Algoritmi randomizzati

Alessandro Verri

31 Maggio 2024

Sommario

Le lezioni su algoritmi randomizzati basate su queste note presuppongono un primo corso di algoritmi e richiedono dimestichezza con il linguaggio della teoria della probabilità. *La tastiera da sola non basta: per capire questi temi sono richieste anche carta e penna.* Gran parte del materiale è adattato o ispirato al contenuto di alcuni capitoli del *Motwani & Raghavan* [1]. Tutti sono benvenuti a inviare all'indirizzo `alessandro.verri@unige.it` rilievi e correzioni.

Indice

1 Introduzione

Introdurre elementi aleatori in un algoritmo può sembrare un'idea bizzarra. Vedremo, invece, che la casualità consente di progettare algoritmi efficienti e spesso più semplici della versione deterministica. Più difficili la valutazione dei costi computazionali attesi e la stima delle probabilità di successo. Limiteremo l'analisi a cinque casi prototipici.

1. Lontani dal caso peggiore: per un algoritmo deterministico il costo computazionale per l'input peggiore può essere molto più alto che nel caso medio. La costruzione di una variante capace di gestire meglio quell'input è spesso frustrata dal fatto che la variante avrà un costo computazionale elevato su un altro input. Mentre è facile costruire l'input peggiore per un algoritmo deterministico fissato, è di fatto impossibile riuscire a farlo per un algoritmo randomizzato. In questo ambito vedremo algoritmi *Las Vegas* per l'*ordinamento*, la *programmazione lineare* e la *valutazione dell'albero di un gioco* e stabiliremo una connessione con la *Teoria dei Giochi*.

Esercizio 1.1. *Mai rendere nota la propria strategia:* costruisci l'input peggiore per le versioni deterministiche di *QuickSort* che scelgono come *pivot* rispettivamente il **primo** e l'**elemento centrale** di una sequenza. Se gli elementi centrali sono due, il *pivot* è l'elemento di sinistra.
Soluzione: ipotizziamo che la sequenza sia costituita dai primi dieci numeri naturali ordinati arbitrariamente. Con l'input peggiore la scelta del *pivot* lascia sempre una delle due sotto-sequenze vuota. Se la scelta del *pivot* ricade sul **primo elemento**, pertanto, l'input peggiore risulta essere la *sequenza ordinata* (tabella in basso, a sinistra). Se ricade invece sull'**elemento centrale**, l'input peggiore - che si costruisce *partendo dal centro* (tabella in basso, a destra) - risulta essere la sequenza dei dispari decrescenti seguita dai pari crescenti.

1	X	X	X	X	X	X	X	X	X
1	**2**	X	X	X	X	X	X	X	X
1	2	**3**	X	X	X	X	X	X	X
1	2	3	**4**	X	X	X	X	X	X
1	2	3	4	**5**	X	X	X	X	X
1	2	3	4	5	**6**	X	X	X	X
1	2	3	4	5	6	**7**	X	X	X
1	2	3	4	5	6	7	**8**	X	X
1	2	3	4	5	6	7	8	**9**	X
1	2	3	4	5	6	7	8	9	**10**

X	X	X	X	**1**	X	X	X	X	X
X	X	X	X	1	**2**	X	X	X	X
X	X	X	**3**	1	2	X	X	X	X
X	X	X	3	1	2	**4**	X	X	X
X	X	**5**	3	1	2	4	X	X	X
X	X	5	3	1	2	4	**6**	X	X
X	**7**	5	3	1	2	4	6	X	X
X	7	5	3	1	2	4	6	**8**	X
9	7	5	3	1	2	4	6	8	X
9	7	5	3	1	2	4	6	8	**10**

2. Campionamento casuale: un campione casuale è rappresentativo della popolazione da cui è estratto. Se la soluzione di un certo problema è uno dei risultati possibili di un campionamento casuale, pertanto, possiamo pensare di risolvere quel problema con alta probabilità ripetendo il campionamento molte volte con un algoritmo *Monte Carlo*: non diversamente da quanto possiamo essere quasi certi di ottenere *testa* lanciando una moneta un grande numero di volte. In questo caso vedremo un algoritmo per la determinazione di un *taglio minimo per un grafo*.

Esercizio 1.2. *Quanti colori?* Un'urna contiene palline di al più tre colori: *rosso*, *bianco* e *verde*. Vuoi determinare quanti colori diversi sono rappresentati nell'urna estraendo a caso tre palline molte volte. Che cosa accade se l'urna contiene (1) 3 palline *rosse*, 2 *bianche* e 1 *verde* e (2) 98 *rosse*, 1 *bianca* e 1 *verde*?
Soluzione: Caso (1): la probabilità $p_1(rbv)$ di ottenere 1 pallina *rossa*, 1 *bianca* e 1 *verde* si ottiene facendo il rapporto tra casi favorevoli e casi possibili, ovvero

$$p_1(rbv) = \frac{\binom{3}{1}\binom{2}{1}\binom{1}{1}}{\binom{6}{3}} = \frac{6}{20} = 30\%$$

Ripetendo l'estrazione venti volte, la probabilità di **osservare** tre colori diversi è molto alta, poiché

$$1 - (1 - 0.3)^{20} = 1 - 0.7^{20} \simeq 99.9\%$$

Caso (2): ragionando come prima otteniamo

$$p_2(rbv) = \frac{\binom{98}{1}\binom{1}{1}\binom{1}{1}}{\binom{100}{3}} \simeq 0.1\%$$

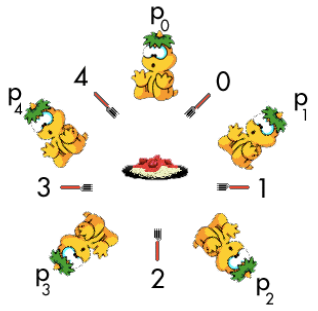
In questo caso, per **osservare** tre colori con la stessa probabilità del caso precedente dobbiamo ripetere l'estrazione diverse migliaia di volte poiché

$$1 - (1 - 0.001)^{6600} = 1 - 0.999^{6600} \simeq 99.9\%$$

3. Sblocco di uno stallo: in molti contesti di calcolo distribuito può capitare di cadere in stallo. Per ottenere una via d'uscita un algoritmo deterministico potrebbe essere costretto a effettuare analisi complesse e dispendiose. Vedremo un algoritmo *Monte Carlo* capace di sbloccare stalli in modo semplice ed efficiente nel problema dell'*accordo bizantino*.

Esercizio 1.3. *Saziarsi o fare la dieta?* Cinque filosofi, $p_0, p_1, \ldots, p_4$ sono seduti a un tavolo circolare, vedi Figura 1 a sinistra. A sinistra del filosofo p_i c'è la bacchetta i e al centro del tavolo una terrina di spaghetti. Ogni filosofo, quando ha fame, smette di pensare, prende una delle due bacchette vicine, poi l'altra, e comincia a mangiare. Quando si sente sazio, rilascia le bacchette, in un qualche ordine, e ricomincia a pensare. Ogni bacchetta può essere usata da un solo filosofo. Se un filosofo affamato non trova disponibile una bacchetta **non gli resta che aspettare**. In presenza di un avversario che può decidere quando ogni filosofo si sente affamato, è possibile trovare un algoritmo deterministico in grado di garantire

1. che se in un qualunque momento un filosofo è affamato allora un filosofo (non necessariamente lo stesso) si sazierà (*deadlock free*) e
2. che ogni filosofo affamato riuscirà a mangiare (*lockout free*)?




```
void filosofo_deterministico():
  while (TRUE):
    pensa()
    prendi_bacchetta_sinistra()
    prendi_bacchetta_destra()
    mangia()
    rilascia_bacchetta_sinistra()
    rilascia_bacchetta_destra()
```

Figura 1: Vedi testo.

Soluzione: No, non è possibile (Lehman and Rabin [2]). Se l'avversario sincronizza la fame dei filosofi, un algoritmo deterministico, vedi Figura 1 a destra per esempio, cade in stallo. Si ottiene una soluzione *deadlock free* se ogni filosofo decide quale bacchetta prendere affidandosi al caso, come in

```
void filosofo_randomizzato():
   pensa()
   prova = TRUE
   while (prova):
      scegli_lato() # uniformemente a caso
      attendi fino a che (bacchetta_disponibile(lato) = TRUE):
         bacchetta_disponibile(lato) = FALSE
      if (bacchetta_disponibile(altro lato) = TRUE):
         bacchetta_disponibile(altro lato) = FALSE
         prova = FALSE
      else:
         bacchetta_disponibile(lato) = TRUE
   mangia()
   bacchetta_disponibile(lato) = bacchetta_disponibile(altro lato) = TRUE
```

In presenza di un filosofo molto affamato, tuttavia, qualche filosofo potrebbe morire di fame. Il *lockout* si può evitare introducendo due variabili aggiuntive, una che informa il filosofo p_{i+1} che p_i ha fame e viceversa, e l'altra che registra chi tra p_i e p_{i+1} ha mangiato per ultimo.

4. Abbondanza di MR-testimoni: se la maggior parte degli elementi di un insieme molto grande soddisfa una certa proprietà, ovvero ne è MR-testimone, scelte casuali ripetute degli elementi dell'insieme consentono di rendere arbitrariamente piccola la probabilità di non campionare un MR-testimone. Armati di alcuni concetti fondamentali della *Teoria dei Numeri*, vedremo un algoritmo *Monte Carlo* che è utilizzabile (e nei fatti molto utilizzato) come *test di primalità*.

Esercizio 1.4. *Agitare prima dell'uso:* del milione di noccioline apparentemente tutte uguali che Pippo trova in una cassa, metà lo trasformano in Super Pippo per 24 ore mentre metà non hanno alcun effetto. Nell'ipotesi che nella cassa siano mescolate, quante noccioline deve mangiare Pippo per trasformarsi in Super Pippo con una probabilità del 99.9%?
Soluzione: la probabilità che Pippo non diventi Superpippo mangiando 10 noccioline è il complemento alla probabilità che tutte le 10 noccioline siano del tipo di quelle che non hanno effetto. Poiché i dieci eventi sono indipendenti avremo

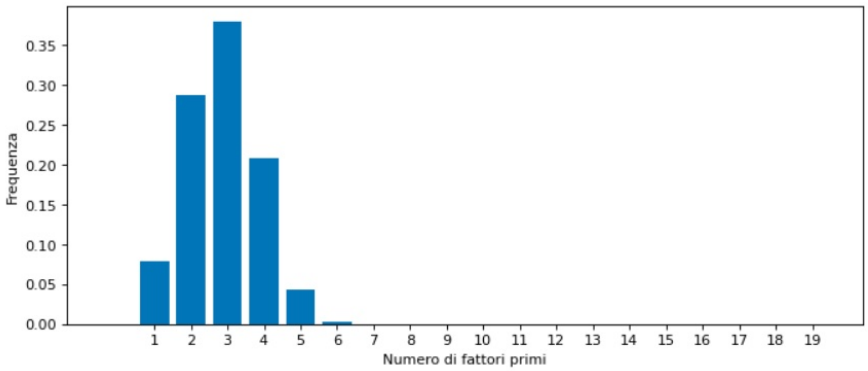
$$1 - \frac{1}{2^{10}} = 1 - \frac{1}{1024} = \frac{1023}{1024} \simeq 99.9\%$$

La Figura *2* illustra l'importanza dell'ipotesi di **mescolamento** dei due tipi di noccioline (rappresentate come palline rosse e bianche).

Figura 2: Vedi testo.

5. Riduzione casuale della dimensionalità: verifiche di uguaglianze in grandi dimensioni possono essere molto costose. Ridurre le dimensioni degli input mediante funzioni casuali, come nel caso del *fingerprinting*, consente con alta probabilità di rilevare differenze a costi molto contenuti. Vedremo algoritmi *Monte Carlo* per la verifica *del risultato del prodotto di due matrici* e *dell'uguaglianza di due file di grandi dimensioni*.

Esercizio 1.5. *La solitudine dei fattori primi:* verifica che i fattori primi di un qualunque numero minore di 2^ℓ sono solo una piccola frazione dei numeri primi minori di ℓ^2. Considera il caso $\ell = 20$.
Soluzione: I primi tra 2 e $20^2 = 400$ sono 78. I fattori primi di un numero tra 2 e $2^{20} = 1,048,576$ sono al più 7.



2 Ordinamento

Le prestazioni di *QuickSort*, un algoritmo per l'ordinamento di una sequenza S di n numeri, dipendono dall'ordinamento iniziale di S: per l'input peggiore, ovvero quello che porta ai partizionamenti più sfavorevoli e che in generale dipende dalla versione deterministica adottata di *QuickSort*, il numero di confronti effettuati è sempre $O(n^2)$. **Vediamo un algoritmo randomizzato che si tiene lontano dal caso peggiore con altissima probabilità e il cui numero atteso di confronti è** $O(n \log n)$. Riprendiamo dapprima un risultato della Teoria della Probabilità (da pagina 166 del *Ross* [3]) di importanza fondamentale.

Linearità del valore atteso

Il valore atteso della somma di n variabili casuali X_1, X_2, ..., X_n, non necessariamente indipendenti, è uguale alla somma dei valori attesi delle singole variabili

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}\left[X_i\right] \tag{1}$$

Nell'uguaglianza (1) il valore atteso a sinistra è calcolato rispetto alla distribuzione di probabilità *congiunta* delle n variabili casuali X_1, X_2, ..., X_n. A destra, invece, il valore atteso di ognuna delle n variabili casuali è calcolato rispetto alla sua distribuzione di probabilità *marginale* e indipendentemente dai valori assunti dalle altre $n-1$ variabili casuali. Per semplicità supponiamo che tutte le variabili $X_1, \ldots, X_n$ assumano valori in un insieme discreto $\mathcal{X}$. Se indichiamo con $x_i \in \mathcal{X}$ un possibile valore assunto da X_i, con $p(x_1, \ldots, x_n)$ la probabilità congiunta per cui $X_1 = x_1$, $X_2 = x_2, \ldots, X_n = x_n$ e con $p_{X_i}(x)$ la probabilità marginale per cui $X_i = x_i$ per $i = 1, \ldots, n$ l'uguaglianza (1) si riscrive

$$\sum_{x_1 \in \mathcal{X}} \cdots \sum_{x_n \in \mathcal{X}} p(x_1, \ldots, x_n)(x_1 + \cdots + x_n) = \sum_{x_1 \in \mathcal{X}} p_{X_1}(x_1)x_1 + \cdots + \sum_{x_n \in \mathcal{X}} p_{X_n}(x_n)x_n \tag{2}$$

Verifichiamo l'uguaglianza (2) nel caso di due sole variabili casuali discrete X_1 e X_2. Scriviamo $p(x_1, x_2)$ per la funzione di probabilità di massa *congiunta* e $p_{X_1}(x_1)$ e $p_{X_2}(x_2)$ per le *marginali*. Abbiamo che

$$\sum_{x_1 \in \mathcal{X}} p(x_1, x_2) = p_{X_2}(x_2) \text{ e } \sum_{x_2 \in \mathcal{X}} p(x_1, x_2) = p_{X_1}(x_1) \tag{3}$$

Sappiamo che se X_1 e X_2 non sono indipendenti $p(x_1, x_2) \neq p_{X_1}(x_1)p_{X_2}(x_2)$. Usando le uguaglianze (3) otteniamo

$$\begin{aligned}
\mathbb{E}[X_1 + X_2] &= \sum_{x_1}\sum_{x_2}(x_1 + x_2)p(x_1, x_2) = \sum_{x_1}\sum_{x_2} x_1 p(x_1, x_2) + \sum_{x_1}\sum_{x_2} x_2 p(x_1, x_2) \\
&= \sum_{x_1} x_1 \left(\sum_{x_2} p(x_1, x_2)\right) + \sum_{x_2} x_2 \left(\sum_{x_1} p(x_1, x_2)\right) \\
&= \sum_{x_1} x_1 p_{X_1}(x_1) + \sum_{x_2} x_2 p_{X_2}(x_2) = \mathbb{E}[X_1] + \mathbb{E}[X_2]
\end{aligned}$$

Problema 2.1. *Il guardarobiere distratto*
A una festa n persone affidano il proprio cappello a un guardarobiere che ripone gli n cappelli alla rinfusa. Quante persone in media ritireranno il proprio cappello alla fine della festa?
Soluzione: introduciamo n variabili casuali I_i con $i = 1, \ldots, n$ note come *variabili indicatrici*. Ogni variabile indicatrice I_i assume il valore 1 se l'i-esima persona ha ritirato il proprio cappello e 0 altrimenti. Il valore atteso di I_i è sempre uguale a $p_{I_i}(1)$. Infatti per $i = 1, \ldots, n$ otteniamo

$$\mathbb{E}[I_i] = 1 \cdot p_{I_i}(1) + 0 \cdot p_{I_i}(0) = p_{I_i}(1)$$

Poiché la probabilità *marginale* che la persona i-esima ritiri il proprio cappello è pari a $1/n$ abbiamo $p_{I_i}(1) = 1/n$. Anche se le I_i non sono indipendenti, invece che calcolare la probabilità congiunta di

n variabili indicatrici, possiamo scrivere il valore atteso della loro somma utilizzando n distribuzioni di probabilità marginali! Pertanto otteniamo immediatamente

$$\mathbb{E}\left[\sum_{i=1}^{n} I_i\right] = \sum_{i=1}^{n} \mathbb{E}\left[I_i\right] = \sum_{i=1}^{n} p_{I_i}(1) = \sum_{i=1}^{n} \frac{1}{n} = 1$$

Esercizio 2.1. *Quattro amici*
Risolvi il **Problema 2.1** con $n = 4$ senza usare l'uguaglianza (1).
Soluzione: per calcolare la probabilità *congiunta* che j amici con $j = 0, 1, \ldots, 4$ ritirino il proprio cappello, dobbiamo considerare tutte le 4! = 24 configurazioni possibili. Riportiamo nella tabella i 24 casi con accanto a ciascuno il numero di amici che ritirano il proprio cappello. In ogni quadrupla il numero identifica il cappello, la sua posizione il proprietario e ogni cappello restituito al proprietario è indicato in grassetto.

