Un algoritmo per l'estrazione di radice quadrata con simboli Maya

Andrea De Domenico

18 Giugno 2018
Versione 1.0

Indice

1 Introduzione

L'obiettivo dell'articolo è quello di dare una descrizione il più completa possibile dell'algoritmo per l'estrazione della radice quadrata (la sua parte intera più il "resto") usando l'aritmetica Maya [1]. Si suppone che il lettore sappia cosa sia e come funzioni un abaco Maya [2].

2 Descrizione dell'algoritmo

2.1 Formalizzazione

In questo capitolo definiremo l'algoritmo in modo rigoroso.

n : radicando

x : radice

$\lfloor \log_{10} n \rfloor + 1$: numero di cifre del radicando (in base 10)

$k = \lceil \frac{\lfloor \log_{10} n \rfloor + 1}{2} \rceil$: numero di cifre della radice (in base 10)

z : variabile di lavoro

Siano m, a, b tre interi non negativi.

Enumerando le cifre della rappresentazione in base 10 di m a partire dalla meno significativa, iniziando da 0, con "$m[a, b]$" indicheremo il numero ottenuto estraendo dalla rappresentazione posizionale in base 10 di m la sequenza di cifre dalla posizione a-esima alla b- esima;

più precisamente: se $a < b$ allora $m[a, b] = 0$, altrimenti $m[a, b] = \lfloor \frac{m}{10^b} \rfloor \ mod \ 10^{a+1}$. Per comodità, se a è maggiore o uguale del numero di cifre della rappresentazione di n in forma normale, si considera l'estensione della rappresentazione di m con zeri a sinistra, il numero restituito avrà eventualmente degli zeri a sinistra.

Algoritmo in pseudocodice:

```
0.      radice(n) {
1.          x = 0;
2.          k = ceil((floor(log10(n))+1)/2);
3.          r = n;
4.          for(j = 1; j <= k; j++) {
5.              z[2k-1, 2(k-j)] = 2x[k-1, k-j] + 1;
6.              while(r[2k-1, 2(k-j)] >= z[2k-1, 2(k-j)]) {
7                   r[2k-1, 2(k-j)] -= z[2k-1, 2(k-j)];
8.                  x[k-1, k-j]++;
9.                  z[2k-1, 2(k-j)] += 2;
10.             }
```

```
11.          }
12.          return (x, r);
13.      }
```

2.2 Dimostrazione della correttezza

Una buona strategia per dimostrare la correttezza di semplici algoritmi iterativi, è quella di fare uso degli *invarianti di ciclo*, una proposizione P è un invariante di ciclo se gode delle seguenti proprietà

Inizializzazione: P deve essere vera prima che inizi il ciclo.

Mantenimento: Se P è vera alla fine dell' i-esima iterazione, allora P dev'essere vera anche alla fine dell'iterazione successiva.

Terminazione: alla fine del ciclo, P vera implica che l'algoritmo è corretto (ovvero produce l'output desiderato).

P(j): $0 \leq$ r[2k-1, 2(k-j)] = n[2k-1, 2(k-j)] - x[k-1, k-j]$^2 \leq$ 2x[k-1, k-j].

Osserviamo preliminarmente che il ciclo *while* interno certamente termina: i due termini nella condizione sono monotoni in senso opposto, di cui uno strettamente.

Prima di iniziare la dimostrazione verranno introdotti alcuni lemmi e definizioni.

S_j: numero totale di iterazioni del ciclo interno nel j-esimo ciclo esterno;
Quindi S_j = x[k-j, k-j], per definizione. (1)

$\phi_{j,0}$: r[2k-1, 2(k-j)] prima di entrare nel ciclo interno.

ϕ_{j,S_j}: r[2k-1, 2(k-j)] alla fine del ciclo interno.

ζ_{j,S_j}: z[2k-1, 2(k-j)] alla fine del ciclo interno.

$\zeta_{j,0}$: x[k-1, k-j] prima di entrare nel ciclo interno. (x[k-1, k.j]$_{j,0}$)

$\zeta_{j,S_j} = 2\zeta_{j,0} + 1 + 2S_j$ (2)
infatti, una volta terminato il ciclo interno,
z[2k-1, 2(k-j)] = 2x[k-1, k-j]$_{j,0}$ + 1 +2x[k-j, k-j].

$\phi_{j,S_j} = \phi_{j,0}$ - ($2S_j\zeta_{j,0} + S_j^2$) (3)
infatti $\phi_{j,S_j} = \phi_{j,0}$ - ($2\zeta_{j,0}$ + 1) - ... - ($2\zeta_{j,0}$ + 2S_j + 1).

