valore` nel `dizionario invertito` dovrebbe essere un `array di lettere`. Ecco una `funzione invertdict` che `inverte un dizionario`:

```
function invertdict(d)
  inverse = Dict()
  for key in keys(d)
    val = d[key]
    if val !in keys(inverse)
      inverse[val] = [key]
    else
      push!(inverse[val], key)
    end
  end
  inverse
end
```

Esempio di inversione di dizionario

Ogni volta che si **esegue il ciclo**, `key` ottiene una chiave inversa da `val` il valore corrispondente.

Se `val` non è in `inverse`, significa che **non** l'abbiamo vista prima, quindi creiamo un nuovo elemento e lo **inizializziamo con un singleton** (un array che contiene un singolo elemento).

Altrimenti abbiamo già visto questo valore, quindi **“aggiungiamo” la chiave corrispondente all'array**.

Ecco un esempio:

```
julia> hist = histogram("parrot");
```

```
julia> inverse = invertdict(hist)
```

```
Dict{Any,Any} with 2 entries:
```

```
  2 => ['r']
```

```
  1 => ['a', 'p', 'o', 't']
```

Section 6

Promemoria

call graph per la funzione fibonacci

Se hai giocato con la **funzione fibonacci** di One More Example, potresti aver notato che **più grande è l'argomento** che fornisci, **più tempo impiega** la funzione a essere eseguita.

Inoltre, il **tempo di esecuzione aumenta rapidamente**.

Per capire il motivo, si **consideri la figura**, che mostra il grafico delle chiamate per fibonacci con $n = 4$:

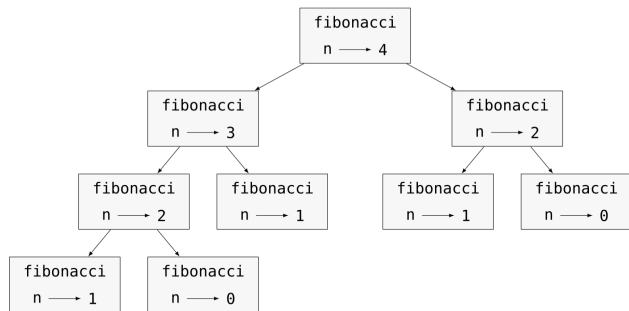


Figure 17. Call graph

Versione “memoizzata” di fibonacci 1/2

Un grafico delle chiamate mostra un insieme di frame di funzione, con linee che collegano ogni frame ai frame delle funzioni che chiama.

Nella parte superiore del grafico, fibonacci con $n = 4$ chiama fibonacci con $n = 3$ e $n = 2$. A sua volta, fibonacci con $n = 3$ chiama fibonacci con $n = 2$ e $n = 1$.

Conta quante volte vengono chiamati fibonacci(0) e fibonacci(1).

Questa è una soluzione inefficiente al problema, e peggiora man mano che l'argomento diventa più grande.

Una soluzione è tenere traccia dei valori che sono già stati calcolati memorizzandoli in un dizionario.

Un valore calcolato in precedenza che viene memorizzato per un uso successivo è chiamato “memo”. Ecco una versione “memoizzata” di fibonacci:

Versione “memoizzata” di fibonacci 2/2

```
known = Dict{0=>0, 1=>1};  
  
function fibonacci(n)  
    if n in keys(known)  
        return known[n]  
    end  
    res = fibonacci(n-1) + fibonacci(n-2)  
    known[n] = res  
    res  
end
```

“known” è un dizionario che tiene traccia dei numeri di Fibonacci che già conosciamo. Inizia con due elementi: $0 \mapsto 0$ e $1 \mapsto 1$.

Ogni volta che fibonacci viene chiamato, controlla known. Se il risultato è già presente, può tornare immediatamente. Altrimenti deve calcolare il nuovo valore, aggiungerlo al dizionario e restituirlo.