(**1**,**2**,**3**,**4**) 4	(**1**,**2**,4,3) 2	(**1**,3,2,**4**) 2	(**1**,3,4,2) 1	(**1**,4,2,3) 1	(**1**,4,**3**,2) 2
(2,1,**3**,**4**) 2	(2,1,4,3) 0	(2,3,1,**4**) 1	(2,3,4,1) 0	(2,4,1,3) 0	(2,4,**3**,1) 1
(3,1,2,**4**) 1	(3,1,4,2) 0	(3,**2**,1,**4**) 2	(3,**2**,4,1) 1	(3,4,1,2) 0	(3,4,2,1) 0
(4,1,2,3) 0	(4,1,**3**,2) 1	(4,**2**,1,3) 1	(4,**2**,**3**,1) 2	(4,3,1,2) 0	(4,3,2,1) 0

Tenendo presente che ogni configurazione ha probabilità 1/24, dalla tabella risulta che

$$p(0) = \frac{9}{24},\ p(1) = \frac{8}{24},\ p(2) = \frac{6}{24},\ p(3) = 0 \text{ e } p(4) = \frac{1}{24}$$

Per il valore atteso degli amici che ritirano il proprio cappello, pertanto, otteniamo

$$\sum_{j=0}^{4} jp(j) = 0 \cdot \frac{9}{24} + 1 \cdot \frac{8}{24} + 2 \cdot \frac{6}{24} + 4 \cdot \frac{1}{24} = \frac{8 + 12 + 4}{24} = 1$$

Osservazione 2.1. *Lineare invece che fattoriale!*
Se non usiamo l'uguaglianza (1), quindi, siamo costretti a calcolare la probabilità *congiunta* con la quale $0, 1, \ldots, n$ amici ritirano il proprio cappello e dobbiamo quindi considerare $n!$ configurazioni. Grazie all'uguaglianza (1), invece, il valore atteso della somma è dato dalla somma dei valori attesi di n variabili casuali indicatrici calcolate a partire da n distribuzioni *marginali*.

Un algoritmo Las Vegas

Siamo ora in grado di fare la conoscenza di un primo tipo di algoritmi randomizzati.

Algoritmo 2.1. *LVQuickSort*(S)
Input: S, sequenza di numeri
Output: S, sequenza ordinata dal più piccolo al più grande

`if` $|S| \leq 1$
  `then return` S
`else`
  1. campiona s da S con probabilità uniforme
  2. forma $S_<$ e $S_\geq$ confrontando ogni numero di $S \setminus \{s\}$ con s
  3. *LVQuickSort*$(S_\geq)$
  4. *LVQuickSort*$(S_<)$
  5. `return` la concatenazione di $S_<$, s e $S_\geq$

LVQuickSort è un esempio di algoritmo randomizzato *Las Vegas*, ovvero di un algoritmo che **fornisce sempre l'output corretto anche se non è eseguito nello stesso tempo sullo stesso input**. Il tempo di esecuzione non dipende dall'input ma dal campionamento ed è quindi una variabile casuale.

Osservazione 2.2. *Quale algoritmo di ordinamento stiamo usando?*
A ogni chiamata *LVQuickSort* si comporta come una versione deterministica di *QuickSort* che usa un *pivot* in una certa posizione della sequenza. Il punto è che questa posizione non è sempre la stessa. Il caso peggiore per *LVQuickSort*, pertanto, costa sempre $O(n^2)$ ma, per n abbastanza grande, la probabilità che si realizzi è certamente trascurabile. □

Stimiamo allora il valore atteso del tempo di esecuzione di *LVQuickSort* inteso come numero atteso dei confronti effettuati (ancora dal *Ross* [3], pagina 306). **Per semplicità assumiamo che** S **sia costituita dai numeri naturali da** 1 **a** n. Introduciamo le variabili indicatrici $I(i,j)$ con $i < j$, con $i = 1, \ldots, n-1$ e $j = 2, \ldots, n$, definite come

$$I(i,j) = \begin{cases} 1 & \text{se } i \text{ e } j \text{ saranno confrontati} \\ 0 & \text{altrimenti} \end{cases}$$

Il numero di confronti effettuati è dato allora dalla variabile casuale

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} I(i,j)$$

Nel caso *peggiore* $I(i,j)$ vale 1 per tutti gli i e j e X è pari a $n(n-1)/2$. Infatti

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} \overbrace{1 + \cdots + 1}^{n-i} = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} = \frac{n(n-1)}{2}$$

Per la *proprietà della linearità del valore atteso* dell'uguaglianza (1) e tenuto conto del fatto che le $I(i,j)$ sono variabili indicatrici, se $p_{ij}(1)$ è la probabilità che i e j saranno confrontati, il valore atteso del numero di confronti X è

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} I(i,j)\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij}(1)$$

Valutiamo ora le $p_{ij}(1)$. **Inizialmente tutti i** $j-i+1$ **numeri da** i **a** j **sono nella stessa partizione**. A ogni passo, se il numero campionato è minore di i o maggiore di j, tutti i valori tra i e j non saranno confrontati tra loro e rimarranno nella stessa partizione. Non cambia nulla fino a che non è campionato un numero $i \leq k \leq j$. Se $k \neq i$ e $k \neq j$, i valori i e j finiscono in partizioni diverse e non saranno mai confrontati. Nel caso, invece, che $k = i$ o $k = j$, i e j sono confrontati. Con 2 casi favorevoli su $j - i + 1$ casi possibili, e con la tacita assunzione di equiprobabilità, si ha

$$p_{ij}(1) = \frac{2}{j-i+1}$$

e, quindi,

$$\sum_{j=i+1}^{n} \frac{2}{j-i+1} \approx \int_{i+1}^{n} \frac{2}{t-i+1} dt = 2\ln(n-i+1) - 2\ln(2) \approx 2\ln(n-i+1) \quad \text{da cui}$$

$$\begin{aligned} \mathbb{E}[X] &\approx \sum_{i=1}^{n-1} 2\ln(n-i+1) \approx \int_{1}^{n-1} 2\ln(n-t+1)dt = 2\int_{2}^{n} \ln y dy \\ &= 2\,(y\ln y - y)|_2^n = 2(n\ln n - n - 2 + 2) \approx 2n\ln n = O(n\log n) \end{aligned}$$

dove l'integrale è ottenuto con la sostituzione $y = n - t + 1$ e scambiando i nuovi estremi di integrazione per assorbire il cambio di segno tra t e y.

Osservazione 2.3. *Tutto qui?*
Il fatto che il costo computazionale atteso di *LVQuickSort*, $O(n\log n)$, sia lo stesso del costo computazionale medio di *QuickSort* non deve trarre in inganno. La randomizzazione ci protegge dalla malizia della distribuzione sugli input. Comunque siano distribuiti gli input, il costo computazionale atteso di *LVQuickSort* è sempre $O(n\log n)$. Nel caso deterministico questo non è vero. Dire che il costo computazionale medio di *QuickSort* è $O(n\log n)$ presuppone di aver fissato una distribuzione di equiprobabilità sugli input. Se le sequenze da ordinare, per esempio, fossero in partenza molto spesso *quasi* ordinate, il costo computazionale medio di *QuickSort* si avvicinerebbe a $O(n^2)$.

Grandi scostamenti dal valore atteso

Usiamo due disuguaglianze classiche della Teoria della Probabilità (vedi il *Ross* [3] a pagina 388) per dare una stima quantitativa del fatto che il tempo di esecuzione di *LVQuickSort* non può discostarsi molto dal suo valore atteso.

Supponiamo di conoscere il valore atteso di una variabile casuale $X \geq 0, \mathbb{E}[X] = \mu$. Allora vale la *disuguaglianza di Markov*, ovvero

$$\forall a > 0, \Pr\{X \geq a\} \leq \frac{\mu}{a} \tag{4}$$

Se supponiamo di conoscere anche la varianza di X, $Var(X) = \sigma^2$, applicando la disuguaglianza di *Markov* a $(X - \mu)^2$ con $a = \epsilon^2$ otteniamo la *disuguaglianza di Chebyshev*, ovvero

$$\forall \epsilon > 0, \Pr\{|X - \mu| \geq \epsilon\} \leq \frac{\sigma^2}{\epsilon^2} \tag{5}$$

Sia v un intero piccolo (come 2 o 3). Dalla disuguaglianza (4) con $a = v\mu$ otteniamo

$$\Pr\{X \geq v\mu\} \leq \frac{\mu}{v\mu} = \frac{1}{v} \tag{6}$$

Dalla disuguaglianza di *Chebyshev* (5) con $\epsilon = (v - 1)\mu$, invece, otteniamo

$$\Pr\{X \geq v\mu\} \leq \frac{\sigma^2}{(v-1)^2\mu^2} \tag{7}$$

Compito 2.1. *Implementazione di LVQuickSort*
Costruisci una sequenza S di numeri con $|S| = 10^4$. Implementa *LVQuickSort* e conta il numero X_r di confronti effettuati in ogni singolo *run* r per ordinare la sequenza S. Calcola il valore medio $\hat{\mu}$ e la deviazione standard empirica $\hat{\sigma}$ del numero di confronti effettuati su $R = 10^5$ *run* usando le formule

$$\hat{\mu} = \frac{1}{R}\sum_{r=1}^{R} X_r \quad \text{e} \quad \hat{\sigma}^2 = \frac{1}{R-1}\sum_{r=1}^{R}(X_r - \hat{\mu})^2$$

Produci un istogramma di 50 bin con i valori ottenuti. Limita dall'alto la probabilità con la quale *LVQuickSort* effettua il doppio e il triplo del valore atteso dei confronti mediante le disuguaglianze (6) e (7) con $\mu = \hat{\mu}$ e $\sigma = \hat{\sigma}$. Confronta la frequenza empirica con la quale hai ottenuto il doppio e il triplo di $\hat{\mu}$ con i limiti delle disuguaglianze (6) e (7). **Commenta i risultati che ottieni**.

3 Programmazione lineare

In molti ambiti si incontrano problemi di programmazione lineare, ovvero problemi di ottimizzazione in cui sia la funzione obiettivo sia tutti i vincoli sono lineari. Dopo aver definito un problema di programmazione lineare partendo da semplice esempio, ci restringiamo al caso in cui la funzione obiettivo dipende da poche variabili e vediamo **un algoritmo Las Vegas capace di gestire efficientemente il caso in cui il numero dei vincoli è molto grande**.

Funzione obiettivo e vincoli lineari

Un semplice esempio

Un'impresa produce due beni, X_1 e X_2, con due macchinari, A e B. La produzione di un'unità del bene X_1 impegna A per 40 e B per 20 minuti, quella di un'unità del bene X_2 impegna A per 20 e B per 60 minuti. All'inizio della settimana ci sono 25 unità del bene X_1 e 90 unità del bene X_2 in magazzino e la pianificazione prevede di avere accesso ad A per 40 ore e a B per 30 e di dover soddisfare una domanda di 75 unità del bene X_1 e 95 del bene X_2. L'impresa vuole ottimizzare la somma delle unità dei beni X_1 e X_2 prodotte alla fine della settimana. Se x_1 e x_2 misurano le quantità di beni X_1 e X_2 prodotti, i vincoli sulla disponibilità temporale dei macchinari espressa in minuti sono descritti dalle disequazioni lineari

$$40x_1 + 20x_2 \leq 2400 \quad \text{per } A \text{ e } 20x_1 + 60x_2 \leq 1800 \quad \text{per } B$$

In Figura 3 questi vincoli sono rappresentati dai semipiani al di sotto delle due rette oblique. I due vincoli per i minimi quantitativi di produzione pianificati, al netto delle riserve di magazzino, sono espressi dalle disequazioni lineari

$$x_1 \geq 75 - 25 = 50 \quad \text{e} \quad x_2 \geq 95 - 90 = 5$$

e, sempre in Figura 3, sono rappresentati dai semipiani a destra della retta verticale tratteggiata e sopra la retta orizzontale tratteggiata. La funzione obiettivo $\mathcal{O}$ da massimizzare è la somma dei due quantitativi, ovvero $\mathcal{O}(x_1, x_2) = x_1 + x_2$. La soluzione grafica di questo problema è immediata. La *regione ammissibile* è il quadrilatero, i cui vertici sono colorati, intersezione dei quattro semipiani. La soluzione ottima si trova nel vertice rosso di coordinate (54, 12) che giace sulla retta tratteggiata in grassetto di equazione $x_1 + x_2 = 66$.

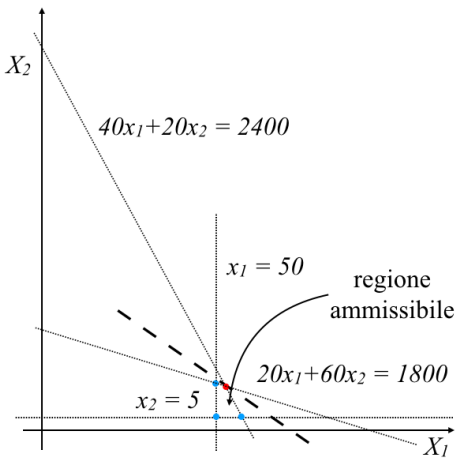


Figura 3: Vedi testo.

Osservazione 3.1. *L'ottimo non può essere all'interno della regione ammissibile*
Partendo da un qualunque punto interno (x_1, x_2) della regione ammissibile possiamo spostarci a destra lungo X_1 o in alto lungo X_2 di un passo $\delta > 0$ sufficientemente piccolo da rimanere all'interno

della regione. In entrambi i casi avremo che

$$\mathcal{O}(x_1 + \delta, x_2) = \mathcal{O}(x_1, x_2 + \delta) = x_1 + x_2 + \delta = \mathcal{O}(x_1, x_2) + \delta > \mathcal{O}(x_1, x_2)$$

Formulazione generale

Un problema di programmazione **lineare** con n variabili ed m vincoli si presenta nella forma

$$\begin{array}{ll} \max & \mathbf{c}^\top \mathbf{x} \\ \text{soggetto a} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{array}$$

dove $\mathbf{c}$ e $\mathbf{x} \in \mathbb{R}^n$ (e la funzione obiettivo è quindi **lineare** nelle n variabili), $\mathbf{b} \in \mathbb{R}^m$ e $\mathbf{A} \in \mathbb{R}^m \times \mathbb{R}^n$. Per semplicità assumiamo inoltre che

A1 : la regione ammissibile non è vuota

A2 : l'ottimo è unico

A3 : ogni vertice della regione ammissibile è definito da esattamente d vincoli

A4 : tutti i vincoli sono **lineari** e linearmente indipendenti

A5 : non sussistono parallelismi accidentali

Metodo del simplesso

Il metodo del **simplesso**, individuato un *vertice* $\mathbf{x}^0$ del poliedro che definisce la regione ammissibile, si sposta su un vertice adiacente nel quale la funzione obiettivo aumenta strettamente. Due i casi possibili: o l'algoritmo *diverge*, perchè la funzione obiettivo cresce indefinitamente in una direzione in cui la regione ammissibile non è limitata, o *si arresta*, perché in nessuno dei vertici adiacenti il valore della funzione obiettivo aumenta. Supponiamo che, nell'esempio di Figura 3, il vertice di partenza sia l'intersezione delle due rette parallele agli assi, ovvero il vertice azzurro $\mathbf{x}^0 = (50, 5)$ dove $\mathcal{O}(\mathbf{x}^0) = 55$. L'algoritmo del simplesso si sposta quindi in direzione verticale sul vertice azzurro adiacente $\mathbf{x}^1 = (50, 13.\bar{3})$, detto *pivot*, intersezione della retta verticale con la retta di equazione $20x_1 + 60x_2 = 1800$ e in cui $\mathcal{O}(\mathbf{x}^1) = 63.\bar{3}$. L'algoritmo, infine, si arresta nel vertice rosso $\mathbf{x}^2 = (54, 12)$, nuovo *pivot*, punto di intersezione delle due rette oblique e in cui $\mathcal{O}(\mathbf{x}^2) = 66$.

Il vertice $\mathbf{x}^0$ da cui partire può essere trovato mediante l'algoritmo del simplesso applicato a una trasformazione del problema originale in una forma *standard* in cui l'inizializzazione è banale e la funzione obiettivo è negativa. Se l'algoritmo restituisce una soluzione negativa, il problema originale non ammette soluzione. Se restituisce un valore nullo, la soluzione ottenuta è un vertice della regione ammissibile del problema originale che può essere usato come inizializzazione.

Osservazione 3.2. *Costo computazionale esponenziale*
Empiricamente, l'algoritmo del simplesso è spesso efficiente. Sono noti tuttavia problemi per i quali la convergenza è raggiunta esplorando metà dei vertici del poliedro che definisce la regione ammissibile. Poiché il numero di vertici di un poliedro cresce esponenzialmente con le dimensioni del problema, per l'input peggiore il costo dell'algoritmo del simplesso è esponenziale.

Base dell'insieme dei vincoli

Nel seguito indichiamo con V l'insieme dei vincoli e con $\mathcal{O}$ la funzione obiettivo. Sia $\mathcal{O}_V$ l'ottimo della funzione obiettivo, raggiunto massimizzando la funzione obiettivo e soddisfacendo tutti i vincoli in V. Se $\mathcal{O}_{V'}$ è l'ottimo della stessa funzione obiettivo sul problema ristretto ai vincoli in $V' \subseteq V$, poiché non tutti i vincoli in V sono necessariamente in V', avremo

$$\mathcal{O}_{V'} \geq \mathcal{O}_V$$

Un vincolo $v \in V$ è *critico* o *estremo* se

$$\mathcal{O}_{V \setminus \{v\}} > \mathcal{O}_V$$

ovvero se l'eliminazione di v dall'insieme V dei vincoli consente all'ottimo della funzione obiettivo di aumentare di una quantità strettamente positiva. Un sottoinsieme di vincoli è una *base* se tutti i suoi vincoli sono critici. Se V è l'insieme dei vincoli, la *base* di V è il sottoinsieme minimale $\mathcal{B}(V) \subseteq V$ per cui $\mathcal{O}_{\mathcal{B}} = \mathcal{O}_V$. Per l'ipotesi sul numero di vincoli che definiscono un vertice, abbiamo che $|\mathcal{B}(V)| = n$. Nell'esempio di Figura 3, $n = 2$ e la base $\mathcal{B}(V)$ è costituita dai vincoli $40x_1 + 20x_2 \leq 2400$ e $20x_1 + 60x_2 \leq 1800$.