Iniziamo a dimostrare la proprietà di inizializzazione con j=1.

r[2k-1, 2(k-1)] = $\phi_{1,S_1} \leq 0$, infatti a ϕ_{1,S_1} non può venire sottratto un numero maggiore di se stesso, a causa della condizione nel while. Questo varrà anche nel caso generale.

Abbiamo $\phi_{1,0}$ = n[2k-1, 2(k-1)] e $\zeta_{1,0}$ = x[k-1, k-1]$_{1,0}$ = 0. Utilizzando la (1) e la (3) otteniamo ϕ_{1,S_1} = n[2k-1, 2(k-1)] - x[k-1, k-1]^2.

Rimane da dimostrare che $\phi_{1,S_1} \leq$ 2x[k-1, k-1]; dalla definizione di S_1, la condizione del ciclo interno deve essere falsa, quindi, usando la (2),

$\phi_{1,S_1} < \zeta_{1,S_1} = 2\zeta_{1,0} + 1 + 2S_1$ = 2x[k-1, k-1] + 1.

Dopo aver dimostrato $P(1)$, passiamo al caso generale P(j) per dimostrare la proprietà di mantenimento. Sfruttando l'ipotesi induttiva otteniamo:

$\phi_{j,0} = 100\phi_{j\text{-}1,S_{j-1}}$ + n[2(k-j)+1, 2(k-j)] =

= 100n[2k-1, 2(k-j+1)] - 100x[k-1, k-j+1]^2 + n[2(k-j)+1, 2(k-j)] =

= n[2k-1, 2(k-j)] - 100x[k-1, k-j+1]^2.

$\zeta_{j,0}$ = 10x[k-1, k-j+1]

Quindi:

$\phi_{j,S_j} = \phi_{j,0}$ - ($2S_j\zeta_{j,0} + S_j^2$) =

= n[2k-1, 2(k-j)] - 100x[k-j, k-j+1]^2 - 20x[k-1, k-j+1]x[k-j, k-j] - x[k-j, k-j]^2

= n[2k-1, 2(k-j)] - x[k-1, k-j]^2 (si omettono i calcoli).

Infine, terminato il ciclo interno la sua condizione è falsa, quindi:

$\phi_{j,S_j} < \zeta_{j,S_j} = 2\zeta_{j,0} + 1 + 2S_j$ =

= 20x[k-1, k-j+1] + 1 + 2x[k-j, k-j] = 2x[k-1, k-j] + 1.

Resta da dimostrare solo la proprietà di terminazione. Osserviamo come P(k) implica r = n - x$^2 \leq$ 2x. In conclusione quindi x è la parte intera della radice di n e l'algoritmo è corretto.

3 Calcolo della complessità

Per calcolare la complessità dell'algoritmo, si userà la stessa tecnica presente in [3], indicheremo con C_i il costo computazionale della i-esima riga dello pseudocodice dell'algoritmo (al paragrafo 2.2).

$$\begin{aligned} T(n) &= C_1 + C_2 + C_3 + \sum_{j=1}^{k}(C_4 + C_5 + \sum_{i=1}^{s_j}(C_6 + C_7 + C_8 + C_9)) = \\ &= O(C_1 + C_2 + C_3 + k(C_4 + C_5 + s(C_6 + C_7 + C_8 + C_9))) = \\ &= O(C_1 + C_2 + C_3 + (C_4 + C_5)k + (C_6 + C_7 + C_8 + C_9)sk) = \\ &= O(k) = O(\log n) \end{aligned} \tag{1}$$

Ovviamente si ha che $k = \lceil \frac{\lfloor \log_{10} n \rfloor + 1}{2} \rceil = O(\log n)$, mentre s indica il massimo numero di volte che viene eseguito il ciclo *while* ad ogni ciclo esterno.