Esegui questa versione di fibonacci e confrontala con l'originale: è molto più veloce.

Section 7

Variabili globali

Le variabili in Main sono global

Nell'esempio precedente, `known` viene creato al di fuori della funzione, quindi appartiene al modulo speciale chiamato `Main`.

Le variabili in `Main` sono talvolta chiamate `global` perché sono accessibili da qualsiasi funzione.

A differenza delle variabili `local`, che scompaiono quando la loro funzione termina, le variabili `global` persistono da una chiamata di funzione a quella successiva.

È comune usare le variabili `global` per `flags`; ovvero, variabili booleane che indicano (`flag = "bandiera"`) se una condizione è true.

Variabili flag

Ad esempio, alcuni programmi **usano un flag** chiamato **verbose** per controllare il livello di **dettaglio nell'output**:

```
verbose = true;
```

```
function example1()  
    if verbose  
        println("Running example1")  
    end  
end
```

Se provi a **riassegnare una variabile globale**, potresti rimanere **sorpreso**. Il seguente **esempio dovrebbe tenere traccia** del fatto che la funzione **sia stata chiamata**:

global variable

```
been_called = false
```

```
function example2()  
    been_called = true # WRONG  
end
```

Ma se lo esegui vedrai che il **valore di `been_called` non cambia**.

Il problema è che `example2` crea una nuova variabile locale chiamata `been_called`. La **variabile locale scompare** quando la **funzione termina** e non ha alcun effetto sulla variabile `global`.

Per **riassegnare una variabile `global`** all'interno di una funzione **devi dichiarare la `variabileglobal`** prima di usarla:

```
been_called = false  
function example2()  
    global been_called  
    been_called = true  
end
```

L'istruzione `global`

L'istruzione `global` dice all'interprete qualcosa del tipo, "In questa funzione, quando dico `been_called`, intendo la variabile `global`; non crearne una `local`."

Ecco un esempio che tenta di aggiornare una variabile globale:

```
count = 0;
```

```
function example3()
count = count + 1 # WRONG
end
```

Se lo esegui ottieni:

```
julia> example3()
ERROR: UndefVarError: count not defined
```

Se una variabile globale è modificabile

Julia presume che `count` sia locale, e sotto questo presupposto lo stai leggendo prima di scriverlo.

La soluzione, ancora una volta, è dichiarare `count` globale.

```
count = 0
function example3()
    global count
    count += 1
end
```

Se una variabile globale fa riferimento a un valore mutabile, puoi modificare il valore senza dichiarare la variabile `global`:

```
known = Dict{0=>0, 1=>1};
function example4()
    known[2] = 1
end
```

Ambito di visibilità (scope) delle variabili

Per completa chiarezza su questo argomento difficile, si può consultare il [Manuale](#) del linguaggio.

Riassegnare una variabile globale

Quindi puoi **aggiungere**, **rimuovere** e **sostituire** elementi di un **array** o di un **dizionario globale**, ma se vuoi **riassegnare la variabile**, devi dichiararla `global`:

```
known = Dict{0=>0, 1=>1}
function example5()
    global known
    known = Dict{0=>0, 1=>1}()
end
```

Per motivi di **prestazioni**, dovresti dichiarare una **variabile globale costante**. Non è più possibile **riassegnare la variabile** ma se riferisce un **valore modificabile**, è possibile **modificare il valore**.

```
const known = Dict{0=>0, 1=>1}
function example4()
    known[2] = 1
end
```

AVVERTIMENTO

Le **variabili globali** possono essere **utili**, ma se ne hai molte e le modifichi frequentemente, possono **rendere i programmi difficili** da eseguire il debug e **funzionare male**.