Vediamo ora come, nel caso in cui il numero di variabili n non è troppo grande, è possibile utilizzare un algoritmo *Las Vegas* (Seidel [4]) che si tiene lontano dal caso peggiore.

Campionamento casuale dei vincoli

Sia V l'insieme degli m vincoli di un problema di programmazione lineare con n variabili e funzione obiettivo $\mathcal{O}$. Se $m \gg n$ sappiamo che la maggior parte dei vincoli sono ridondanti. L'algoritmo *LVIncrementalLP* campiona uniformemente un vincolo v nell'insieme V e chiama ricorsivamente se stesso sull'insieme di vincoli $V \setminus \{v\}$. Se il vincolo campionato è ridondante, *LVIncrementalLP* restituisce l'ottimo ottenuto. Altrimenti, *LVIncrementalLP* proietta tutti i vincoli in $V \setminus \{v\}$ su v e chiama ricorsivamente se stesso su un problema di programmazione lineare di dimensione $n - 1$ su $m - 1$ vincoli, vedi Figura 4, poiché l'ottimo deve trovarsi sulla faccia delimitata da v.

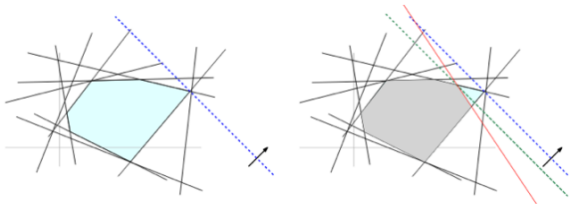


Figura 4: A sinistra: la regione ammissibile, in azzurro chiaro, ottenuta dopo aver campionato (e quindi escluso) il vincolo corrispondente alla retta rossa. L'ottimo è il vertice in cui la retta tratteggiata blu tocca la regione ammissibile. A destra: la nuova regione ammissibile, in grigio, ottenuta dopo aver proiettato tutti i vincoli sulla retta rossa che viola l'ottimo del passo precedente. Il nuovo ottimo è il vertice in cui la retta tratteggiata verde tocca la nuova regione ammissibile.

Algoritmo 3.1. *LVIncrementalLP*(V)
Input: un insieme V di $|V|$ vincoli
Output: $\mathbf{x}^*$, ottimo per V, e $\mathcal{B}(V)$, base di V

---

1. `if` $n = 1$ or $m = 1$
   determina $\mathbf{x}^*$
   `return` $\mathbf{x}^*$
2. campiona un vincolo v uniformemente in V
3. *LVIncrementalLP*$(V \setminus \{v\})$
4. `if` $\mathbf{x}^*$ non viola v
   `return` $\mathbf{x}^*$
   `else`
   proietta i vincoli in $V \setminus \{v\}$ su v ottenendo V'
   LVIncrementalLP(V')

Osservazione 3.3. *Complessità di LVIncrementalLP*
Se $T(m, n)$ è il valore atteso del tempo di esecuzione dell'algoritmo per m vincoli in n dimensioni

si ha

$$T(m,n) \leq T(m-1,n) + O(n) + \frac{n}{m}\left(O(nm) + T(m-1,n-1)\right)$$

dove $T(m-1,n)$ è il costo della ricorsione su $V \setminus \{v\}$ mentre $O(n)$ quello del controllo della violazione di v da parte dell'ottimo ottenuto. La frazione n/m esprime la probabilità che v sia estremo, $O(nm)$ il costo delle proiezioni dei vincoli in $V \setminus \{v\}$ su v e $T(m-1,n-1)$ quello della ricorsione su un problema che ha una dimensione e un vincolo in meno. Verifichiamo, con il metodo di sostituzione, che per b abbastanza grande

$$T(m,n) \leq bmn!$$

Infatti

$$\begin{aligned} T(m,n) &\leq b(m-1)n! + O(n) + \frac{n}{m}\left(O(nm) + b(m-1)(n-1)!\right) \\ &= b(m-1)n! + O(n) + \frac{n}{m}O(nm) + bn! - \frac{1}{m}bn! \\ &\leq bmn! + O(n) + \frac{n}{m}O(nm) \leq bmn! \end{aligned}$$

La dipendenza dal fattoriale del numero delle dimensioni confina l'utilizzo di questo algoritmo ai casi in cui la funzione obiettivo dipende da poche variabili

Compito 3.1. *Campionamento manuale*
Nel primo quadrante del piano cartesiano $X \times Y$ è dato il problema di programmazione lineare

$$\begin{array}{ll} \max & 2y - x \\ \text{soggetto a} & v_1: \ y - x \leq 0 \\ & v_2: \ y - 1 \leq 0 \\ & v_3: \ y + x - 4 \leq 0 \end{array}$$

Disegna la regione ammissibile e il fascio improprio di rette parallele definito dalla funzione obiettivo. Risolvi il problema individuando la coppia di vincoli critici, il punto di massimo e la retta del fascio che lo contiene. Determina passo per passo come *LVIncrementalLP* risolve il problema se campioni

- v_1 da $\{v_1, v_2, v_3\}$ e v_2 da $\{v_2, v_3\}$ o
- v_2 da $\{v_1, v_2, v_3\}$ e v_1 da $\{v_1, v_3\}$

Ammetti come possibile il valore $+\infty$ nel caso in cui l'ottimo della funzione obiettivo non è limitato.

4 Teoria dei giochi

Introduciamo ora i concetti e i risultati della *Teoria dei Giochi* utili per analizzare le prestazioni degli algoritmi *Las Vegas*. Partiamo considerando un gioco nel quale i due avversari, a ogni turno, giocano contemporaneamente.

Teorema di Von Neumann

Roberta e Carlo giocano alla *morra cinese*. La strategia 1 (prima riga o colonna) corrisponde a giocare *carta*, la 2 *sasso* (seconda riga o colonna) e la 3 *forbice* (terza riga o colonna). Supponiamo che Roberta e Carlo si accordino di usare come matrice dei *payoff*

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & -2 \\ -1 & 0 & 3 \\ 2 & -3 & 0 \end{pmatrix}$$

Il *payoff* $\mathrm{M}_{ij} > 0$ è quanto Carlo, che ha giocato la strategia della colonna j, paga a Roberta, che ha giocato la strategia della riga i. Se $\mathrm{M}_{ij} < 0$ il pagamento è da Roberta a Carlo. Il gioco è a *somma zero* poiché la somma di quanto paga Carlo e quanto riceve Roberta, o viceversa, è sempre uguale a 0. Una **strategia pura** per uno dei due giocatori è quella di giocare sempre nello stesso modo (stessa riga o stessa colonna). Quale strategia pura conviene adottare a Roberta e quale a Carlo? Per entrambi, *in assenza di informazioni sulla strategia dell'avversario*, la strategia pura ottimale è scegliere **il migliore tra gli scenari peggiori**: per Roberta significa minimizzare le perdite, per Carlo massimizzare i guadagni.

Per **Roberta** il *payoff* ottimo corrisponde quindi al **massimo dei minimi per riga**, ovvero

$$V_R = \max_i \min_j \mathrm{M}_{ij} = \max\{-2, -1, -3\} = -1$$

La strategia ottimale di Roberta, pertanto, è giocare sempre *sasso*. Per **Carlo** il *payoff* ottimo corrisponde al **minimo dei massimi per colonna**, ovvero

$$V_C = \min_j \max_i \mathrm{M}_{ij} = \min\{2, 1, 3\} = 1$$

Anche la strategia pura ottimale di Carlo, pertanto, è giocare sempre *sasso*.

Poiché $V_R < V_C$, questo gioco non ha *soluzione*. Se $V_R = V_C = V$, invece, il gioco ha soluzione e le strategie per ottenere V sono ottimali.

Osservazione 4.1. *Punto di sella*
Nel caso della matrice dei pagamenti

$$\mathbf{N} = \begin{pmatrix} 1 & -3 & -2 \\ 2 & 5 & 4 \\ 2 & 3 & 2 \end{pmatrix}$$

N_{21} è un punto di sella, ovvero $N_{21} = 2$ è un minimo per la riga 2 e un massimo per la colonna 1. Anche N_{31} è un punto di sella, poiché $N_{31} = 2$ è un minimo per la riga 3 e un massimo per la colonna 1. In entrambi i casi, quindi, $V_R = V_C = V = 2$ e il gioco ha soluzione. La posizione, non il valore, è importante: $N_{33} = 2$ ma N_{33} **non è** un punto di sella!

Fatto 4.1. *Disuguaglianza generale*
Per qualunque matrice $\mathbf{M}$ dei pagamenti

$$V_R \leq V_C \quad \text{ovvero} \quad \max_i \min_j \mathrm{M}_{ij} \leq \min_j \max_i \mathrm{M}_{ij}$$

Dimostrazione: tutti gli elementi della riga i, M_{ik} per ogni k, sono certamente maggiori o uguali del minimo valore di $\mathbf{M}$ sulla riga i, ovvero

$$\forall i, k \quad \min_j \mathrm{M}_{ij} \leq \mathrm{M}_{ik}$$

Se la disuguaglianza vale per ogni riga i, allora vale certamente per il massimo su i, ovvero

$$\max_i \min_j \mathrm{M}_{ij} \leq \max_i \mathrm{M}_{ik} \quad \forall k \tag{8}$$

Poiché il lato sinistro non dipende da k, la disuguaglianza (8) vale anche per il minimo valore sul lato destro, ovvero

$$\max_i \min_j \mathrm{M}_{ij} \leq \min_j \max_i \mathrm{M}_{ij}$$

■

Supponiamo ora che Roberta e Carlo scelgano quale strategie adottare casualmente. Siano

$$\mathbf{p} = (p_1\ p_2\ p_3)^\top \ \text{ e } \ \mathbf{q} = (q_1\ q_2\ q_3)^\top, \ \text{ con } \ 0 \leq p_i \leq 1, 0 \leq q_i \leq 1 \ \text{ e} \sum_i p_i = \sum_i q_i = 1$$

le probabilità con le quali Roberta e Carlo scelgono quale riga e quale colonna giocare. Un risultato centrale della Teoria dei Giochi sancisce l'esistenza di strategie miste ottimali per un gioco a somma zero, $\mathbf{p}^*$ e $\mathbf{q}^*$, tale da garantire valore al gioco sottostante.

Teorema 4.1. *Minimax (Von Neumann)*
Per qualunque gioco a somma zero con matrice dei pagamenti $\mathbf{M}$ esistono strategie miste $\hat{\mathbf{p}}$ e $\hat{\mathbf{q}}$ per le quali il gioco ha valore V, ovvero,

$$\max_{\mathbf{p}} \min_{\mathbf{q}} \mathbf{p}^\top \mathbf{M}\mathbf{q} = \min_{\mathbf{q}} \max_{\mathbf{p}} \mathbf{p}^\top \mathbf{M}\mathbf{q} = \hat{\mathbf{p}}^\top \mathbf{M}\hat{\mathbf{q}} = V$$

La dimostrazione del **Teorema 4.1** esula dai nostri scopi.

Esercizio 4.1. *Strategie miste ottimali per la morra cinese*
Verifica che per la matrice $\mathbf{M}$ della morra cinese le strategie miste ottimali sono

$$\hat{\mathbf{p}} = \hat{\mathbf{q}} = (1/2\ \ 1/3\ \ 1/6)^\top$$

e che il valore del gioco è nullo.
Soluzione
Sostituendo, abbiamo

$$\hat{\mathbf{p}}^\top \mathbf{M} = (1/2\ \ 1/3\ \ 1/6) \begin{pmatrix} 0 & 1 & -2 \\ -1 & 0 & 3 \\ 2 & -3 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

e

$$\mathbf{M}\hat{\mathbf{q}} = \begin{pmatrix} 0 & 1 & -2 \\ -1 & 0 & 3 \\ 2 & -3 & 0 \end{pmatrix} \begin{pmatrix} 1/2 \\ 1/3 \\ 1/6 \end{pmatrix} = (0\ \ 0\ \ 0)$$

Osservazione 4.2. *Il potere dell'informazione*
Se Roberta conosce la strategia adottata da Carlo, Roberta può massimizzare il proprio *payoff* per mezzo di un'opportuna strategia pura. Per esempio, se Roberta sa che Carlo gioca sempre *sasso*, vincerebbe ogni mano giocando sempre *carta* con un *payoff* pari a $V_R = 2$. Tuttavia, a Carlo conviene cambiare strategia se sa, o capisce, che Roberta gioca sempre *sasso*. Una strategia mista ottimale consente di ottenere il miglior risultato possibile in presenza di una qualunque possibile strategia adottata dall'avversario.

Osservazione 4.3. *Test di Turing rovesciato*
Al link `https://www.essentially.net/rsp/play.jsp` puoi giocare alla morra cinese nella versione in cui vincite e perdite sono unitarie e osservare quanto un programma possa sfruttare velocemente l'adozione di strategie non ottimali da parte di un essere umano! Il programma, infatti, invece della strategia mista ottimale adotta una strategia che sfrutta la nostra debolezza nel generare scelte veramente casuali.

Esercizio 4.2. *Somma zero non significa che il valore del gioco è nullo*
Data la matrice dei pagamenti

$$\mathbf{M} = \begin{pmatrix} 7 & 3 \\ 2 & 5 \end{pmatrix}$$

determina le strategie pure ottimali per Roberta e Carlo, il valore del gioco e le strategie miste ottimali.
Soluzione
Osserviamo preliminarmente che anche se Carlo perde sempre, in quanto tutti gli elementi della matrice $\mathbf{M}$ sono positivi, il gioco è a somma zero perché quello che guadagna Roberta è esattamente quello che perde Carlo. Per le strategie pure cerchiamo il migliore degli scenari peggiori per entrambi. Roberta, che seleziona il massimo dei minimi per riga, deve giocare sempre la prima riga ottenendo il *payoff*

$$V_R = \max\{3, 2\} = 3$$

Carlo, che invece seleziona il minimo dei massimi per colonna, deve giocare sempre la seconda colonna ottenendo il *payoff*

$$V_C = \min\{7, 5\} = 5$$

Se $\mathbf{p} = (p \quad 1-p)^\top$ è una strategia mista per Roberta e $\mathbf{q} = (q \quad 1-q)^\top$ una strategia mista per Carlo possiamo riscrivere il prodotto $\mathbf{p}^\top\mathbf{M}\mathbf{q}$ come

$$\begin{aligned}
\mathbf{p}^\top\mathbf{M}\mathbf{q} &= (p \quad 1-p)\begin{pmatrix} 7 & 3 \\ 2 & 5 \end{pmatrix}\begin{pmatrix} q \\ 1-q \end{pmatrix} \\
&= (5p+2 \quad 5-2p)^\top \begin{pmatrix} q \\ 1-q \end{pmatrix} \\
&= 5pq + 2q + 5 - 2p - 5q + 2pq = 7pq + 3q - 2p + 5 \\
&= 7\left(pq - \frac{3}{7}q - \frac{2}{7}p + \frac{35}{49}\right) = 7\left(pq - \frac{3}{7}q - \frac{2}{7}p + \frac{6}{49} + \frac{29}{49}\right) \\
&= 7\left(p - \frac{3}{7}\right)\left(q - \frac{2}{7}\right) + \frac{29}{7}
\end{aligned}$$

Pertanto, se Roberta adotta la strategia $\hat{\mathbf{p}} = (3/7 \quad 4/7)^\top$ ottiene $V_R = 29/7$. Similmente, se Carlo adotta la strategia $\hat{\mathbf{q}} = (2/7 \quad 5/7)^\top$ ottiene $V_C = 29/7$. Abbiamo quindi che il gioco ha valore

$$V = V_R = V_C = \frac{29}{7}$$

Ovviamente abbiamo che il valore del gioco è intermedio ai *payoff* delle strategie pure ottimali, in quanto

$$3 < \frac{29}{7} < 5$$

Compito 4.1. *Strategia vincente*
Determina la strategia che deve adottare Roberta per massimizzare il proprio *payoff* alla *morra cinese* se Carlo gioca una strategia mista del tipo $\mathbf{q} = (1/3 \;\; 1/3 \;\; 1/3)^\top$.
Soluzione
Con $\mathbf{q} = (1/3 \;\; 1/3 \;\; 1/3)^\top$ otteniamo

$$\mathbf{M}\mathbf{q} = \begin{pmatrix} 0 & 1 & -2 \\ -1 & 0 & 3 \\ 2 & -3 & 0 \end{pmatrix}\begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} = \begin{pmatrix} -1/3 \\ 2/3 \\ -1/3 \end{pmatrix}$$

È immediato allora concludere che Roberta massimizza il proprio payoff giocando la strategia pura *sasso* $\mathbf{p} = (0 \;\; 1 \;\; 0)^\top$. Infatti, Roberta perderebbe 1 contro *carta* un terzo delle volte, pareggerebbe un terzo delle volte contro *sasso* e vincerebbe 3 contro *forbice* un terzo delle volte, ovvero

$$V_R = -1 \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} + 3 \cdot \frac{1}{3} = \frac{2}{3}$$

Lo stesso valore atteso si ottiene anche direttamente dal prodotto

$$(0 \;\; 1 \;\; 0) \begin{pmatrix} -1/3 \\ 2/3 \\ -1/3 \end{pmatrix} = \frac{2}{3}$$

Conoscendo la strategia adottata da Carlo, Roberta massimizza il proprio ritorno adottando una strategia pura appropriata. La strategia mista ottimale è quella da adottare quando non si conosce la strategia dell'avversario, o quando si presume di giocare contro un avversario che conosce i rudimenti della *Teoria dei giochi*!