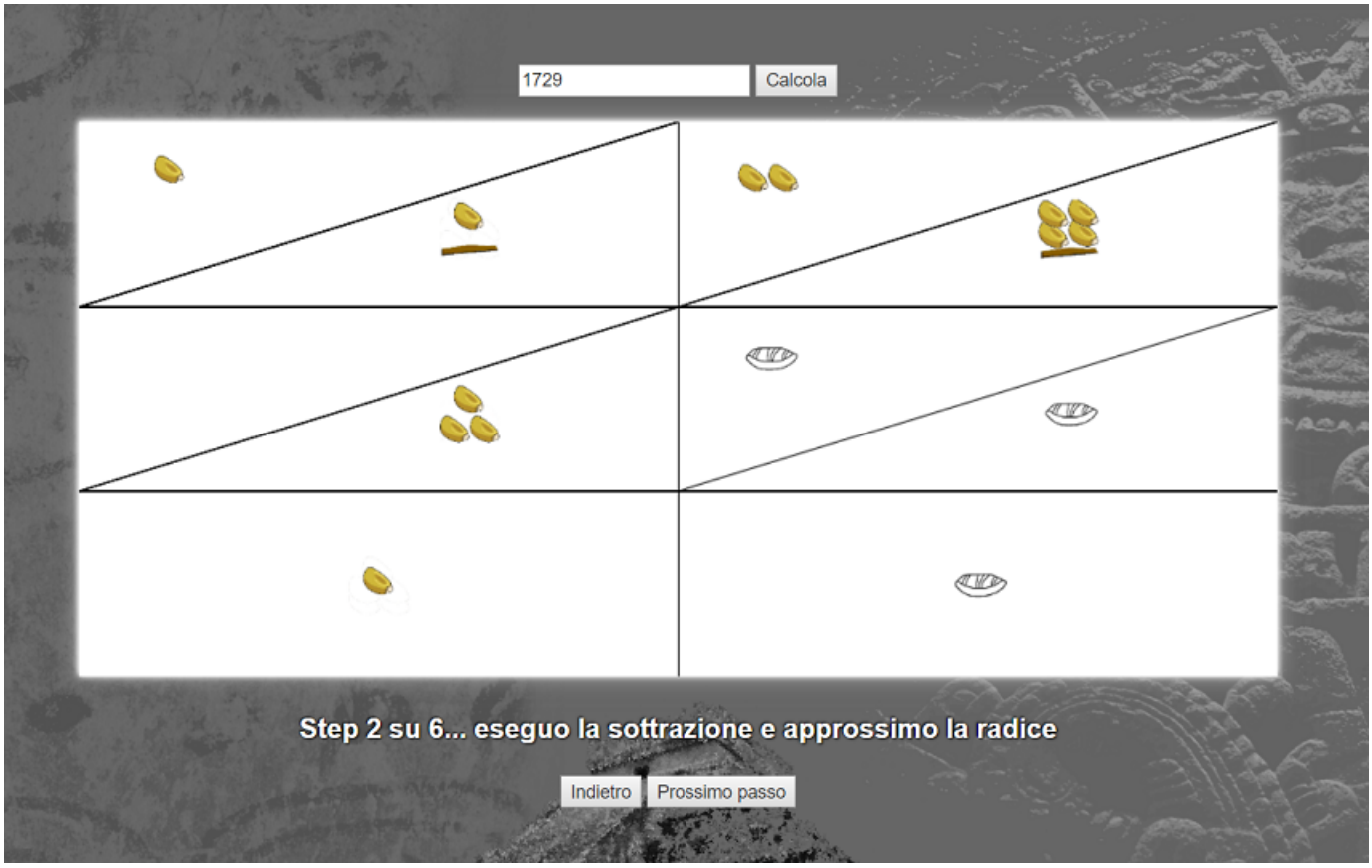
È necessario fare alcune precisazioni. Più realisticamente, si potrebbe pensare che C_2 non sia costante, ma ciò è irrilevante, perché il risultato rimarrebbe

invariato [4]. Inoltre, affinchè tutto funzioni, dobbiamo assicurarci che il termine s sia limitato; è sufficiente osservare che $0 \leq s < b$, dove b indica la base nella quale stiamo lavorando (10, in questo caso).

4 Realizzazione del simulatore

Il simulatore[5] fa uso di un abaco a tre righe con un numero arbitrariamente lungo di colonne (sufficienti per contenere il radicando), le caselle delle prime due righe sono divise a metà lungo la loro diagonale secondaria.