Section 8

Debug

Debug di dataset di grandi dimensioni

Man mano che si lavora con set di dati più grandi, può diventare difficile eseguire il debug stampando e controllando l'output manualmente. Ecco alcuni suggerimenti per il debug di set di dati di grandi dimensioni:

- **Riduci l'input:**

Se possibile, riduci le dimensioni del set di dati. Ad esempio, se il programma legge un file di testo, inizia con solo le prime 10 righe o con l'esempio più piccolo per trovare eventuali errori. Non è possibile modificare i file stessi, ma piuttosto modificare il programma in modo che legga solo le prime n righe.

In caso di errore, è possibile ridurre n al valore più piccolo che manifesta l'errore, quindi aumentarlo gradualmente man mano che si trovano e correggono gli errori.

Debug di dataset di grandi dimensioni

Man mano che **si lavora** con set di dati **più grandi**, può diventare **difficile** eseguire il **debug** stampando e controllando l'output **manualmente**. Ecco alcuni **suggerimenti** per il debug di set di **dati di grandi dimensioni**:

- **Riduci l'input:**

Se possibile, **riduci le dimensioni** del set di dati. Ad esempio, se il programma legge un file di testo, inizia con solo le **prime 10 righe** o con l'**esempio più piccolo** per trovare eventuali errori. **Non è possibile modificare i file stessi**, ma piuttosto **modificare il programma** in modo che **legga solo le prime n righe**.

In caso di errore, è possibile **ridurre n al valore più piccolo** che **manifesta l'errore**, quindi aumentarlo gradualmente man mano che si **trovano e correggono** gli errori.

- **Controlla i riepiloghi e i tipi:**

Invece di stampare e controllare l'intero set di dati, prendi in considerazione la **stampa di riepiloghi dei dati**: ad esempio, il **numero di elementi** in un **dizionario** o il **totale** di una **matrice** di numeri.

Una causa comune degli **errori di runtime** è un valore che **non è del tipo corretto**. Per eseguire il debug di **questo tipo** di errore, spesso è **sufficiente stampare il tipo** di un valore.

Pattern di controllo di “sanità” e “consistenza” (coerenza)

- **Scrivi controlli automatici:**

A volte è possibile **scrivere codice** per **verificare automaticamente** la presenza di **errori**. Ad esempio, se stai **calcolando la media** di un array di numeri, potresti controllare che **il risultato** non sia **maggiore** dell'**elemento più grande** dell'array o **minore** del **più piccolo**.

Questo è chiamato “controllo di integrità”.

Pattern di controllo di “sanità” e “consistenza” (coerenza)

- **Scrivi controlli automatici:**

A volte è possibile **scrivere codice** per **verificare automaticamente** la presenza di **errori**. Ad esempio, se stai **calcolando la media** di un array di numeri, potresti controllare che **il risultato** non sia **maggiore** dell'**elemento più grande** dell'array o **minore** del **più piccolo**.

Questo è chiamato “controllo di integrità”.

- Un altro tipo di controllo **confronta i risultati** di due diversi **calcoli** per vedere se sono **coerenti**. Questo è chiamato “**controllo di coerenza**”.

Pattern di controllo di “sanità” e “consistenza” (coerenza)

- **Scrivi controlli automatici:**

A volte è possibile **scrivere codice** per **verificare automaticamente** la presenza di **errori**. Ad esempio, se stai **calcolando la media** di un array di numeri, potresti controllare che **il risultato** non sia **maggiore** dell'**elemento più grande** dell'array o **minore** del **più piccolo**.

Questo è chiamato “controllo di integrità”.

- Un altro tipo di controllo **confronta i risultati** di due diversi **calcoli** per vedere se sono **coerenti**. Questo è chiamato “**controllo di coerenza**”.

- **Formatta l'output:**

La **formattazione dell'output** di debug può **semplificare** l'individuazione di un errore. Abbiamo visto un esempio in Debugging.

Anche in questo caso, **il tempo impiegato** per la creazione dello scaffolding **può ridurre** il tempo impiegato per il **debug**.

Section 9

Glossario

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

dizionario Un **mapping** dalle chiavi ai valori corrispondenti.