Algoritmi deterministici e randomizzati

Consideriamo ora un gioco nel quale ogni elemento della matrice dei pagamenti è il tempo di esecuzione $\mathbf{T}$ di un dato algoritmo deterministico A su un certo input I. Le strategie pure di Carlo, le colonne, corrispondono a tutti i possibili algoritmi deterministici corretti che risolvono un problema dato mentre quelle di Roberta, le righe, a tutti gli input accettati di lunghezza fissata. Il valore V_C, che si ottiene con la strategia pura in cui Carlo sceglie *l'algoritmo deterministico con il tempo di esecuzione migliore sull'input peggiore*, è la **complessità deterministica** del problema, ovvero

$$V_C = \min_{A \in \mathcal{A}} \max_{I \in \mathcal{I}} \mathbf{T}(I, A)$$

Le interpretazioni più interessanti sono quelle che derivano dall'adozione di strategie miste. Indichiamo con A_q un algoritmo randomizzato *Las Vegas* campionato da una distribuzione q su $\mathcal{A}$ e con I_p un input campionato da una distribuzione p su $\mathcal{I}$.

Teorema 4.2. *Principio di Yao*
Per tutte le distribuzioni p e q,

$$\min_{A \in \mathcal{A}} \mathbb{E}[\mathbf{T}(I_p, A)] \leq \max_{I \in \mathcal{I}} \mathbb{E}[\mathbf{T}(I, A_q)] \tag{9}$$

Pertanto, fissata una qualunque distribuzione p degli input, il tempo di esecuzione atteso del miglior algoritmo deterministico è un limite inferiore per il tempo di esecuzione atteso di un qualunque algoritmo randomizzato.

Dimostrazione: scrivendo il **Teorema 4.1** come

$$\min_{q'} \max_{p'} \mathbb{E}[\mathbf{T}(I_{p'}, A_{q'})] = \max_{p'} \min_{q'} \mathbb{E}[\mathbf{T}(I_{p'}, A_{q'})]$$

otteniamo

$$\begin{aligned} \min_{A \in \mathcal{A}} \mathbb{E}[\mathbf{T}(I_p, A)] &\leq \max_{p'} \min_{A \in \mathcal{A}} \mathbb{E}[\mathbf{T}(I_{p'}, A)] \\ &\leq \max_{p'} \min_{q'} \mathbb{E}[\mathbf{T}(I_{p'}, A_{q'})] = \min_{q'} \max_{p'} \mathbb{E}[\mathbf{T}(I_{p'}, A_{q'})] \\ &\leq \min_{q'} \max_{I \in \mathcal{I}} \mathbb{E}[\mathbf{T}(I, A_{q'})] \leq \max_{I \in \mathcal{I}} \mathbb{E}[\mathbf{T}(I, A_q)] \end{aligned}$$

■

Osservazione 4.4. *Interpretazione*
Per ottenere un limite inferiore al tempo di esecuzione atteso di un algoritmo randomizzato possiamo scegliere la distribuzione degli input p più conveniente e studiare la complessità dell'algoritmo deterministico ottimo su p.

Osservazione 4.5. *Un caso fortunato*
Per *LVQuickSort* abbiamo potuto calcolare il costo computazionale atteso direttamente e abbiamo trovato che coincide con il costo computazionale di *QuickSort* sull'input migliore. Dalla disuguaglianza (9) ricaviamo che deve esistere una distribuzione di probabilità sugli input per cui il costo computazionale atteso di *QuickSort* è $O(n \log n)$: distribuzione rispetto alla quale tutti gli $n!$ ordinamenti iniziali possibili sono equiprobabili.

5 Valutazione dell'albero di un gioco

Costruiamo ora l'albero di un gioco e vediamo come prendere decisioni basate sul caso fornisca nuovamente un algoritmo *Las Vegas* capace di tenersi lontano dal caso peggiore.

Costruzione e valutazione deterministica

Consideriamo un gioco in cui due giocatori si confrontano effettuando una mossa a turno. Entrambi i giocatori sono razionali e fanno del loro meglio per vincere. Se un giocatore vince, l'altro perde. Restringiamo quindi la nostra analisi a giochi in cui non c'è il pareggio. Costruiamo un albero i cui nodi rappresentano tutte le possibili configurazioni del gioco e i cui archi le possibili mosse che portano da una configurazione a un'altra. Per semplicità restringiamo la nostra attenzione a un gioco nel quale, in ogni configurazione, ognuno dei due giocatori può scegliere una tra d mosse:

- un *albero del gioco*, $T_{d,k}$, è un albero uniforme i cui tutti i nodi interni hanno d figli che corrispondono alle mosse eseguibili dalla configurazione padre
- $T_{d,k}$ ha d^{2k} foglie che si trovano a distanza $2k$ dalla radice; nel caso binario $2^{2k} = 4^k$
- i nodi interni di $T_{d,k}$ a distanza pari (dispari) dalla radice sono di tipo MIN (MAX)
- il *valore del gioco* è un numero reale associato a ogni foglia di $T_{d,k}$
- il **problema della valutazione del gioco** consiste nel determinare il valore restituito dalla radice tenendo conto che
  1. ogni foglia restituisce il proprio valore
  2. ogni nodo restituisce il valore massimo (minimo) dei suoi d figli, se di tipo MAX (MIN)
- se i valori possibili di ogni foglia sono 0 e 1, il valore di un nodo di tipo MAX è dato dall'OR del valore dei figli, mentre il valore di un nodo di tipo MIN dall'AND, vedi Figura (5)
- il giocatore che muove dai nodi di tipo MAX cerca di massimizzare il valore del gioco, il giocatore che muove dai nodi di tipo MIN di minimizzarlo.

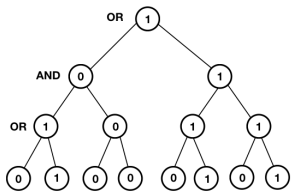


Figura 5: Albero di un gioco valutato in ogni nodo.

Nel semplice gioco in Figura 5 la radice è di tipo OR. Il giocatore che muove per primo potrà vincere, ovvero arrivare a una foglia di valore 1, se avrà a disposizione **almeno una mossa verso un nodo di valore 1**. Un percorso vincente dalla radice a una foglia deve quindi necessariamente passare per nodi AND per i quali, **indipendentemente dalla mossa scelta dall'altro giocatore**, il valore del gioco rimane 1. Nel caso di Figura 5 i due percorsi vincenti corrispondono ai cammini dalla radice alle foglie di valore 1 nel sottoalbero di destra. Consideriamo una procedura deterministica ricorsiva che legge l'albero da sinistra a destra e verifichiamo, su questo semplice esempio, che la procedura deve necessariamente leggere tutte le foglie prima di poter determinare l'esistenza di strategie vincenti.

Partiamo ricorsivamente dalla radice: per valutare il valore del nodo AND di sinistra, la procedura comincia valutando l'OR di sinistra. La procedura può attribuire il valore 1 all'OR solo dopo aver letto il valore 1 della seconda foglia poiché il valore 0 della prima foglia non consente di determinare con certezza il valore dell'OR. Risalendo nella ricorsione la procedura, avendo valutato a 1 l'OR di sinistra, per determinare il valore del nodo AND deve necessariamente determinare anche il valore del secondo OR. Il valore 0 del secondo OR è ottenuto dalla procedura solo dopo aver letto il valore 0 di entrambe le foglie. Risalendo nella ricorsione la procedura, dopo aver letto tutti i nodi del

sottoalbero di sinistra, può finalmente attribuire il valore 0 al nodo AND. Per via di questo risultato, tuttavia, il valore della radice non può essere determinato prima di visitare il sottoalbero di destra. L'unico percorso vincente, quello che porta dalla radice alla foglia all'estrema destra, è determinato solo dopo aver letto anche l'ultima foglia. In generale, qualunque algoritmo deterministico potrebbe essere costretto a leggere tutte le foglie prima di valutare un gioco.

La forza del caso

Consideriamo ora il comportamento di un algoritmo randomizzato.

Teorema 5.1. *Valutazione nel caso binario*
Il costo atteso per valutare una qualsiasi realizzazione $T_{2,k}$ è al più 3^k.

Dimostrazione (per induzione):

Caso base, $k = 1$**:** consideriamo solo un nodo radice di tipo AND (vedi Figura 6). Per un nodo radice di tipo OR basta scambiare gli 1 con gli 0. Se la radice AND restituisce 1 (a sinistra

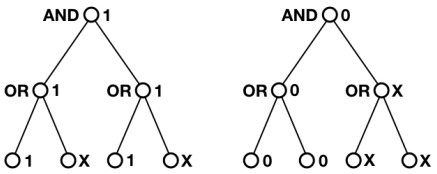


Figura 6: Vedi testo.

in Figura 6) entrambi i nodi figli di tipo OR devono restituire 1. Almeno una delle due foglie di entrambi i nodi, pertanto, vale 1 e in un caso su due sarà sufficiente la lettura di quella sola foglia, mentre nell'altro dovranno essere lette entrambe le foglie (nel caso più sfavorevole la prima foglia letta potrebbe valere 0). Per il valore atteso, tenuto conto che dobbiamo valutare entrambi gli OR, avremo allora

$$\frac{1}{2}(1+2)+\frac{1}{2}(1+2)=3$$

Se la radice AND restituisce 0 (a destra in Figura 6) il caso più sfavorevole si ha se un OR restituisce 1 con foglie di valore pari 0 e 1, mentre l'altro OR deve valere 0 e quindi con entrambe le foglie di valore 0. Nel caso più sfavorevole, pertanto, valuteremo metà delle volte solo l'OR di valore 0, 2 foglie in tutto, e metà delle volte

$$\frac{1}{2}\,(2+1)+2=\frac{7}{2}$$

foglie perché valutiamo per primo l'OR uguale a 1. Pertanto, il valore atteso delle foglie valutate è

$$\frac{2+7}{2}=\frac{9}{4}<3$$

Passo induttivo: assumiamo allora che il costo atteso per la valutazione di una realizzazione $T_{2,k-1}$ sia al più 3^{k-1}. Abbiamo quattro casi:

1. la radice, nodo OR, restituisce 1. I figli sono radici di sottoalberi di tipo $T_{2,k-1}$. Almeno uno dei figli deve restituire 1 e può essere scelto con probabilità $1/2$. In questo caso verrebbe letto uno solo dei sottoalberi, nell'altro caso entrambi. Per l'ipotesi induttiva avremo quindi
$$\frac{1}{2}\cdot 3^{k-1}+\frac{1}{2}\cdot 2\cdot 3^{k-1}=\frac{3^k}{2}\leq 3^k$$
2. la radice, nodo OR, restituisce 0. Entrambi i figli devono essere valutati per cui avremo
$$2\cdot 3^{k-1}\leq 3^k$$

3. la radice, nodo AND, restituisce 1. Entrambi i sottoalberi di tipo $T_{2,k-1}$ devono restituire 1 ed essere valutati per cui avremo, come nel caso 2, $2 \cdot 3^{k-1} \leq 3^k$
4. la radice, nodo AND, restituisce 0. Almeno uno dei figli deve restituire 0 e può essere scelto con probabilità $1/2$. In questo caso verrebbe letto uno solo dei sottoalberi, nell'altro caso entrambi. Per l'ipotesi induttiva avremo nuovamente $3^k/2 \leq 3^k$. ■

Osservazione 5.1. *Costo sublineare*
Se $n = 4^k$ e poiché $\log_4 3 \approx 0.793$, il tempo di esecuzione atteso è **inferiore** al caso peggiore di un algoritmo deterministico in quanto

$$3^k = 3^{\log_4 n} = 4^{\log_4 3^{\log_4 n}} = 4^{\log_4 n^{\log_4 3}} = n^{\log_4 3} = n^{0.793} < n$$

Limite inferiore

Valutiamo ora un limite inferiore al costo computazionale dell'algoritmo randomizzato appena visto usando il principio di Yao. Un algoritmo di tipo *Depth First Pruning* è un algoritmo deterministico che, nella valutazione di un albero T, non esplora ulteriormente sottoalberi di T che non portano informazione utile. Diamo per evidentemente vera la seguente proposizione:

> L'algoritmo deterministico che valuta T col numero atteso di passi minimale è di tipo *Depth First Pruning*.

Denotiamo il numero atteso di passi minimale di un albero binario T di profondità $\log_2 n$ con $W_T(\log_2 n)$. Per valutare $W_T(\log_2 n)$ consideriamo un albero costituito da nodi tutti di tipo NOR. Tabelle di verità alla mano è immediato verificare su un semplice esempio di un albero binario con quattro foglie che il valore della radice è lo stesso che otterremmo usando AND per la radice e OR per i due nodi interni, come in Figura 7.

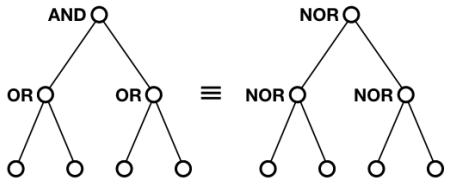


Figura 7: Vedi testo.

Fissiamo una distribuzione di probabilità per la quale le probabilità p e $1-p$ con cui una foglia assume i valori 1 e 0 sono indipendenti dai valori delle altre foglie e uguali a

$$p = \frac{3-\sqrt{5}}{2} \text{ e } 1-p = \frac{\sqrt{5}-1}{2}$$

Assumendo indipendenza tra i nodi, la probabilità che l'output di un nodo sia uguale a 1 è quella ottenuta moltiplicando le probabilità che entrambi gli input siano uguali a 0, ovvero $(1-p)^2$. Osserviamo che per $p = (3-\sqrt{5})/2$ abbiamo

$$(1-p)^2 = \left(\frac{\sqrt{5}-1}{2}\right)^2 = \frac{5+1-2\sqrt{5}}{4} = \frac{3-\sqrt{5}}{2} = p$$

Sia T un albero di tipo NOR di profondità $\log_2 n$ in cui le n foglie sono *indipendentemente* uguali a 1 con probabilità p. Per il numero atteso di passi necessari a valutare un nodo a distanza h possiamo scrivere

$$W_T(h) = W_T(h-1) + (1-p)W_T(h-1) = (2-p)W_T(h-1)$$

poiché il primo sottoalbero deve essere sempre valutato con un numero atteso di passi necessari uguali a $W_T(h-1)$, mentre il secondo sottoalbero è valutato solo nel caso in cui il primo abbia restituito 0 (evento che accade con probabilità uguale a $1-p$). Per l'intero albero, ovvero per $h = \log_2 n$, otteniamo

$$W_T(\log_2 n) = (2-p)^{\log_2 n} = \left(\frac{1+\sqrt{5}}{2}\right)^{\log_2 n} = 2^{\log_2 \left(\frac{1+\sqrt{5}}{2}\right)^{\log_2 n}} = n^{\log_2 \frac{1+\sqrt{5}}{2}} = n^{0.694} < n^{0.793}$$

Osservazione 5.2. *Che cosa manca?*
Il limite inferiore ottenuto sembra lasciare aperta l'esistenza di un algoritmo randomizzato migliore di quello discusso. In effetti non è vero: nel modello impiegato il valore dei nodi è scelto a caso e indipendentemente. Pertanto può capitare che entrambi gli input a un nodo NOR siano uguali a 1. Ma in questo caso nessun algoritmo gli leggerebbe entrambi. Un modello realistico, pertanto, deve tenere in conto che i valori dei nodi sono *dipendenti*.

6 Taglio minimo

Per un grafo connesso e non orientato G un taglio è una partizione dei suoi n vertici in due sottoinsiemi disgiunti non vuoti. Un taglio minimo è l'insieme di archi S di cardinalità minima la cui rimozione divide G in due o più componenti connesse. **Discutiamo un algoritmo Monte Carlo basato sulla rappresentatività di un campione casuale di una popolazione**.

Un algoritmo deterministico

Senza discuterne la correttezza, presentiamo dapprima un algoritmo deterministico [5] che comprime ripetutamente G in due vertici, s e t, ottenuti collassando tutti i vertici. *DeterministicMinCut*, vedi Figura 8, seleziona il vertice a e procede a costruire un grafo A includendo a ogni passo il vertice in G, marcato in giallo, collegato ad A con più archi. Arrivato a includere tutti i vertici meno gli ultimi 2, *DeterministicMinCut* termina la prima iterazione includendo il penultimo vertice (ridenominato s) e tagliando l'ultimo (ridenominato t). Dopo la prima iterazione il taglio, indicato dalla linea tratteggiata rossa, ha cardinalità 2 ed è salvato come taglio minimo. A questo punto *DeterministicMinCut* riparte *(i)* identificando i due ultimi vertici, e ed f in questo caso, *(ii)* aumentando di 1 il peso dell'arco che congiunge il nuovo vertice al grafo originale per compensare la scomparsa dell'arco di peso 1 tra i nodi d ed f, e *(iii)* ignorando l'elisione dell'arco che congiunge e ed f. Nella seconda iterazione produce ancora un taglio di cardinalità 2 per via del peso dell'arco che congiunge il vertice d al vertice ef. Alla terza iterazione questo arco è eliminato e *DeterministicMinCut* produce un nuovo taglio minimo di cardinalità 1. In Figura 8 sono mostrati anche i tagli ottenuti nelle ultime due iterazioni che ovviamente non modificano l'*output* dell'algoritmo.

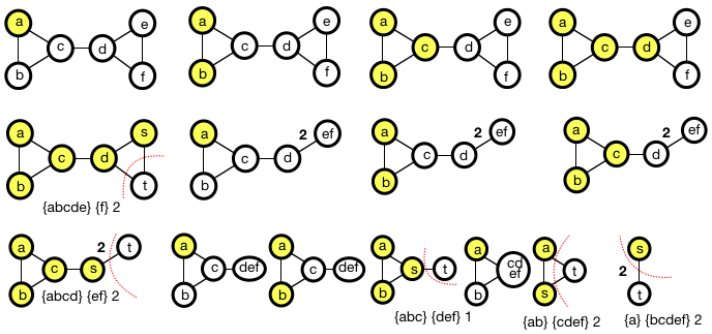


Figura 8: Vedi testo.