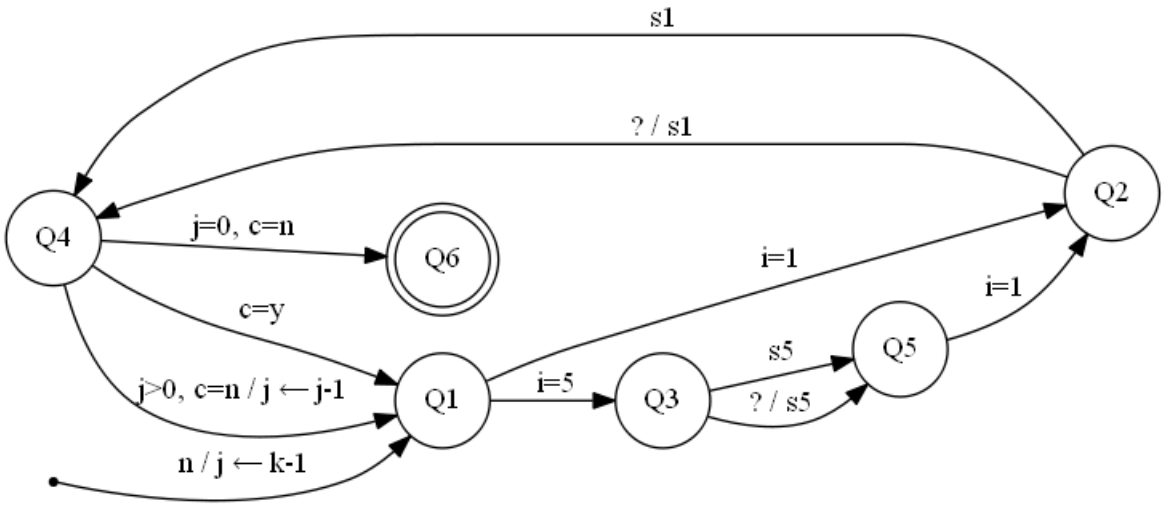
La prima riga contiene l'input (il radicando), la terza riga conterrà l'output (la radice), calcolato secondo approssimazioni successive, a partire dalla sua cifra più significativa. La riga centrale contiene il potenziale sottraendo. Le caselle dell'abaco sono allineate in modo tale che ad ogni coppia di cifre del radicando corrisponda verticalmente una cifra della radice, com'è facile osservare nel seguente screenshot.



Una volta inserito il radicando dalla textbox in alto, usando i due bottoni in basso, ogni passo dell'algoritmo verrà mostrato sull'abaco. Quando l'algoritmo avrà terminato, nella prima riga avremo il resto, nella terza riga ci sarà la parte intera della radice.

Oltre alla modalità base appena presentata, è in via di realizzazione una modalità esercizio, dove l'utente potrà esercitarsi a computare la radice quadrata usando l'abaco.

Ai lati dell'abaco ci saranno delle caselle che permetteranno di inserire gli elementi nell'abaco nella posizione desiderata sfruttando il *drag and drop*. Come nella modalità base, l'utente inizierà inserendo il radicando, ma eseguirà l'algoritmo da sé. L'utente, in alcuni punti ben definiti, potrà verificare se la configurazione dell'abaco è corretta con due pulsanti di verifica che si troveranno rispettivamente accanto alla prima ed alla seconda riga dell'abaco.



Diamo una descrizione del comportamento del simulatore in modalità esercizio usando un diagramma a stati, dove si astrae dal contenuto delle singole righe tenendo traccia delle colonne.

Ad ogni arco di transizione è associata un'etichetta del tipo "*evento/azione*". Se un *evento* (che è la causa del cambiamento di stato) è formato da più condizioni, queste sono separate da una virgola; allo scatenarsi dell'*evento* viene eseguita (se è specificata) la corrispondente *azione*.

Nello stato iniziale, l'utente inserisce n, il radicando, facendo arrivare il simulatore nello stato *Q1*.

La variabile j (inizializzata a $\lceil \frac{\lfloor \log_{10} n \rfloor + 1}{2} \rceil - 1$) tiene traccia della colonna di lavoro. A questo punto l'utente sceglie se tentare di aggiungere una (*i=1*) o cinque (*i=5*) unità all'attuale approssimazione della radice nella colonna di lavoro, mandando il simulatore nello stato *Q2* oppure *Q3*.

Raggiunto uno di questi due stati, l'utente proverà ad inserire il potenziale sottraendo nella riga centrale (se non ci riesce, potrà avere la risposta cliccando sull'apposito pulsante "*?*"). Inserito il sottraendo (*s1* o *s5* nel diagramma), si avrà la transizione dallo stato *Q2* allo stato *Q4* o dallo stato *Q3* allo stato *Q5*.

Dallo stato *Q5*, che sia stato o meno possibile effettuare la sottrazione, si potrà transire solo allo stato *Q2* con successivo incremento unitario dell'attuale approssimazione della radice nella colonna di lavoro.

