Piccola rassegna di altri quesiti e task proposti gli scorsi anni nelle gare Bebras https://bebras.it e Kangourou https://www.kangourou.it

- Interpretazione / composizione di comandi
- Shift con rotazione
- Codici (ambigui)
- Successioni periodiche
- Automi a stati finiti (deterministici)
- Automi cellulari
- Ricerca di stringhe
- Project scheduling
- Geometria Manhattan
- Calcolo di percorsi

Old computing machine (Belgio, 2013)

Beaver Adrien found an old computing machine in his attic. He wants to play with it and try to add two numbers. The computing machine have three registers (memory area which holds a number), named R1, R2 and R3. To program the machine, a sequence of operations has to be encoded on it.

Here are the possible operations with their effect (*i* and *j* are register numbers and *q* is the number of an operation):

Zero (i)	Puts 0 in register Ri
Inc (i)	Adds 1 to the value of register Ri
Dec (i)	Removes 1 to the value of register Ri
Store (i, j)	Copies the value of register *j* in register Ri
Jump (i, j, q)	If registers Ri and Rj have the same value, jumps to the *q*th operation
JumpNeg (i, j, q)	If registers Ri and Rj have different values, jumps to the *q*th operation

If Beaver Adrien places the two numbers that he wants to add in registers R1 and R2, which of the following program adds the two numbers together, and put the final result in R1?

A) 1: Zero(3) 2: JumpNeg(1,3,5) 3: Inc(2) 4: Dec(1) 5: Jump(1,3,3) 6: Store(1,2)	C) 1: Zero(3) 2: JumpNeg(1,3,5) 3: Inc(1) 4: Dec(2) 5: Jump(1,3,3) 6 Store(1,2)
B) 1: Zero(3) 2: Jump(1,3,5) 3: Inc(2) 4: Dec(1) 5: JumpNeg(1,3,3) 6: Store(1,2)	D) 1: Zero(3) 2: Jump(1,3,5) 3: Inc(1) 4: Dec(2) 5: JumpNeg(1,3,3) 6: Store(1,2)

The correct answer is B.

Drawing robot

(Francia, 2013)

Beaver has bought a new robot that will paint the floors of his house.
Beaver can write a program that controls the movement of the robot.

Here are some example programs for this robot:

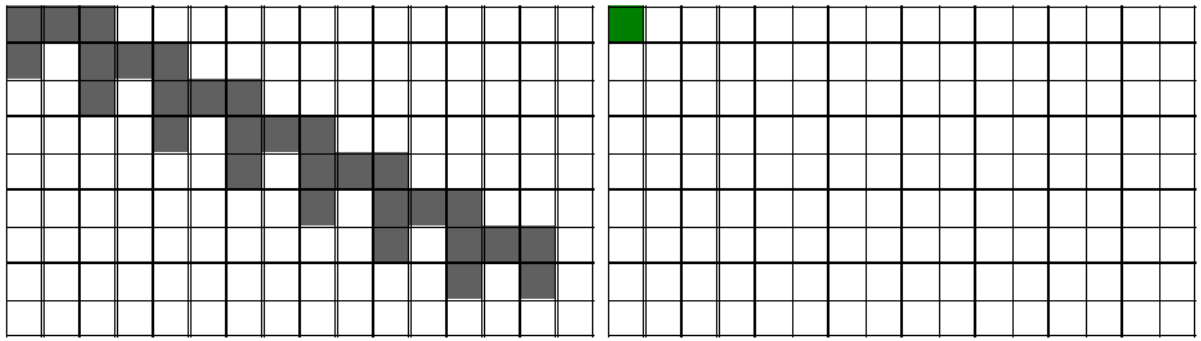
Program	Effect
1S	move 1 step South
3E 1W	move 3 steps East then 1 step West
3S 2E 1N	move 3 steps South, then 2 steps East and 1 step North
4(3S 2E 1N)	do the previous program 4 times

Beaver has seen the pattern of the floor of a friend of his. He wants to have the same pattern on his floor.

Help beaver write a program to copy this drawing.

You may try as many programs as you like, until you manage to draw exactly the figure given on the left. Your program cannot be longer than 50 characters.