Algoritmo 6.1. *DeterministicMinCut*(G, a, S)
Input: G, un grafo connesso e non orientato con n vertici e a un vertice di G
Output: S, un insieme di archi di cardinalità minimale che taglia G in due o più componenti connesse

---

$currentMin = n$
`for` $i = 1$ `to` $n - 1$

1. $A \leftarrow \{a\}$
2. `while` $A \neq G$
   aggiungi ad A il vertice di G connesso ad A con più archi
3. salva S, taglio di A che lascia s e t (gli ultimi due vertici aggiunti) in componenti separate
4. `if` $|S| < currentMin$
   S taglio minimo
   $currentMin = |S|$
5. identifica i vertici s e t in G rimuovendo l'arco che li congiunge e aumentando il peso degli archi che partono dal nuovo vertice se necessario

---

Regola di moltiplicazione e distribuzione geometrica

Richiamiamo ora due risultati della Teoria della Probabilità (vedi il *Ross* [3] alle pagine 63 e 155).

Intersezione di eventi

Applicando ripetutamente la formula $\Pr(AB)/\Pr(A) = \Pr(B|A)$, la probabilità dell'intersezione di n eventi $E_1, E_2, \ldots, E_n$ si può scrivere come

$$\begin{aligned}\Pr(E_1E_2\ldots E_n) &= \frac{\Pr(E_1E_2\ldots E_n)}{\Pr(E_1E_2\ldots E_{n-1})}\frac{\Pr(E_1E_2\ldots E_{n-1})}{\Pr(E_1E_2\ldots E_{n-2})}\cdots\frac{\Pr(E_1E_2)}{\Pr(E_1)}\Pr(E_1) \\ &= \Pr(E_n|E_1E_2\ldots E_{n-1})\Pr(E_{n-1}|E_1E_2\ldots E_{n-2})\ldots\Pr(E_2|E_1)\Pr(E_1)\end{aligned} \tag{10}$$

Esercizio 6.1. *Tiriamo un dado due volte*
Se E_1 è la probabilità che la somma dei risultati dei due lanci sia 8 o 10, E_2 la probabilità che al primo lancio esca 4 ed E_3 che al secondo esca 6, calcola $Pr(E_1E_2E_3)$ usando l'identità (10).
Soluzione: delle 36 coppie, 5 hanno somma 8 e 3 somma 10 (ellisse rossa in Figura 9), da cui segue che $Pr(E_1) = 2/9$. Delle 8 coppie nell'ellisse rossa solo $(4,4)$ e $(4,6)$ hanno 4 come primo risultato (intersezione con l'ellisse gialla), per cui $Pr(E_2|E_1) = 1/4$. Infine, solo $(4,6)$ ha 6 come secondo elemento (intersezione con l'ellisse azzurra), e quindi $Pr(E_3|E_1E_2) = 1/2$. Pertanto, si ha

$$Pr(E_1E_2E_3) = Pr(E_1)Pr(E_2|E_1)Pr(E_3|E_1E_2) = \frac{2}{9}\cdot\frac{1}{4}\cdot\frac{1}{2} = \frac{1}{36}$$

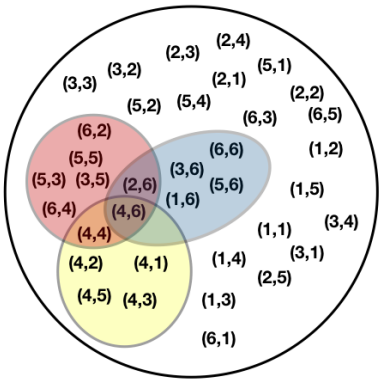


Figura 9: Vedi testo.

Numero di prove per ottenere un successo

Una variabile casuale geometrica X conta il numero di prove indipendenti richieste per ottenere un successo. Se p è la probabilità di successo per ogni prova, la probabilità che siano necessarie almeno i prove, $\Pr(X \geq i)$, è pari alla probabilità che le prime $i-1$ prove siano insuccessi, o

$$\Pr(X \geq i) = (1-p)^{i-1} \tag{11}$$

La probabilità di ottenere un successo *esattamente* al tentativo i è allora $\Pr(X = i) = (1-p)^{i-1}p$. Come per tutte le distribuzioni di probabilità abbiamo che

$$\sum_{i=1}^{\infty}\Pr(X = i) = \sum_{i=1}^{\infty}(1-p)^{i-1}p = 1 \tag{12}$$

Esercizio 6.2. *Calcolo del valore atteso della distribuzione geometrica*
Poiché $\mathbb{E}[X] = \sum_{i=1}^{\infty} i(1-p)^{i-1}p$ otteniamo

$$\mathbb{E}[X] = \sum_{i=1}^{\infty}(i-1+1)(1-p)^{i-1}p = \sum_{i=1}^{\infty}(i-1)(1-p)^{i-1}p + \sum_{i=1}^{\infty}(1-p)^{i-1}p = \sum_{j=0}^{\infty} j(1-p)^{j}p + 1$$

Nell'ultimo passaggio abbiamo posto $j = i - 1$ nella prima serie e usato l'uguaglianza (12) per la seconda. Riconoscendo che la prima serie non è altro che $(1-p)\mathbb{E}[X]$ ricaviamo infine

$$\mathbb{E}[X] = \frac{1}{p}$$

Esercizio 6.3. *Testa, quasi sicuramente*
Data una moneta che se lanciata restituisce *testa* con probabilità p, calcola quante volte devi lanciarla per avere il 99.9% di probabilità di ottenere *testa* almeno una volta.
Soluzione: poiché la probabilità di ottenere almeno una *testa* in n lanci è uguale al complemento a 1 della probabilità di ottenere n volte *croce*, scriviamo $(1-p)^n = 10^{-3}$ e otteniamo $n \approx -7/\ln(1-p)$. Per $p = 1/2$ abbiamo $n = 10$, mentre per $p = 1/10$ $n = 66$.

Un algoritmo Monte Carlo

Studiamo ora un algoritmo randomizzato per determinare un taglio minimo, (vedi il *Motwani e Raghavan* [1]). Consideriamo un multigrafo G connesso e non orientato con n vertici e introduciamo un algoritmo di *contrazione* che produce un insieme di archi che potrebbe essere un taglio minimo.

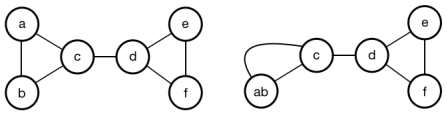


Figura 10: Contrazione dell'arco che unisce i vertici a e b.

Algoritmo 6.2. *MCMinCut*(G)
Input: G, un multigrafo connesso e non orientato con n vertici
Output: C, un insieme di archi candidato a essere un taglio minimo

---

`for` $i = n$ `to` 2

1. campiona un arco G con probabilità uniforme e identifica i suoi vertici, u e v, in un nuovo vertice uv (vedi Figura 10 con $u = a$ e $v = b$)
2. rimuovi tutti gli archi che univano i vertici u e v, incluso quello campionato, e mantieni tutti gli archi che incidono sul nuovo vertice uv
3. $i \leftarrow i - 1$

C è costituito dagli archi che uniscono gli ultimi due vertici rimasti di G

A ogni iterazione la cardinalità dell'insieme dei vertici del multigrafo diminuisce di un'unità mentre il grado del nuovo vertice uv non è mai inferiore al grado di u e v. L'insieme C degli archi che uniscono i due vertici rimasti è pertanto un buon *candidato* a essere un taglio minimo.

Dimostriamo che *MCMinCut* restituisce l'output corretto con probabilità $\hat{p} \geq 2/n^2$. Sia S un taglio minimo di dimensione k per G. **Il multigrafo G deve avere almeno $nk/2$ archi perché altrimenti esisterebbe almeno un vertice di grado minore di k i cui archi costituiscono un taglio minimo di dimensione strettamente minore di k.** Selezioniamo ogni arco con probabilità uniforme e stimiamo la probabilità che nessun arco di S sia contratto durante ogni iterazione. Sia E_i l'evento

di *non selezionare* un arco di S all'i-esima iterazione per $1 \leq i \leq n-2$. La probabilità p_1 che un arco campionato a caso nella prima iterazione sia in S è al più

$$p_1 \leq \frac{k}{nk/2} = \frac{2}{n} \quad \text{per cui} \quad \Pr(E_1) \geq 1 - \frac{2}{n}$$

Alla seconda iterazione il multigrafo avrà almeno $(n-1)k/2$ archi. Se l'evento E_1 si è realizzato, la probabilità p_2 che un arco campionato a caso sia in S è al più

$$p_2 \leq \frac{k}{(n-1)k/2} = \frac{2}{n-1} \quad \text{per cui} \quad \Pr(E_2|E_1) \geq 1 - \frac{2}{n-1}$$

All'i-esima iterazione il multigrafo avrà almeno $(n-i+1)k/2$ archi. Se gli eventi $E_1, \ldots, E_{i-1}$ si sono realizzati, la probabilità p_i che un arco campionato a caso sia in S è al più

$$p_i \leq \frac{k}{(n-i+1)k/2} = \frac{2}{n-i+1} \quad \text{per cui} \quad \Pr(E_i|E_1E_2 \ldots E_{i-1}) \geq 1 - \frac{2}{n-i+1}$$

Scrivendo $\hat{p}$ per $\Pr(E_1 \ldots E_{n-2})$ e applicando la *regola di moltiplicazione* (10) otteniamo

$$\hat{p} \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2} \frac{n-i-1}{n-i+1} = \frac{(n-2)\cdot(n-3)\ldots 2\cdot 1}{n\cdot(n-1)\ldots 4\cdot 3} = \frac{2}{n(n-1)}$$

Quindi *MCMinCut* restituisce un taglio minimo con probabilità $\hat{p} \geq 2/n^2$. Se eseguiamo *MCMinCut* per m volte, dalla formula (11) **per la probabilità $\mathcal{P}(m)$ di non ottenere un taglio minimo in m *run* otteniamo**

$$\mathcal{P}(m) < \left(1 - \frac{2}{n^2}\right)^{m-1}$$

che, al crescere di m, si avvicina a 0. Per $m = 4n^2$, ad esempio, si ha $\mathcal{P}(m) < 0.0003$.

Osservazione 6.1. *Scarsa efficacia su grafi con molti nodi*
MCMinCut restituisce un taglio minimo solo se, **a ogni iterazione**, **non campiona mai archi che compongono il taglio minimo stesso**. Per n grande la probabilità $\hat{p}$ di ottenere un taglio minimo è piccola il che rende *MCMinCut* poco appetibile al crescere del numero di nodi, non diversamente dal caso discusso nell'**Esercizio 1.2**.

Compito 6.1. *Implementazione di MCMinCut*
Genera il grafo di *Fritsch* in Figura 11 e calcola la frequenza empirica $\hat{p}$ con la quale ottieni un taglio

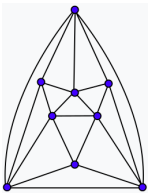


Figura 11: Il grafo di *Fritsch* ha $n = 9$ vertici e 21 archi.

minimo applicando *MCMinCut* 10^5 volte. Utilizza $\hat{p}$ per calcolare il numero di *run* R necessari per ottenere il taglio minimo con una probabilità del 99.9%.

7 Accordo bizantino

Il problema di come raggiungere il consenso in un sistema distribuito, introdotto da Lamport *et al.* [6], ha importanti implicazioni in moltissimi campi sia teorici sia applicativi. Partiamo con un semplice esempio limitandoci al caso in cui tutti i processi sono sullo stesso piano e ricevono sempre tutti i messaggi.

Ricerca del consenso

Durante una battaglia l'esercito bizantino è diviso in tre accampamenti al comando dei generali $genU$, $genV$ ed $genZ$. Uno tra $genU$, $genV$ e $genZ$ è un traditore ma la sua identità è ignota ai due generali leali. **Se le armate dei generali leali attaccano o si ritirano assieme, la maggior parte dei soldati tornerà a casa. Se, invece, prendono una decisione diversa entrambe le armate andranno incontro a una pesante disfatta.** Ogni sera, i generali possono comunicare tra loro spedendosi un messaggio in cui manifestano la loro intenzione di **attaccare**, *attack*, o di **ritirarsi**, *withdraw*. I generali leali spediscono un messaggio consistente (lo stesso messaggio agli altri due generali), mentre il generale traditore può spedire messaggi diversi al fine di infliggere il massimo danno. Sulla base dei contenuti dei messaggi ricevuti la sera successiva ogni generale può aggiornare il contenuto del messaggio che invierà. Esiste un protocollo di comunicazione capace di garantire il raggiungimento del consenso tra i due generali leali in un numero finito di *round*? Supponiamo che il generale traditore sia $genZ$ e diamo l'**intuizione** del perché la risposta è *negativa*.

Inizialmente in accordo: $genU$ e $genV$ spediscono entrambi lo stesso messaggio, per esempio *attack*. Il generale traditore $genZ$, il cui interesse è boicottare il raggiungimento del consenso, spedisce invece *retreat* a entrambi. I generali leali $genU$ e $genV$ ricevono quindi messaggi discordanti e, non essendo in grado di capire quale sia stato spedito dal generale leale e quale dal traditore, *si trovano in stallo*.

Inizialmente in disaccordo: mentre $genU$ spedisce *attack*, $genV$ spedisce *retreat*. Il generale traditore $genZ$, questa volta, spedisce *attack* a $genU$ e *retreat* a $genV$. Nuovamente, i generali leali $genU$ e $genV$ ricevono messaggi discordanti e, non essendo in grado di capire quale sia stato spedito dal generale leale e quale dal traditore, *si trovano in stallo*.

Poniamo ora il problema in termini più formali. Assumiamo di avere n processi, f dei quali *faulty*. Gli f processi *faulty* sono fissati ma non identificabili. Limitiamo la nostra analisi a come gli $n - f$ processi affidabili possano raggiungere il consenso su un solo *bit* nel caso *sincrono*, ovvero nel caso in cui, a ogni *round*, ricevono sempre n messaggi. Questa condizione è necessaria solo per gli algoritmi deterministici ma riduce notevolmente la difficoltà del problema. La comunicazione tra processi è affidabile e senza rumore. Senza perdita di generalità assumiamo infine che i processi affidabili siano i primi $n - f$ indicati con $p_1, \ldots, p_{n-f}$.

Problema 7.1. *Accordo bizantino*
Sia $b(j) \in \{0, 1\}$ il valore del *bit* del processo p_j per $j = 1, \ldots, n - f$. Seguendo le specifiche

(1) durante ogni *round* ognuno degli n processi spedisce un *messaggio* a ogni altro processo
(2) prima dell'inizio di un nuovo *round* ogni processo ha ricevuto un *messaggio* da ogni altro processo
(3) nello stesso *round* ogni processo affidabile spedisce lo stesso *messaggio* a ogni altro processo

ogni processo affidabile deve seguire **un protocollo che, dopo un numero finito di *round*, termina con una *decisione finale*** che rispetta i requisiti di

Consenso: i *bit* dei processi affidabili assumono tutti lo stesso valore, ovvero, dopo un certo numero di *round*, se p_j è un qualunque processo affidabile allora $b(j) = v$;

Validità: se il valore iniziale di tutti i processi affidabili è lo stesso, ovvero $b_0(j) = v^*$ per ogni processo affidabile p_j, allora $v = v^*$.

Per il vincolo di **validità** il raggiungimento del consenso, pertanto, non può essere basato su un valore concordato di *default*.

Si può dimostrare [6] che il consenso può essere raggiunto se e solo se

$$n \geq 3f + 1 \tag{13}$$

Vediamo ora a grandi linee un algoritmo deterministico basato sull'*Exponential Information Gathering (EIG)* che consente il raggiungimento di un accordo in $f + 1$ *round*. Per tutti gli $f + 1$ *round* ogni processo affidabile p_j spedisce sempre il valore iniziale $b_0(j)$ e costruisce un albero che cresce di un livello per *round*. Per semplicità assumiamo che ogni processo spedisca il proprio messaggio anche a se stesso.

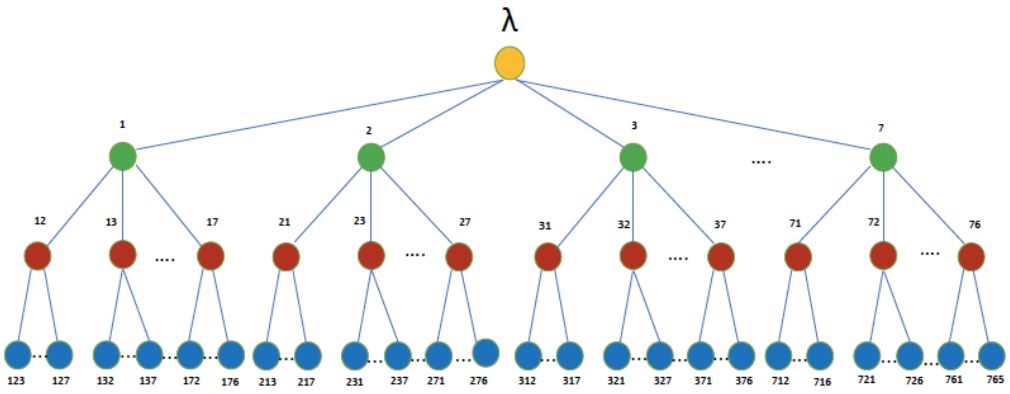


Figura 12: Vedi testo.

La Figura 12 mostra l'albero costruito da un processo affidabile p nel caso $n = 7$ ed $f = 2$ e che consiste quindi di tre livelli. Al primo *round* ai 7 figli della radice, inizializzata a λ, p assegna i valori dei *bit* ricevuti da tutti e 7 i processi. Al *round* successivo ogni processo spedisce i valori del livello corrente del proprio albero (come sempre, solo i processi affidabili devono applicare il protocollo coerentemente). Il processo p raggruppa i valori ricevuti in un nuovo livello del proprio albero seguendo le *dicerie* riguardo il valore del bit di ognuno dei 7 processi. Al secondo *round*, pertanto, i 6 figli di un qualunque nodo j (con $j = 1, \ldots, 7$) contengono il valore del bit di p_j che p_k, per tutti i sei valori di k diversi da j, **dice** di aver ricevuto al *primo round* da p_j. Al terzo livello, ultimo poiché $f = 2$, la foglia con etichetta jkh contiene il valore del bit di p_j che p_h, per tutti i cinque valori di h diversi da k, **dice** di aver ricevuto al *secondo round* da p_k, per tutti i sei valori di k diversi da j. Nel caso generale, ogni nodo ha $2f + 1$ foglie. **Il processo p procede ricorsivamente dalle foglie e aggiorna il valore di ogni nodo col valore maggioritario dei figli (o un valore di *default* nel caso di pareggio). Il valore v della radice è lo stesso in tutti gli alberi dei processi affidabili ed è la decisione finale che soddisfa i requisiti dell'accordo bizantino**. La complessità dell'algoritmo è lineare, ma il costo delle comunicazioni cresce esponenzialmente come $O(n^{f+1})$.