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

dizionario Un **mapping** dalle chiavi ai valori corrispondenti.

coppia chiave-valore La rappresentazione del mapping **da una chiave a un valore**.

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

dizionario Un **mapping** dalle chiavi ai valori corrispondenti.

coppia chiave-valore La rappresentazione del mapping **da una chiave a un valore**.

articolo In un **dizionario**, un altro nome per una coppia **chiave-valore**.

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

dizionario Un **mapping** dalle chiavi ai valori corrispondenti.

coppia chiave-valore La rappresentazione del mapping **da una chiave a un valore**.

articolo In un **dizionario**, un altro nome per una coppia **chiave-valore**.

chiave Oggetto che appare in un dizionario come **prima parte** di una **coppia** chiave-valore.

Glossario I

- mapping** Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.
- dizionario** Un **mapping** dalle chiavi ai valori corrispondenti.
- coppia chiave-valore** La rappresentazione del mapping **da una chiave a un valore**.
- articolo** In un **dizionario**, un altro nome per una coppia **chiave-valore**.
- chiave** Oggetto che appare in un dizionario come **prima parte** di una **coppia** chiave-valore.
- valore** Un oggetto che appare in un dizionario come **seconda parte** di una **coppia** chiave-valore. Questo è più specifico del nostro precedente uso della parola “valore”.

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

dizionario Un **mapping** dalle chiavi ai valori corrispondenti.

coppia chiave-valore La rappresentazione del mapping **da una chiave a un valore**.

articolo In un **dizionario**, un altro nome per una coppia **chiave-valore**.

chiave Oggetto che appare in un dizionario come **prima parte** di una **coppia** chiave-valore.

valore Un oggetto che appare in un dizionario come **seconda parte** di una **coppia** chiave-valore. Questo è più specifico del nostro precedente uso della parola “valore”.

implementazione Un **modo per eseguire** un calcolo.

Glossario I

mapping Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.

dizionario Un **mapping** dalle chiavi ai valori corrispondenti.

coppia chiave-valore La rappresentazione del mapping **da una chiave a un valore**.

articolo In un **dizionario**, un altro nome per una coppia **chiave-valore**.

chiave Oggetto che appare in un dizionario come **prima parte** di una **coppia** chiave-valore.

valore Un oggetto che appare in un dizionario come **seconda parte** di una **coppia** chiave-valore. Questo è più specifico del nostro precedente uso della parola “valore”.

implementazione Un **modo per eseguire** un calcolo.

tabella hash L'**algoritmo** utilizzato per implementare i **dizionari Julia**.

Glossario I

- mapping** Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.
- dizionario** Un **mapping** dalle chiavi ai valori corrispondenti.
- coppia chiave-valore** La rappresentazione del mapping **da una chiave a un valore**.
- articolo** In un **dizionario**, un altro nome per una coppia **chiave-valore**.
- chiave** Oggetto che appare in un dizionario come **prima parte** di una **coppia** chiave-valore.
- valore** Un oggetto che appare in un dizionario come **seconda parte** di una **coppia** chiave-valore. Questo è più specifico del nostro precedente uso della parola “valore”.
- implementazione** Un **modo per eseguire** un calcolo.
- tabella hash** L'**algoritmo** utilizzato per implementare i **dizionari Julia**.
- funzione hash** Una **funzione** utilizzata da una **tabella hash** per calcolare la **posizione** di una chiave.