Enter your program : Execute

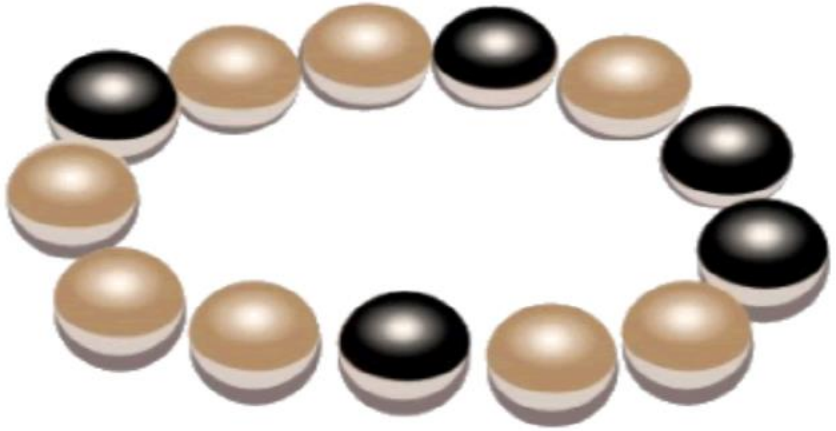


Solution

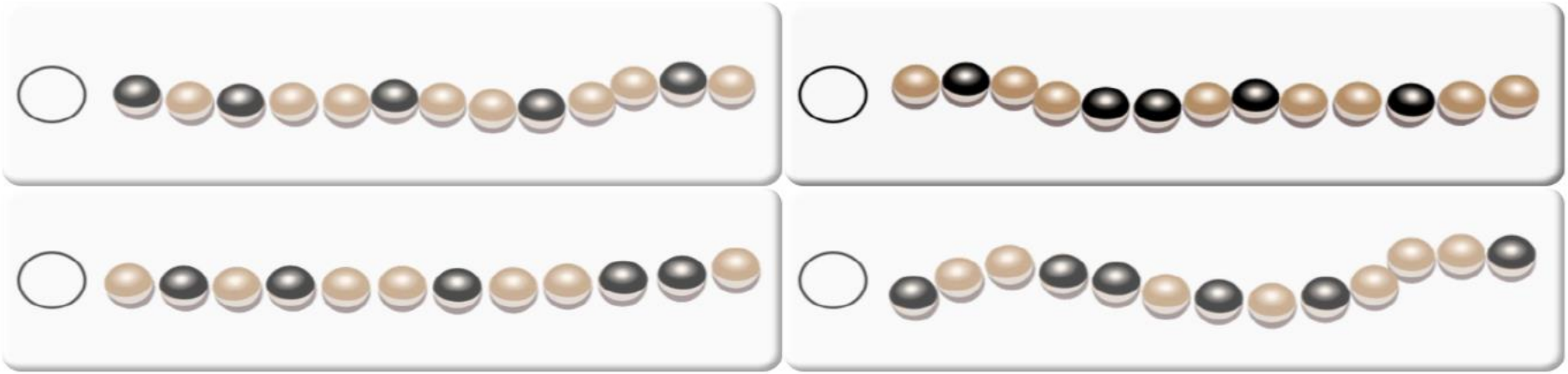
7(1S 1N 2E 1S)

Il braccialetto magnetico (Repubblica Ceca, 2014)

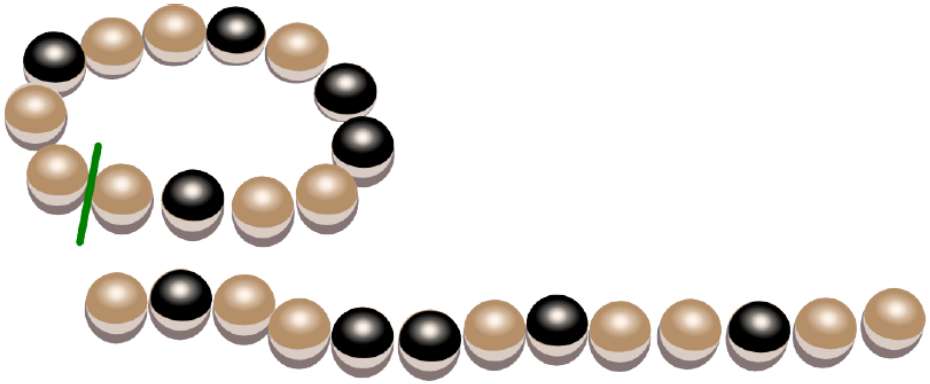
Tra i castori sono di gran moda braccialetti magnetici fatti di palline chiare e scure, che si possono agganciare e sganciare a piacere. La castorina Bea ne possiede 4 che ripone sempre ben allineati in un bauletto. Ieri sera indossava il braccialetto raffigurato qui a destra, che è stato molto ammirato, e quindi stasera vuole rimetterlo.