Osservazione 7.1. *Cenni alla correttezza*
La dimostrazione che per questo algoritmo il valore della radice è lo stesso per tutti gli alberi dei processi affidabili (vedi [6]) è basata sul principio di induzione ed è piuttosto complessa. Ci accontentiamo di alcune considerazioni intuitive limitate al caso in cui $n = 3f + 1$. Il punto centrale è che i processi affidabili trasmettono sempre affidabilmente i valori ricevuti. Nel caso in cui il valore iniziale è lo stesso per tutti i processi affidabili, la correttezza è garantita dal fatto che nella ricorsione questo valore emerge sempre nei $2f + 1$ figli della radice corrispondenti ai processi affidabili. Eventuali azioni di disturbo dei processi *faulty* sono ininfluenti. Nel caso in cui i valori iniziali siano diversi tra loro, la forza dell'*EIG* risiede nel fatto che i valori riportati dai processi non affidabili sono propagati affidabilmente dai processi affidabili e quindi partecipano al raggiungimento del consenso. La Figura 13 mostra, e discute in dettaglio, il caso $f = 1$.

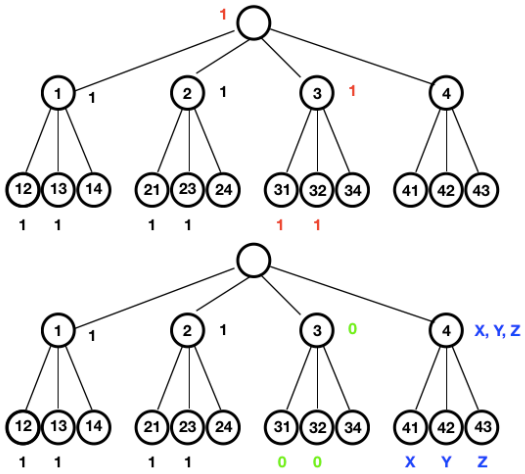


Figura 13: Albero costruito dai processi p_1, p_2 e p_3 se $b_0(1) = b_0(2) = 1$ e $b_0(3) = 1$ (rosso, in alto) e $b_0(3) = 0$ (verde, in basso). Nel primo caso i primi tre nodi sono sempre valutati a 1 e, indipendentemente dai valori spediti da p_4, il valore della radice è sempre 1 e soddisfa il requisito di validità. Nel secondo caso i valori spediti dal processo p_4 al primo *round* - X per p_1, Y per p_2 e Z per p_3 - compaiono come foglie del quarto nodo in tutti e tre gli alberi. In questo modo i valori di X, Y e Z determinano lo stesso valore per il quarto nodo e quindi per tutte e tre le radici, garantendo il raggiungimento di un accordo. In questo caso, se il valore del quarto nodo è 1 l'accordo è raggiunto sul valore 1, altrimenti sul valore di *default*.

Protocollo *Monte Carlo*

Per procedere con il protocollo randomizzato abbiamo bisogno di un'ultima ipotesi.
Moneta globale: a ogni *round* è comunicato a tutti i processi il risultato del lancio di una *moneta* con probabilità

$$\Pr(testa) = \Pr(croce) = \frac{1}{2}$$

Osservazione 7.2. *La moneta globale non è un'autorità*
Il fatto che tutti i processi, a ogni *round*, abbiano accesso all'esito del lancio di una moneta globale non prefigura l'esistenza di un'autorità centrale. Il vincolo di validità, infatti, preclude la possibilità di raggiungere un accordo basato sull'esito del lancio di una moneta. □

Il punto di vista è quello del generico processo affidabile j. Introduciamo la soglia $T = 2f + 1$.

Algoritmo 7.1. *MCByzantineGeneral*
Input: $b(j) = b_0(j)$
Output: con probabilità almeno 1/2 $b(j) = v$

`while (TRUE)`

1. trasmetti $b(j)$ agli altri $n - 1$ processi
2. ricevi i valori spediti dagli altri $n - 1$ processi
3. $maj(j) \leftarrow$ valore maggioritario tra i ricevuti (incluso il proprio)
4. $tally(j) \leftarrow$ numero dei valori uguali a $maj(j)$
5. `if` $tally(j) \geq T$
   `then` $b(j) \leftarrow maj(j)$
   `else if` *testa*
   `then` $b(j) \leftarrow 1$
   `else` $b(j) \leftarrow 0$

Analizziamo la correttezza di *MCByzantineGeneral*, algoritmo dovuto a Rabin [7]. Abbiamo due casi.

I $2f + 1$ processi affidabili sono unanimemente d'accordo: il consenso è raggiunto poiché al passo 5, per un qualunque processo affidabile j, risulta che $tally(j) \geq T$. Questo è vero sia se $b(j) = v_0$ al primo *round*, sia se $b(j) = v$ in qualche *round* successivo.

Non tutti i processi affidabili concordano: dobbiamo considerare due ulteriori casi. Nel primo non esiste alcun processo affidabile j per cui $tally(j) \geq T$: il consenso è quindi raggiunto nel primo *round* e coincide con l'esito del lancio della moneta. Nel secondo, per qualche processo j^* avremo $tally(j^*) \geq T$. Poiché i processi *faulty* sono al più f e per almeno $f + 1$ processi affidabili il bit spedito è $maj(j^*)$, non può esserci alcun processo affidabile k^* per il quale

$$tally(k^*) \geq T \quad \text{con} \quad maj(k^*) \neq maj(j^*)$$

Pertanto, con probabilità 1/2, dopo l'esito del lancio della moneta, tutti i processi affidabili convergeranno su $maj(j^*)$.

Osservazione 7.3. *Uno solo, ma importante*
Dalla disuguaglianza (13), otteniamo che i processi affidabili devono essere almeno $T = 2f + 1$. Il motivo di questa soglia è facile da spiegare. Poiché i processi *faulty* sono f, se in un *round* un processo affidabile riceve almeno T bit con lo stesso valore, almeno $f + 1$ processi affidabili devono essere concordi su quel valore. In quello stesso round, pertanto, nessun processo affidabile può aver ricevuto T bit con il valore complementare.

Osservazione 7.4. *Costante invece che lineare!*
MCByzantineGeneral converge in un numero atteso di *round* che non dipende da f. Il valore atteso del numero di *round* per il raggiungimento della convergenza, infatti, è 2. Il costo delle comunicazioni, inoltre, è lineare.

Compito 7.1. *Caso minimale di MCByzantineGeneral*
È dato un sistema distribuito costituito da $n = 4$ processi di cui il quarto è *faulty*. I tre processi affidabili seguono fedelmente il protocollo *Monte Carlo* mentre il processo *faulty* è malizioso: a ogni *round*, spedisce al processo affidabile j (con $j = 1, 2$ e 3) il *bit* $1 - b(j)$. Considera $R = 2^{10}$ *run* nei quali i bit iniziali sono divisi nell'unico *split* interessante, 2 a 1. Grafica la frazione dei *run* in cui l'accordo è raggiunto in $r = 1, 2, \ldots, 10$ *round* e commenta ciò che ottieni. Perché se in un qualche round per un processo affidabile j risulta che

$$tally(j) \geq T$$

e l'esito del lancio della moneta coincide con $maj(j)$, il round successivo l'accordo è certamente raggiunto?

Compito 7.2. *L'EIG per $n = 3$ ed $f = 1$ non funziona!*
Tre amici vogliono andare assieme al cinema e sono indecisi tra due film. Il primo film è in centro, il secondo in periferia. Due dei tre amici sono affidabili e sinceramente interessati a trovare un accordo, mentre il terzo è dispettoso. In caso di pareggio la scelta ricade sul cinema in centro. Ciascuno dei due amici affidabili non sa quale degli altri due sia l'amico dispettoso. Costruendo gli alberi opportuni per i due amici affidabili, verifica che l'amico dispettoso, se in grado di intercettare i messaggi degli altri due, riesce a sabotare il raggiungimento di un accordo tranne che nel caso in cui i due amici siano inizialmente concordi sul cinema in centro. Per non scoprirsi l'amico dispettoso, in entrambi i *round*, non cambia il titolo del film che spedisce a ognuno degli altri due amici.

8 Test di primalità

Vediamo ora un algoritmo *Monte Carlo* per determinare la primalità di n che sfrutta proprietà elementari della teoria delle congruenze, vedi capitolo 5 dell'Apostol [8], evitando la ricerca di un'eventuale scomposizione in fattori primi di n. Per valutare l'efficacia dell'algoritmo è necessario disporre di un teorema fondamentale della teoria dei gruppi finiti. Il materiale è adattato dal *Kormen et al.* [9].

Minimalia su teoria dei numeri

Definizione 8.1. *Congruenza*
Sia $n \neq 0 \in \mathbb{Z}$: a ed $b \in \mathbb{Z}$ sono *congruenti modulo* n se $a-b$ è divisibile per n. Nel caso, scriviamo

$$a \equiv b \pmod{n}$$

Esempio 8.1. *Alcune congruenze*
Abbiamo che $5 \equiv 8 \pmod 3$. Inoltre, $12 \equiv 4 \pmod 8$, $12 \equiv 4 \pmod 4$ e $12 \equiv 4 \pmod 2$.

Osservazione 8.1. *Congruenza come relazione di equivalenza*
La *congruenza* è una relazione di equivalenza su $\mathbb{Z}$: infatti, è evidente che (1) $\forall a$ $a \equiv a$ (riflessività), (2) $\forall a$ e b $a \equiv b$ se e solo se $b \equiv a$ (simmetria) e (3) $\forall a, b$ e c se $a \equiv b$ e $b \equiv c$ allora $a \equiv c$ (transitività). L'insieme quoziente $\mathbb{Z}_n^+ = \{0, 1, \ldots, n-1\}$ è il *sistema completo di residui*.

Definizione 8.2. *Coprimalità*
Se indichiamo con $\mathrm{MCD}(m, n)$ il *massimo comun divisore* di m ed $n \in \mathbb{Z}$, m ed n sono *coprimi* se $\mathrm{MCD}(m, n) = 1$.

Osservazione 8.2. *Invarianza per moltiplicazione*
Se $a \equiv b \pmod n$, allora $ad \equiv bd \pmod n$. Il contrario, invece, è vero solo se $\mathrm{MCD}(n, d) = 1$. Ad esempio, $42 \equiv 18 \pmod 8$ mentre $7 \not\equiv 3 \pmod 8$ poiché $\mathrm{MCD}(8, 6) = 2$.

Definizione 8.3. *Gruppo moltiplicativo modulo* n
L'insieme $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n^+ \setminus \{0\} \mid \mathrm{MCD}(a, n) = 1\}$, sottoinsieme di $\mathbb{Z}_n^+$, è un gruppo rispetto alla moltiplicazione $\pmod n$. Un altro modo di dire la stessa cosa è che $\mathbb{Z}_n^*$ è costituito da tutti e soli gli elementi di $\mathbb{Z}_n^+$ che ammettono un *inverso* moltiplicativo. □

L'ordine di Z_n^* è indicato con $\phi(n)$. Se n è primo, $\phi(n) = n - 1$, altrimenti $\phi(n) < n - 1$.

Esempio 8.2. *Valutazione di $\phi(n)$ in un caso non banale*
Sia $n = p^\alpha$ con p primo e $\alpha \geq 1 \in \mathbb{Z}$. Poichè i multipli di p tra 1 e p^α sono $p^\alpha/p = p^{\alpha-1}$, abbiamo $\phi(n) = p^\alpha - p^{\alpha-1}$. Per esempio, per $n = 9 = 3^2$, abbiamo $\phi(9) = |\mathbb{Z}_9^*| = 3^2 - 3 = 6$.

Enunciamo ora una proprietà importante dei numeri primi.

Proposizione 8.1. *Radici quadrate dell'unità*
Se p è primo, $x_+ = 1$ e $x_- = -1$ sono le uniche soluzioni dell'equazione

$$x^2 \equiv 1 \pmod p$$

Dimostrazione: da

$$x^2 - 1 = (x-1) \cdot (x+1) \equiv 0 \pmod p$$

otteniamo che almeno uno tra $x - 1$ e $x + 1$ è congruente a 0 e quindi che, per la primalità di p, $x_+ = 1$ o $x_- = -1 = p - 1$. ■

Minimalia su teoria dei gruppi

Sia G un gruppo finito e H un suo sottogruppo. Indichiamo con $|G|$ l'*ordine* di G, ovvero il numero dei suoi elementi. Se $a \in G$, l'insieme

$$Ha = \{ha : \ h \in H\}$$

è la *classe laterale destra* di H rappresentata da a. Chiaramente per $h \in H$, $Hh = H$. Abbiamo bisogno di un lemma preparatorio.

Lemma 8.1. *Le classi laterali destre sono uguali o disgiunte*
$\forall r$ e $s \in G$, $Hr = Hs$ se e solo se $rs^{-1} \in H$, altrimenti $Hr \cap Hs = \emptyset$.

Dimostrazione: se $rs^{-1} = h \in H$, allora

$$H = Hh = (Hr)s^{-1}$$

Moltiplicando a destra per s il primo e l'ultimo membro dell'uguaglianza otteniamo

$$Hs = Hr.$$

Se invece $Hs = Hr$, poiché $r \in Hr$ abbiamo che $r = h's$ per qualche $h' \in H$. Moltiplicando a destra per s^{-1} quest'ultima uguaglianza otteniamo

$$rs^{-1} = h' \in H.$$

Supponiamo infine che Hr ed Hs abbiamo qualche elemento in comune. Questo significa che per qualche $h_1, h_2 \in H$ abbiamo

$$h_1 r = h_2 s.$$

Allora moltiplicando a sinistra per h_1^{-1} e a destra per s^{-1} otteniamo

$$rs^{-1} = h_1^{-1} h_2 \in H \quad \rightarrow \quad Hr = Hs$$

■

Teorema 8.1. *Lagrange*
Se H è un sottogruppo di G, allora $|G| = n|H|$ con $n > 0 \in \mathbb{N}$ e $|H|$ l'*indice* di H in G. Inoltre esistono $g_1, \ldots, g_n \in G$ tali che $G = Hg_1 \cup \cdots \cup Hg_n$

Dimostrazione: prendiamo un elemento $g_1 \in H$ e osserviamo che $|Hg_1| = |H|$. Se $Hg_1 \neq G$, allora prendiamo un elemento $g_2 \in G \setminus Hg_1$. Per il **Lemma 8.1** Hg_1 e Hg_2 sono disgiunti e abbiamo $|Hg_1 \cup Hg_2| = 2|H|$. Ma G ha un numero finito di elementi e dopo n passi per qualche intero positivo n avremo ottenuto

$$G = Hg_1 \cup \cdots \cup Hg_n$$

■

Un'applicazione immediata del **Teorema 8.1** si ottiene introducendo la nozione di ordine di un elemento di un gruppo finito.

Definizione 8.4. *Ordine*
L'ordine di un elemento a di un gruppo finito G è il più piccolo numero naturale g_a per cui $a^{g_a} = 1$.
□

Fatto 8.1. *Sottogruppi e sottomultipli*
$C(a) = \{a, a^2, \ldots, a^{g_a}\}$ è un sottogruppo di G. Per il **Teorema 8.1**, $C(a) \subset G$ implica $g_a \leq |G|/2$.

Algoritmo di Miller-Rabin

Premettiamo un teorema fondamentale.

Teorema 8.2. *Eulero*
Se $a \in Z_n^*$ con n intero qualunque, allora

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Dimostrazione: per ogni $a \in Z_n^*$ abbiamo necessariamente che $g_a \leq \phi(n)$. Sappiamo che l'insieme

$$C = \{a, a^2, \ldots, a^{g_a}\}$$

è un sottogruppo di Z_n^* di ordine g_a. Ma per il **Teorema 8.1** abbiamo che $mg_a = \phi(n)$ per qualche intero $m \geq 1$ e, quindi, che $a^{\phi(n)} = a^{mg_a} = (a^{g_a})^m \equiv 1 \pmod{n}$. ■

Se n è primo, allora $\phi(n) = n - 1$ e il **Teorema 8.2** è noto come Piccolo Teorema di Fermat.

Teorema 8.3. *Piccolo di Fermat*
Se p è primo e a un intero qualunque non divisibile per p

$$a^{p-1} \equiv 1 \pmod{p}$$

Osservazione 8.3. *Riscrittura di un numero dispari*
Un qualunque $n \geq 3$ dispari può sempre essere riscritto con q dispari e $s \geq 1$ come

$$n = q \cdot 2^s + 1$$

□

Conseguentemente, per qualunque $a \in \{2, 3, \ldots n-2\}$, abbiamo la seguente scomposizione

$$\begin{aligned} a^{n-1} - 1 &= (a^q)^{2^s} - 1 = \left((a^q)^{\frac{2^s}{2}} + 1\right)\left((a^q)^{\frac{2^s}{2}} - 1\right) \\ &= \left((a^q)^{\frac{2^s}{2}} + 1\right)\left((a^q)^{\frac{2^s}{2^2}} + 1\right)\left((a^q)^{\frac{2^s}{2^2}} - 1\right) \\ &= \ldots \\ &= \left((a^q)^{\frac{2^s}{2}} + 1\right)\left((a^q)^{\frac{2^s}{2^2}} + 1\right)\ldots\left((a^q)^{\frac{2^s}{2^s}} + 1\right)\left((a^q)^{\frac{2^s}{2^s}} - 1\right) \\ &= \overbrace{\left(a^{q2^{s-1}} + 1\right)\left(a^{q2^{s-2}} + 1\right)\ldots(a^q + 1)}^{s \text{ fattori}}(a^q - 1) \end{aligned} \tag{14}$$

Per il Teorema 8.3, se n è primo uno dei fattori di questa scomposizione deve essere nullo. Possiamo ora presentare un test di primalità dovuto a Miller [10] e Rabin [11] (che forse più correttamente dovrebbe essere presentato come test per determinare se un numero è composto).