Nello stato *Q4*, se è possibile effettuare la sottrazione (*c=y*), dopo averla svolta si torna nello stato *Q1*. Se invece non è possibile svolgere la sottrazione, se *j=0* sono state approssimate tutte le cifre della radice e quindi l'algoritmo termina (passando allo stato finale *Q6*; se *j>0* allora si passa alla colonna successiva (*j <- j-1*) e si torna allo stato *Q1*.

Nella modalità esercizio, per risparmiare tempo e rendere la computazione più efficiente, sarà possibile scegliere anche un incremento di cinque unità.

Per l'utente, è facile calcolare l'eventuale sottraendo corrispondente ad un aumento di cinque unità dell'approssimazione della radice (infatti $(x+5)^2 = x^2 + 10x + 25$), o accorgersi che l'incremento non è fattibile (anche prima di computare il sottraendo).

Questa opzione non è presente nella modalità base del simulatore, per mantenere l'algoritmo il più semplice possibile. Si può osservare che la descrizione permette sequenze di scelte poco convenienti, quali ad esempio provare un incremento di cinque unità dopo un incremento unitario o dopo che si è già effettuato un incremento di cinque. Questa scelta è motivata dalla semplicità della descrizione, ma l'implementazione potrà eventualmente realizzarla in maniera più restrittiva.

L'utente potrà verificare la configurazione dell'abaco nello stato *Q1* con il pulsante all'altezza della prima riga, negli stati *Q4* e *Q5* usando il pulsante accanto alla seconda riga. Il primo verificherà la correttezza della sottrazione effettuata sul residuo nella prima riga, mentre il secondo verificherà la correttezza del calcolo del sottraendo nella riga centrale.

5 Discussione

5.1 Uso dell'algoritmo nella didattica

Rispetto al metodo comunemente insegnato nelle scuole secondarie di primo grado, l'algoritmo presentato in questo articolo ben si presta ad essere presen-

tato in aula, non come un'immotivata sequenza di prescrizioni operative, bensì, come il risultato dello sviluppo di un ragionamento matematico, che parte da una definizione accurata della funzione da calcolare, sfruttando proprietà elementari (quadrato di binomio, posizionalità) e le analogie con la divisione (risultato con resto, approssimazione mediante sottrazioni successive), per dedurre e giustificare i passi operativi che costituiscono l'algoritmo.

La sua natura intuitiva permette inoltre di essere ricordato facilmente; l'apprendimento dell'algoritmo mira a sviluppare le capacità deduttive del discente piuttosto che quelle esecutive. Il simulatore prevede inoltre una modalità esercizio, dove sia discenti che docenti potranno fare pratica con l'algoritmo, per apprenderne il funzionamento.

5.2 Confronto con altri algoritmi

L'algoritmo di L. F. Magaña, illustrato attraverso degli esempi in [6], ha un'ispirazione simile a quello presentato nel presente articolo.

Infatti anch'esso individua un'analogia fra estrazione di radice quadrata e divisione ed è perciò anch'esso vantaggioso rispetto a quello tradizionale; tuttavia, è costruito su un ragionamento matematico diverso, basato sulla considerazione della divisione come operazione quasi-inversa della moltiplicazione.

Questo risulta in un algoritmo operativamente più complesso già per la divisione, rispetto a quello per sottrazioni successive. La maggiore semplicità del metodo per sottrazioni successive si ritrova anche nell'algoritmo per l'estrazione della radice quadrata.

l'articolo di Magaña presenta un esempio di approssimazione decimale di una radice; il nostro algoritmo può essere facilmente modificato per trattare anche questi casi.

Riferimenti bibliografici

[1] L. F. Magaña, *The ludic and powerful Mayan mathematics for teaching*, *Procedia - Social and Behavioral Sciences* 106 (2013) 2921–2930. https://www.sciencedirect.com/science/article/pii/S1877042813049616

[2] https://www.dmi.unict.it/archelab/archive/slidy/ae-2018/ae_e01/it/ae_e01.html

[3] Cormen, Leiserson, Rivest, Stein. *Introduction to algorithms*, The MIT Press, 2007 (Seconda edizione).

[4] https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

[5] https://www.dmi.unict.it/archelab/projects/ae2017unictradicemaya/

[6] L. F. Magaña, *La radice quadrata con l'aritmetica Maya*, *Calcolo Matematico Precolombiano*, Istituto Italo-LatinoAmericano, Roma, 21 Ottobre 2003; Atti del Convegno, IILA, Roma (2004), pp. 321–339.