Glossario I

- mapping** Una **relazione** in cui ogni elemento di un insieme corrisponde a un elemento di un altro insieme.
- dizionario** Un **mapping** dalle chiavi ai valori corrispondenti.
- coppia chiave-valore** La rappresentazione del mapping **da una chiave a un valore**.
- articolo** In un **dizionario**, un altro nome per una coppia **chiave-valore**.
- chiave** Oggetto che appare in un dizionario come **prima parte** di una **coppia** chiave-valore.
- valore** Un oggetto che appare in un dizionario come **seconda parte** di una **coppia** chiave-valore. Questo è più specifico del nostro precedente uso della parola “valore”.
- implementazione** Un **modo per eseguire** un calcolo.
- tabella hash** L'**algoritmo** utilizzato per implementare i **dizionari Julia**.
- funzione hash** Una **funzione** utilizzata da una **tabella hash** per calcolare la **posizione** di una chiave.
- hashable** Un **tipo** che ha una funzione hash.

Glossario II

lookup Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.

Glossario II

lookup Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.

ricerca inversa Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.

Glossario II

lookup Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.

ricerca inversa Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.

singleton Un array (o un'altra **sequenza**) con un **singolo elemento**.

Glossario II

lookup Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.

ricerca inversa Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.

singleton Un array (o un'altra **sequenza**) con un **singolo elemento**.

call graph Un **diagramma** che mostra ogni frame creato durante l'esecuzione di un **programma**, con una freccia da ogni **chiamante** a ogni **chiamato**.

Glossario II

- lookup** Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.
- ricerca inversa** Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.
- singleton** Un array (o un'altra **sequenza**) con un **singolo elemento**.
- call graph** Un **diagramma** che mostra ogni frame creato durante l'esecuzione di un programma, con una freccia da ogni **chiamante** a ogni **chiamato**.
- memo** Un **valore** calcolato **memorizzato** per evitare calcoli futuri non necessari.

Glossario II

- lookup** Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.
- ricerca inversa** Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.
- singleton** Un array (o un'altra **sequenza**) con un **singolo elemento**.
- call graph** Un **diagramma** che mostra ogni frame creato durante l'**esecuzione di un programma**, con una freccia da ogni **chiamante** a ogni **chiamato**.
- memo** Un **valore** calcolato **memorizzato** per evitare calcoli futuri non necessari.
- variabile globale** Una variabile **definita all'esterno di una funzione**. È possibile accedere alle variabili globali da qualsiasi funzione.

Glossario II

- lookup** Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.
- ricerca inversa** Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.
- singleton** Un array (o un'altra **sequenza**) con un **singolo elemento**.
- call graph** Un **diagramma** che mostra ogni frame creato durante l'esecuzione di un programma, con una freccia da ogni **chiamante** a ogni **chiamato**.
- memo** Un **valore** calcolato **memorizzato** per evitare calcoli futuri non necessari.
- variabile globale** Una variabile **definita all'esterno di una funzione**. È possibile accedere alle variabili globali da qualsiasi funzione.
- dichiarazione globale** Un'istruzione che dichiara un **nome** di variabile come `global`.

Glossario II

- lookup** Un'operazione di dizionario che accetta una **chiave** e **trova il valore** corrispondente.
- ricerca inversa** Un'operazione di dizionario che accetta un **valore** e **trova una o più chiavi** associate ad esso.
- singleton** Un array (o un'altra **sequenza**) con un **singolo elemento**.
- call graph** Un **diagramma** che mostra ogni frame creato durante l'esecuzione di un programma, con una freccia da ogni **chiamante** a ogni **chiamato**.
- memo** Un **valore** calcolato **memorizzato** per evitare calcoli futuri non necessari.
- variabile globale** Una variabile **definita all'esterno di una funzione**. È possibile accedere alle variabili globali da qualsiasi funzione.
- dichiarazione globale** Un'istruzione che dichiara un **nome** di variabile come `global`.
- flag** Una **variabile booleana** utilizzata per indicare se una **condizione** è vera.

Glossario III

dichiarazione Un'**istruzione** come `global` che dice all'**interprete** qualcosa su una variabile.

Glossario III

dichiarazione Un'**istruzione** come `global` che dice all'**interprete** qualcosa su una variabile.

variabile globale costante Una variabile globale che **non** può essere **riassegnata**.

Section 10

Esercizi

aaaa

aaaa