Algoritmo 8.1. *MCPrimalityTest*(n)
Input: $n \geq 3$ dispari
Output: n `composto` o `probabilmente primo`

---

1. $s = 0$
2. $q = n - 1$
3. `while` (q pari)
   $s \leftarrow s + 1$
   $q \leftarrow q/2$
4. campiona a uniformemente a caso in $\{2, 3, \ldots, n-2\}$
5. calcola $x \equiv a^q \pmod{n}$
6. `if` $x \equiv \pm 1$
   `return` n `probabilmente primo`
7. `while` $s - 1 \geq 0$
   $x \equiv x^2 \pmod{n}$
   `if` $x \equiv -1 \pmod{n}$
   `return` n `probabilmente primo`
   $s \leftarrow s - 1$
8. `return` n `composto`

Per il **Teorema 8.3**, se n è primo, l'output del *MCPrimalityTest* per una qualunque base a è sempre n `probabilmente primo`. Se per qualche a *MCPrimalityTest* produce l'output n `composto`, a è un **MR- testimone** per n. In questo caso, sempre per il **Teorema 8.3**, n è **certamente composto**. Un numero a che produce l'output n `probabilmente primo` per n composto, invece, è un **MR-bugiardo** per n.

Osservazione 8.4. *Solo due tipologie di sequenze possibili* $(\mathrm{mod}\ n)$*, per* n *primo*
Per n primo fissiamo a. Per via della fattorizzazione (14) e del fatto che se $x \equiv 1$ o $x \equiv -1$ allora $x^2 \equiv 1$, la sequenza delle $s+1$ quadrature $\{a^q, a^{q2}, \ldots, a^{q2^{s-1}}, a^{q2^s}\}$ ottenute è solo di due tipologie:

$$\overbrace{(1,\ldots,1)}^{\text{tutti 1}} \ \text{o}\ (\ldots,-1,\overbrace{1,\ldots,1}^{\text{tutti 1}}) \tag{15}$$

La prima si ha se $a^q \equiv 1$, la seconda se per qualche $i = 0, \ldots, s$ si ottiene $a^{q2^i} \equiv -1$. In questo caso con $a^{q2^{i-1}} \not\equiv 1$ se $i > 0$ e con la sequenza che si chiude con -1 se $i = s$. *MCPrimalityTest* cerca un MR-testimone della non primalità di $n = q2^s + 1$ scegliendo $a \in \{2, \ldots, n-1\}$ casualmente. Partendo da a^q e procedendo per quadrature successive, se la sequenza delle quadrature non appartiene a una delle due tipologie in (15), a è **certamente** un MR-testimone ed n composto.

Osservazione 8.5. *MR-bugiardi e MR-testimoni banali*
Per n composto, 1 ed $n-1$ sono sempre MR-bugiardi. Inoltre, un a per cui $\mathrm{MCD}(n, a) > 1$ è sempre un MR-testimone per n poiché $a \notin \mathbb{Z}_n^*$.

Esempio 8.3. *Gli MR-bugiardi sono contenuti in un sottogruppo*
Per $n = 65$ abbiamo $|\mathbb{Z}_{65}^*| = 48$, $q = 1$ ed $s = 6$. Un controllo esaustivo mostra che ci sono 2 coppie di MR-bugiardi non banali, in quanto, sempre $(\mathrm{mod}\ 65)$,

$8^1 \equiv 8$	$\mathbf{8^2} = 64 \equiv \mathbf{-1}$	$8^4 \equiv 1$	...	`probabilmente primo`
$57^1 \equiv 57$	$\mathbf{57^2} = 49 \cdot 65 + 64 \equiv \mathbf{-1}$	$57^4 \equiv 1$	...	`probabilmente primo`
$18^1 \equiv 18$	$\mathbf{18^2} = 4 \cdot 65 + 64 \equiv \mathbf{-1}$	$18^4 \equiv 1$	...	`probabilmente primo`
$47^1 \equiv 47$	$\mathbf{47^2} = 33 \cdot 65 + 64 \equiv \mathbf{-1}$	$47^4 \equiv 1$	...	`probabilmente primo`

Tuttavia, $8 \cdot 18 \equiv 14$ e $8 \cdot 47 \equiv 51$ con $14^4 \equiv 1$ e $51^4 \equiv 1$. Pertanto, 14 e 51 sono MR-testimoni e l'insieme $H_{65} = \{1, 8, 14, 18, 47, 51, 57, 64\}$ è un sottogruppo proprio di $\mathbb{Z}_{65}^*$.

Gli MR-testimoni sono in maggioranza

Per ogni n l'insieme

$$H_n = \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\}$$

è un sottogruppo di $\mathbb{Z}_n^*$. Se n è composto, H_n contiene tutti i MR-bugiardi. **Se H_n è un sottogruppo proprio di $\mathbb{Z}_n^*$, dal Teorema 8.1 segue**

$$|H_n| \leq \frac{|\mathbb{Z}_n^*|}{2} \leq \frac{n-1}{2}$$

Conseguentemente, il numero di MR-bugiardi è certamente minore di $(n-1)/2$, ovvero più della metà delle basi sono MR-testimoni per n.

I numeri dispari per i quali $H_n = \mathbb{Z}_n^*$ sono noti come numeri di *Carmichael* e, seppur non frequenti, sono infiniti. I primi tre numeri di *Carmichael* sono

$$561 = 3 \cdot 11 \cdot 17, \quad 1105 = 5 \cdot 13 \cdot 17 \quad \text{e} \quad 1729 = 7 \cdot 13 \cdot 19$$

La dimostrazione che anche per un numero di *Carmichael* gli MR-testimoni sono in maggioranza è data in appendice e **si appoggia sull'esistenza di un MR-testimone non banale, ovvero di una base** $\bar{a} \in H_n$ (e, quindi, di un sottogruppo proprio di H_n che contiene tutti gli MR-bugiardi). Per la frazione $\mathcal{P}(n)$ di MR-bugiardi di un numero composto n otteniamo nuovamente

$$\mathcal{P}(n) < \frac{|\mathbb{Z}_n^*|/2}{(n-1)} < \frac{(n-1)/2}{(n-1)} = \frac{1}{2}$$

e, quindi, la probabilità che un numero n composto sia dichiarato k volte dal *MCPrimalityTest* `probabilmente primo` non è più grande di $1/2^k$. Per via dell'efficienza dell'algoritmo che calcola quadrature successive, se $\log n$ sono i bit necessari per codificare un numero n, il costo di *MCPrimalityTest* ripetuto k volte è $O(k(\log n)^3)$. Poiché il miglior algoritmo deterministico [12] costa $O((\log n)^6)$, il *MCPrimalityTest* è ancora il più utilizzato nelle applicazioni crittografiche.

Compito 8.1. *Pochi MR-bugiardi*
Determina tutti gli MR-bugiardi dei primi sette numeri di *Carmichael*: 561, 1105, 1729, 2465, 2821, 6601 e 8911.

9 Verifiche di uguaglianza

Vediamo ora due algoritmi *Monte Carlo* molto efficienti basati su semplici proprietà algebriche. **Entrambi gli algoritmi utilizzano una funzione casuale M che verifica l'uguaglianza tra due elementi x e y appartenenti a un universo U grande, a partire dalle loro immagini $M(x)$ ed $M(y)$, o *fingerprint*, appartenenti a un universo V molto più piccolo**.

Controllo della moltiplicazione fra matrici

L'algoritmo per moltiplicare due matrici $n \times n$ $\mathbf{A}$ e $\mathbf{B}$ richiede $O(n^3)$ moltiplicazioni. Se $\mathbf{C} = \mathbf{AB}$, infatti, servono n moltiplicazioni per calcolare ognuno degli $n \times n$ elementi

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Nel caso di matrici 2×2, tuttavia, si può ottenere lo stesso risultato con 7 moltiplicazioni ponendo

$$\begin{array}{ll} I = (a_{11} + a_{22})(b_{11} + b_{22}) & II = (a_{21} + a_{22})\, b_{11} \\ III = a_{11}(b_{12} - b_{22}) & IV = a_{22}(-b_{11} + b_{21}) \\ V = (a_{11} + a_{12})\, b_{22} & VI = (-a_{11} + a_{21})(b_{11} + b_{12}) \\ VII = (a_{12} - a_{22})(b_{21} + b_{22}) & \end{array}$$

troviamo infatti che

$$\begin{array}{rr} I + IV - V + VII = a_{11}b_{11} + a_{12}b_{21} = c_{11} & III + V = a_{11}b_{12} + a_{12}b_{22} = c_{12} \\ II + IV = a_{21}b_{11} + a_{22}b_{21} = c_{21} & I + III - II + VI = a_{21}b_{12} + a_{22}b_{22} = c_{22} \end{array}$$

Questa riscrittura è alla base di un algoritmo ricorsivo che richiede $O(n^{\log_2 7}) = O(n^{2.807})$ moltiplicazioni [9]. Negli anni l'esponente è stato ripetutamente abbassato. Al momento l'algoritmo ottimale richiede $O(n^{2.372})$ moltiplicazioni, ma la verifica di una sua corretta implementazione è estremamente complicata. Un algoritmo *Monte Carlo* verifica l'uguaglianza **matriciale**

$$\mathbf{AB} = \mathbf{C} \tag{16}$$

con $O(n^2)$ moltiplicazioni verificando l'uguaglianza **vettoriale**

$$\mathbf{ABr} = \mathbf{Cr}$$

dove $\mathbf{r}$ è un vettore n-dimensionale casuale. Vediamo prima l'algoritmo e poi discutiamo la sua notevole efficienza.

Algoritmo 9.1. *MCMatrixMultiplicationVerifier*$(\mathbf{A}, \mathbf{B}, \mathbf{C})$
Input: $\mathbf{A}$, $\mathbf{B}$ e $\mathbf{C}$ matrici $n \times n$
Output: $\mathbf{AB} \neq \mathbf{C}$ o `probabilmente` $\mathbf{AB} = \mathbf{C}$

---

1. campiona $\mathbf{r}$ uniformemente a caso in $\{0,1\}^n$
2. $\mathbf{s} \leftarrow \mathbf{Br}$ # $Pr(r_j = 0) = Pf(r_j = 1) = 1/2$ per $j = 1, \ldots, n$
3. $\mathbf{t} \leftarrow \mathbf{As}$ # $\mathbf{t} = \mathbf{A}(\mathbf{Br})$
4. $\mathbf{u} \leftarrow \mathbf{Cr}$
5. `if` $\mathbf{t} = \mathbf{u}$
   `return probabilmente` $\mathbf{AB} = \mathbf{C}$
   `else`
   `return` "$\mathbf{AB} \neq \mathbf{C}$"

Osservazione 9.1. *Da 3 a 2*
I tre vettori n-dimensionali $\mathbf{s}$, $\mathbf{t}$ e $\mathbf{u}$ sono calcolati come prodotti di una matrice $n \times n$ con un vettore n-dimensionale e richiedono quindi solo $O(n^2)$ moltiplicazioni ciascuno.

Osservazione 9.2. *Una questione di associatività*
MCMatrixMultiplicationVerifier si fonda sul fatto che, sfruttando la proprietà associativa del prodotto, l'azione della matrice $\mathbf{AB}$ su un vettore $\mathbf{r}$ può essere calcolata a partire dalle matrici $\mathbf{A}$ e $\mathbf{B}$ in cascata. Infatti, abbiamo che

$$(\mathbf{AB})\mathbf{r} = \mathbf{A}(\mathbf{Br})$$

□

Limitiamo dall'alto $Pr(\mathbf{ABr} = \mathbf{Cr})$, ovvero la probabilità che *MCMatrixMultiplicationVerifier* restituisca `probabilmente` $\mathbf{AB} = \mathbf{C}$ quando invece $\mathbf{AB} \neq \mathbf{C}$.

Proposizione 9.1. *MCMatrixMultiplicationVerifier è un algoritmo Monte Carlo*
Se $\mathbf{AB} \neq \mathbf{C}$, allora

$$Pr(\mathbf{A}(\mathbf{Br}) = \mathbf{Cr}) \leq \frac{1}{2}$$

Dimostrazione: se $\mathbf{AB} \neq \mathbf{C}$ almeno un elemento della matrice $\mathbf{D} = \mathbf{AB} - \mathbf{C}$ deve essere non nullo. Supponiamo che sia l'elemento d_{ij} di riga i e colonna j. In questo caso $\mathbf{A}(\mathbf{Br}) = \mathbf{Cr}$ può verificarsi solo se la moltiplicazione dell'i-esima riga di $\mathbf{D}$ con il vettore colonna casuale $\mathbf{r}$ restituisce 0, ovvero se $\sum_k d_{ik} r_k = 0$. Isolando il termine che contiene d_{ij} e ponendo $y = \sum_{k \neq j} d_{ik} r_k$ otteniamo

$$\sum_k d_{ik} r_k = d_{ij} r_j + \sum_{k \neq j} d_{ik} r_k = d_{ij} r_j + y$$

Possiamo quindi riscrivere la probabilità di errore $Pr(\mathbf{A}(\mathbf{Br}) = \mathbf{Cr})$ con $Pr(d_{ij} r_j + y = 0)$. Poiché $Pr(y = 0 \cup y \neq 0) = 1$ sappiamo che $Pr(d_{ij} r_j + y = 0)$ può essere espressa come probabilità totale, ovvero

$$Pr(d_{ij} r_j + y = 0) = Pr(d_{ij} r_j + y = 0 | y = 0) Pr(y = 0) + Pr(d_{ij} r_j + y = 0 | y \neq 0) Pr(y \neq 0)$$

Ma

$$Pr(d_{ij} r_j + y = 0 | y = 0) = Pr(d_{ij} r_j = 0) = Pr(r_j = 0) = \frac{1}{2}$$

e

$$Pr(d_{ij} r_j + y = 0 | y \neq 0) = Pr(r_j = 1 \cap d_{ij} = -y) \leq Pr(r_j = 1) = \frac{1}{2}$$

da cui otteniamo

$$Pr(\mathbf{A}(\mathbf{Br}) = \mathbf{Cr}) = Pr(d_{ij} r_j + y = 0) \leq \frac{1}{2} Pr(y = 0) + \frac{1}{2} Pr(y \neq 0) = \frac{1}{2}$$

■

Forti di questo risultato, eseguendo *MCMatrixMultiplicationVerifier* k volte, possiamo quindi verificare l'uguaglianza (16) con probabilità

$$Pr(k) = 1 - \frac{1}{2^k}$$

Compito 9.1. *Implementazione di MCMatrixMultiplicationVerifier*
Costruisci $\mathbf{A}$ e $\mathbf{B}$ matrici 100×100 campionando uniformemente dall'insieme $\{-2, -1, 0, 1, 2\}$. Calcola $\mathbf{C} = \mathbf{AB}$ e aggiungi e sottrai un'unità a due elementi di una stessa riga (C_{22} e C_{28} ad esempio). Implementa *MCMatrixMultiplicationVerifier* e lancialo 100 volte. Spiega i risultati che ottieni per $k = 2, 5$ e 10.

Confronto di file o stringhe binarie

Supponiamo che Bob voglia verificare che il file B in suo possesso sia uguale al file A di Alice, entrambi di ℓ bit. Nel caso di file grandi, la verifica può essere dispendiosa soprattutto per il tempo necessario per la trasmissione del file A a Bob (problema della sincronizzazione di copie di uno stesso file). Vediamo come un algoritmo *Monte Carlo* consenta di verificare l'uguaglianza di A e B trasmettendo solo $2 \ln \ell$ bit.

Ci serve un risultato fondamentale della teoria dei numeri che enunciamo senza dimostrazione.

Teorema 9.1. *Numerosità asintotica dei numeri primi*
Indichiamo con $\pi(x)$ il numero di primi minori o uguali a $x \in \mathbb{R}^+$ e con $\ln x$ il logaritmo naturale di x. Si ha che

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$$

□

La tabella di seguito mostra alcuni valori e chiarisce come la convergenza a 1 del rapporto al crescere di x non equivale alla convergenza del numeratore al denominatore

x	$\pi(x)$	$\pi(x) - x/\ln x$	$\pi(x)/(x/\ln x)$
10	4	-0.3	0.921
10^3	168	23	1.161
10^6	78,498	6116	1.084
10^{12}	37,607,912,018	1,416,705,193	1.039
10^{23}	1,925,320,391,606,818,000,000	37,083,513,766,592,670,000	1.020

Abbiamo bisogno, infine, anche di un risultato molto semplice.

Proposizione 9.2. *Maggiorazione del numero di fattori primi*
I fattori primi di un qualunque numero $n < 2^\ell$, per ogni $\ell \in \mathbb{N}$, sono al più ℓ.
Dimostrazione: tutti i numeri primi sono maggiori o uguali a 2. Se n avesse più di ℓ fattori primi, avremmo che $n > 2^\ell$. ■

Definizione 9.1. *Impronta digitale*
Sia p un numero primo compreso tra 2 e ℓ^2 ed $n \in \mathbb{N}$ con $n < 2^\ell$. L'*impronta digitale* $f(n)$ di n è definita come

$$f(n) := n \pmod p$$

□

L'idea di *MCStringEqualityVerifier* è che Alice legge il file A come il numero naturale a. Poiché il file A è formato da al più ℓ bit, abbiamo che $a \leq 2^\ell$. Poiché $p < \ell^2$, invece, $f(n)$ richiede solo $2\ln\ell$ bit. Alice, pertanto, calcola $f(a)$ e spedisce a Bob $2\ln\ell$ bit invece che ℓ bit. Bob, per parte sua, legge il file B come il numero naturale b e confronta $f(a)$ con $f(b)$.

Algoritmo 9.2. *MCStringEqualityVerifier*(a, b)
Input: a e b stringhe binarie di lunghezza ℓ
Output: `input diversi` o `input probabilmente uguali`

---

1. campiona p, primo, uniformemente a caso in $\{2, \ldots, \ell^2\}$
2. $f(a) \leftarrow a \pmod p$
3. $f(b) \leftarrow b \pmod p$
4. `if` $|f(a) - f(b)| \equiv 0 \pmod p$
   `return input probabilmente uguali`
   `else`
   `return input diversi`

Il campionamento del numero primo p potrebbe essere basato o su una lista pre-calcolata di numeri primi o sulla ripetizione del *MCMillerRabinTest* per determinare la primalità di un numero p campionato uniformemente in $\{2, \ldots, \ell^2\}$.

Valutiamo ora la probabilità d'errore per *MCStringEqualityVerifier*.

Proposizione 9.3. *MCStringEqualityVerifier è un algoritmo Monte Carlo molto efficiente*
Se $\mathcal{P}$ è la probabilità che l'algoritmo **9.2** restituisca `input probabilmente uguali` quando $a \neq b$, abbiamo

$$\mathcal{P} \approx \frac{2\ln\ell}{\ell}$$

Dimostrazione: Se $a \neq b$ allora $1 \leq |a-b| \leq 2^\ell$. Per la **Proposizione 9.2**, allora, $|a-b|$ ha al più ℓ fattori primi e, poiché dal **Teorema 9.1** sappiamo che i numeri primi tra 1 e ℓ^2 sono dell'ordine di $\ell^2/2\ln\ell$, per la probabilità di errore $\mathcal{P}$ abbiamo

$$\mathcal{P} \approx \frac{\ell}{\ell^2/2\ln\ell} = \frac{2\ln\ell}{\ell}$$

■

Osservazione 9.3. *Niente male per un Monte Carlo*
La probabilità di errore di *MCStringEqualityVerifier* nel caso di ℓ grandi è decisamente piccola. Se dovessimo confrontare file dell'ordine di Gbit, ovvero $\ell = 2^{30}$ bit, otterremmo

$$\mathcal{P}\left(2^{30}\right) \approx \frac{2\ln 2^{30}}{2^{30}} = \frac{60\ln 2}{2^{30}} < \frac{2^6}{2^{30}} = 2^{-24} < 10^{-7}$$

Compito 9.2. *File piccoli ma non troppo piccoli*
Considera due file diversi, a e b, di $\ell = 20$ bit. Campiona un numero primo p a caso tra 2 e $\ell^2 = 400$. Implementa *MCStringEqualityVerifier* e commenta il risultato che ottieni ripetendo il campionamento di p per 20 volte.

Appendice

MR-testimoni per i numeri di *Carmichael*

Dimostriamo dapprima che un numero di *Carmichael* non può essere una potenza di un numero primo p.

Un numero di Carmichael non può essere una potenza di un numero primo

Procediamo per assurdo: assumiamo quindi che n sia di *Carmichael* e possa scriversi come $n = p^\alpha$ con p primo e $\alpha \geq 2$.

Dall'ipotesi che n è di *Carmichael* segue che per ogni base $a \in \mathbb{Z}_n^*$

$$a^{n-1} \equiv 1 \pmod{n}$$

Per il **Teorema 8.2**, inoltre, vale la congruenza

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Ma per $n = p^\alpha$ si ha $n - 1 = p^\alpha - 1$ e $\phi(n) = p^{\alpha-1}(p-1)$. Ora, poiché p è coprimo di $p^\alpha - 1$ abbiamo

$$\mathrm{MCD}(p^{\alpha-1}(p-1), p^\alpha - 1) = p - 1$$

e pertanto

$$a^{p-1} \equiv 1 \pmod{n}$$

ovvero l'ordine g_a di un qualunque $a \in \mathbb{Z}_n^*$ divide $p - 1$ (e *non può quindi essere divisibile per* p).

Se peró consideriamo $a = p + 1$, ponendo $p^{\alpha-1} = k$, ricaviamo

$$\begin{aligned}(1+p)^k &= 1 + \binom{k}{1}p + \binom{k}{2}p^2 + \cdots + \binom{k}{k-1}p^{k-1} + p^k \\ &= 1 + kp + \frac{k(k-1)}{2}p^2 + \cdots + kp^{k-1} + p^k \\ &= 1 + p^\alpha + \frac{p^\alpha(p^{\alpha-1}-1)}{2}p + \cdots + p^\alpha p^{p^{\alpha-1}-2} + p^\alpha p^{p^{\alpha-1}-\alpha}\end{aligned}$$

Poiché tutti i termini della somma, a parte il primo, sono multipli di p^α otteniamo

$$(p+1)^{p^{\alpha-1}} \equiv 1 + 0 + 0 + \cdots + 0 + 0 \equiv 1 \pmod{p^\alpha}$$

L'ordine della base $a = p + 1$, pertanto, *è divisibile per* p, in contraddizione col fatto che l'ordine di nessuna base in $\mathbb{Z}_n^*$ poteva esserlo. In conclusione se n è di *Carmichael* non può essere una potenza di un numero primo.

Prodotto di due numeri dispari coprimi tra loro

Se n è di *Carmichael*, pertanto, si deve poter scrivere come prodotto di due numeri dispari coprimi, ovvero

$$n = cd \quad \text{con} \quad \mathrm{MCD}(c,d) = 1 \quad \text{e} \quad c, d \geq 3$$

Premettiamo un ultimo risultato, versione particolare del Teorema Cinese del Resto (pag. 779 del *Kormen et al.* [9]).

Teorema 1. *Una questione di resti (per 2 soli fattori)*
Se c e d sono coprimi, allora le equazioni

$$x \equiv p \pmod{c} \quad \text{and} \quad x \equiv q \pmod{d}$$

hanno un'unica soluzione $x^* \in \mathbb{Z}_{cd}^+$, sistema completo dei residui $\pmod{cd}$.
Dimostrazione: sia c_d^{-1} l'inverso di c in $\mathbb{Z}_d^*$ e con d_c^{-1} l'inverso di d in $\mathbb{Z}_c^*$. L'esistenza di c_d^{-1} in $\mathbb{Z}_d^*$ e di d_c^{-1} in $\mathbb{Z}_c^*$ sono garantite dalla coprimalità di c e d. Chiaramente, abbiamo

$$cc_d^{-1} \equiv 0 \pmod{c}, \quad dd_c^{-1} \equiv 0 \pmod{d}, \quad pdd_c^{-1} \equiv p \pmod{c} \quad \text{e} \quad qcc_d^{-1} \equiv q \pmod{d}$$

Pertanto

$$x^* = pdd_c^{-1} + qcc_d^{-1}$$

è soluzione delle due congruenze in quanto

$$\begin{aligned} pdd_c^{-1} + qcc_d^{-1} &\equiv p \pmod{c} \\ pdd_c^{-1} + qcc_d^{-1} &\equiv q \pmod{d} \end{aligned}$$

Inoltre, x^* in $\mathbb{Z}^+_{cd}$ è unica: infatti se y^* è un'altra soluzione, abbiamo necessariamente $x^* \equiv y^* \pmod{c}$ e $x^* \equiv y^* \pmod{d}$. Poiché $\mathrm{MCD}(c,d) = 1$, abbiamo allora che $x^* \equiv y^* \pmod{cd}$. ■

Esempio 1. *Per chi non crede ai teoremi*
Supponiamo di voler risolvere le equazioni

$$x \equiv 3 \pmod{14} \quad \text{e} \quad x \equiv 4 \pmod{9}$$

Osserviamo preliminarmente che $\mathrm{MCD}(14,9) = 1$ e $14 \cdot 9 = 126$. Abbiamo che l'inverso di 14 in $\mathbb{Z}^*_9$ è 2, poiché $14 \cdot 2 = 3 \cdot 9 + 1 \equiv 1 \pmod{9}$. Similmente, l'inverso di 9 in $\mathbb{Z}^*_{14}$ è 11, poiché $9 \cdot 11 = 14 \cdot 7 + 1$. Di conseguenza

$$x^* = 3 \cdot 9 \cdot 11 + 4 \cdot 14 \cdot 2 = 297 + 112 = 409 \equiv 31 \pmod{126}$$

è la soluzione cercata. Infatti

$$31 \equiv 3 \pmod{14} \quad \text{e} \quad 31 \equiv 4 \pmod{9}$$

Osservazione 1. *Casi particolari*
Dal fatto che

$$x^* \equiv 1 \pmod{cd} \;\rightarrow\; x^* \equiv 1 \pmod{c} \text{ e } x^* \equiv 1 \pmod{d}$$

segue che

$$x^* \equiv 1 \pmod{c} \text{ e } x^* \not\equiv 1 \pmod{d} \;\rightarrow\; x^* \not\equiv 1 \pmod{cd}$$

Analogamente, dal fatto che

$$x^* \equiv -1 \pmod{cd} \;\rightarrow\; x^* \equiv -1 \pmod{c} \text{ e } x^* \equiv -1 \pmod{d}$$

segue che

$$x^* \equiv -1 \pmod{c} \text{ e } x^* \not\equiv -1 \pmod{d} \;\rightarrow\; x^* \not\equiv -1 \pmod{cd}$$

■

Siamo ora in grado di dimostrare che se n è di *Carmichael* ed $n = cd$ con $\mathrm{MCD}(c,d) = 1$ e $c, d \geq 3$, gli MR-bugiardi di *MCPrimalityTest* per n appartengono a un sottogruppo proprio di $\mathbb{Z}^*_n$.

Poniamo $n - 1 = q2^s$ con q dispari e $s \geq 1$. Per ogni base a, scriviamo la sequenza di $s+1$ quadrature a^{q2^t} per $t = 0, \ldots, s$, ovvero $\{a^q, a^{q2}, \ldots, a^{q2^s}\}$. Poiché n è di Carmichael sicuramente $a^{q2^s} \equiv 1 \pmod{n}$ per tutti gli a (MR-bugiardi o meno). Per qualche base a e $t < s$ avremo certamente $a^{q2^t} \equiv -1 \pmod{n}$. Per esempio, per $a = n-1$ e $t = 0$ abbiamo $(n-1)^q \equiv -1 \pmod{n}$. Tra tutte le coppie (a,t) per le quali $a^{q2^t} \equiv -1 \pmod{n}$, scegliamo la coppia $(\bar{a}, \bar{t})$ costituita da $\bar{t}$, valore massimale di t, e dalla corrispondente base $\bar{a}$. Osserviamo che l'insieme

$$B = \{a \in \mathbb{Z}^*_n \mid a^{q2^{\bar{t}}} \equiv \pm 1 \mod n\}$$

è un sottogruppo di $\mathbb{Z}^*_n$. Il valore massimale $\bar{t}$ garantisce che tutti gli MR-bugiardi appartengono a B. Mostriamo ora l'esistenza di un MR-testimone $\hat{a}$ che appartiene a $\mathbb{Z}^*_n \setminus B$. Abbiamo che

$$\bar{a}^{q2^{\bar{t}}} \equiv -1 \pmod{n} \quad \rightarrow \quad \bar{a}^{q2^{\bar{t}}} \equiv -1 \pmod{c}$$

Consideriamo ora la soluzione $a = \hat{a}$ delle due congruenze

$$a \equiv \bar{a} \pmod{c} \quad \text{e} \quad a \equiv 1 \pmod{d}$$

che otteniamo dal **Teorema 1**. Elevando $\hat{a}$ alla potenza $q2^{\bar{t}}$ otteniamo

$$\hat{a}^{q2^{\bar{t}}} \equiv -1 \pmod{c} \quad \text{e} \quad \hat{a}^{q2^{\bar{t}}} \equiv 1 \pmod{d}$$

Per l'**Osservazione 1** abbiamo allora che

$$\hat{a}^{q2^{\bar{t}}} \not\equiv \pm 1 \pmod{n}$$

e, quindi, che $\hat{a} \in \mathbb{Z}_n^* \setminus B$.

Nuovamente per il **Teorema 8.1** avremo che

$$|B| \leq \frac{|\mathbb{Z}_n^*|}{2} \leq \frac{n-1}{2}$$

Similmente a prima, pertanto, il numero di MR-bugiardi è certamente minore di $(n-1)/2$, ovvero la frazione dei MR-testimoni è maggiore di 1/2.

Temi e domande d'esame

0. **Ordinamento**
   (a) Costruisci l'input peggiore per una versione deterministica di *QuickSort* che sceglie come *pivot (a)* il primo elemento di una sequenza e *(b)* l'elemento mediano
   (b) Supponi di avere una probabilità congiunta di 100 variabili casuali di Bernoulli non indipendenti. Se devi calcolare il valore atteso della somma partendo da questa distribuzione quanti sono i termini da sommare? Quanti invece se utilizzi il fatto che il valore atteso di una somma è uguale alla somma dei valori attesi? Che cosa cambia se fossero indipendenti?
   (c) Nell'assunzione che la sequenza S consista dei primi n numeri naturali, spiega per quale motivo la probabilità che i sia confrontato con j può essere espressa come

$$\frac{2}{|i-j|+1}$$

1. **Programmazione lineare**
   (a) Discuti che tipo di problemi possono essere impostati come problemi di programmazione lineare, che cosa è la funzione obiettivo e qual è il ruolo dei vincoli e della regione ammissibile.
   (b) Quale sarebbe l'ottimo della funzione obiettivo in assenza di vincoli?
   (c) Per quale motivo, sotto assunzioni piuttosto blande, l'ottimo di un problema di programmazione lineare si ottiene su un vertice della regione ammissibile?

2. **Teoria dei giochi**
   (a) Determina il migliore degli scenari possibili per Roberta e Carlo data una matrice $\mathbf{M}$ dei pagamenti in un gioco a somma zero
   (b) Che cosa sono le strategie miste e, in particolari, le strategie miste ottimali?
   (c) Che cosa puoi fare per ottenere un limite inferiore al costo computazionale atteso di un algoritmo randomizzato?

3. **Valutazione dell'albero di un gioco**
   (a) Produci un esempio di un albero di un gioco per il quale un algoritmo deterministico deve valutare tutte le foglie.
   (b) Dimostra che nel caso base di un albero binario di un gioco un algoritmo randomizzato controlla, in media, 3 delle 4 foglie

4. **Taglio minimo**
   (a) Spiega il motivo per il quale se k è la cardinalità di un taglio minimo per un grafo G con n vertici, il numero di archi non può essere minore di $nk/2$.
   (b) Sia G un grafo connesso di n vertici e taglio minimo uguale a k. Se E_i è l'evento di non selezionare un arco di S all'i-esima iterazione di *MCMinCut*, spiega il motivo per cui

$$Pr(E_i|E_1, E_2, \ldots, E_{i-1}) \geq \frac{2}{n-i+1}$$

   (c) Se un algoritmo *Monte Carlo* ottiene il risultato corretto con probabilità p, quante volte deve essere ripetuto per restituire il risultato corretto almeno una volta con probabilità del 99.9%?

5. **Accordo bizantino**
   (a) Per quali motivi raggiungere il consenso distribuito è un problema complicato?
   (b) Quali sono le specifiche per il raggiungimento del consenso bizantino e quali le motivazioni per i vincoli di *consenso* e *validità*?
   (c) Spiega il motivo per cui se $n = 3f$ e $T = 2f$ il protocollo *MGByzantineGeneral* potrebbe non raggiungere mai un accordo

(d) Siano dati $n = 3f+1$ processi, f dei quali inaffidabili. Per quale motivo, se n è abbastanza grande e il bit di ogni processo affidabile è inizializzato uniformemente a caso, i $2f+1$ processi affidabili raggiungono l'accordo dopo il primo *round* con grande probabilità?

6. **Test di primalità**

   (a) Verifica che gli elementi di $\mathbb{Z}_{10}^{+}$ che non sono coprimi di 10 non ammettono inverso e quindi non possono appartenere a $\mathbb{Z}_{10}^{*}$.

   (b) Determina gli elementi di $\mathbb{Z}_{12}^{*}$ e verifica che ogni elemento è l'inverso di se stesso.

   (c) Eseguendo a mano i passi di *MCPrimalityTest* verifica che per ogni $a \in \{2, \ldots, 5\}$, 7 è `probabilmente primo`.

7. **Verifiche di uguaglianza**

   (a) Per quale motivo verificare il prodotto di due matrici $n \times n$ richiede solo $O(n^2)$ moltiplicazioni?

   (b) Come costruiresti un insieme di numeri primi minori di ℓ^2?

   (c) Poni $\ell = 8$ e costruisci due *file* a e b di 8 bit con $a \neq b$. Campiona un numero primo a caso tra 2 e 64 e confronta le due *fingerprint* ottenute. Spiega quale strategia consentirebbe di determinare tutti i numeri primi per i quali le *fingerprint* sono uguali.

Riferimenti bibliografici

[1] R. Motwani & P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[2] D. Lehman & M. O. Rabin. *On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem*. Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 133-138 , 1981.

[3] S. Ross. *A First Course on Probability*. Prentice Hall, 2010.

[4] R.G. Seidel. *Small-dimensional linear programming and convex hulls made easy*. Discrete and Computational Geometry, 6: 423-434, 1991.

[5] M. Stoer & F. Wagner. *A simple min-cut algorithm*. Journal of the ACM, 44(4): 585–591, 1997.

[6] L. Lamport, R. Shostak & M. Pease. *The Byzantine Generals Problem*. ACM Transactions on Programming Languages and Systems, 4(3): 382-401, 1982.

[7] M.O. Rabin. *Randomized Byzantine Generals* In Proceedings of the 24th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1983.

[8] T.M. Apostol. *Introduction to Analytic Number Theory*. Springer, 1976.

[9] T.H. Kormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[10] G. Miller. *Riemann's Hypothesis and Tests for Primality*. Journal of Computer and System Sciences, 13 (3): 300–317, 1976.

[11] M.O. Rabin. *Probabilistic algorithm for testing primality*. Journal of Number Theory, 12 (1): 128–138, 1980.

[12] M. Agrawal, N. Kayal & N. Saxena. *Primes is in P*. Annals of Mathematics. 160 (2): 781–793, 2004.