Quale tra questi è il braccialetto che Bea indossava ieri?



Soluzione.
La soluzione è quella in alto a destra.



Approfondimenti. Questo problema considera una sequenza di oggetti (palline chiare e scure, rappresentabili con le cifre binarie, o *bit*), chiusa a formare una struttura circolare (il braccialetto); si deve stabilire se, aprendo la struttura in un punto opportuno, si può ottenere una certa sequenza (uno dei braccialetti aperti). Immaginiamo di aprire la struttura circolare in un punto arbitrario e di disporre le palline in una fila. Controlliamo quindi se tale fila è uguale a una di quelle date o a un loro ribaltamento (il braccialetto aperto può essere infatti rigirato da destra a sinistra!); poi prendiamo l'ultima pallina della fila e la mettiamo all'inizio della fila (in informatica, questa operazione fatta su una sequenza di bit si chiama *shift con rotazione*), e ripetiamo il controllo... Alla peggio, faremo tanti passi quante sono le palline.

Parole chiave: Rappresentazione dell'informazione, *shift con rotazione*.

Un codice ambiguo (3 punti) (Canada, 2013)

Berto Canguro vuole mandare un messaggio ad Anna, ma può usare solo i tasti 0 e 1 perché gli altri sono rotti. Berto non è troppo ferrato nella teoria dei codici e decide di usare le seguenti corrispondenze per le prime 5 lettere dell'alfabeto:

```
A -> 0110
B -> 01
C -> 110
D -> 1
E -> 0
```

Berto codifica il suo messaggio e ottiene 01101101. Anna è perplessa perché non sa come interpretare il codice ricevuto.
Quale dei seguenti non può sicuramente essere il messaggio codificato da Berto?

○	ACD	○	BDECD
○	BDCD	○	EDDAD

L'unico messaggio che non può essere stato codificato in 01101101 è BDCD. Per convincersene basta controllare (vedi tabella qui sotto) le codifiche dei messaggi indicati come possibili. Si può anche notare che la codifica di BDCD e piú breve (7 bit) di quella indicata (8 bit).

Messaggio	Codifica
ACD	01101101
BDECD	01101101
BDCD	0111101
EDDAD	01101101

In generale, volendo elencare tutte le possibili interpretazioni del messaggio in codice 01101101, si deve procedere per esaurimento; il messaggio in chiaro può iniziare per A o B o E:

- Se inizia per A, può continuare con C o con D: dopo AC può stare D, quindi una interpretazione è ACD; dopo AD può stare un'altra D, seguita da B o da ED, quindi altre due interpretazioni sono ADDB e ADDED.
- Se inizia per B, deve continuare con D; rimane 01101, che può essere interpretato come AD o BDB o BDED o ECD o EDDB o EDDED, quindi si aggiungono altre sei interpretazioni.
- Infine, se inizia per E, può continuare con C o con D; nel primo caso rimane 1101 (= CD o DDB o DDED, v. punto precedente), nel secondo caso deve continuare con D e rimane 01101 come al punto precedente; vi sono dunque ulteriori nove possibili interpretazioni: in tutto ben 18!

Conviene forse partire dal fondo? Una condizione sufficiente per avere un codice non ambiguo è che nessuna lettera codificata sia prefisso di un'altra; conviene poi codificare le lettere più frequenti con le sequenze di bit più corte (D. A. Huffman, 1952).

Quesito 8 – Una successione periodica

Definiamo una successione (di numeri naturali) $\{u_n\}$, con $n \in \mathbb{N}$, nel seguente modo:

$$u_0 = u_1 = 4, \quad u_2 = 3, \quad u_{n+3} = (5u_{n+2} - 3u_{n+1} + 2u_n) \; mod \; 5,$$

dove *mod* (%, in Python) è il nome dell'operazione che dà il resto della divisione intera.
Da un certo u_p in poi, la sequenza di valori si ripete; più precisamente, il periodo p è il più piccolo numero intero positivo tale che, per ogni $n \in \mathbb{N}$, $u_{n+p} = u_n$.
Qual è il periodo della successione data?

La risposta giusta, 124, può essere trovata eseguendo questo programma in Python, che termina non appena ritrova i tre valori iniziali in sequenza:

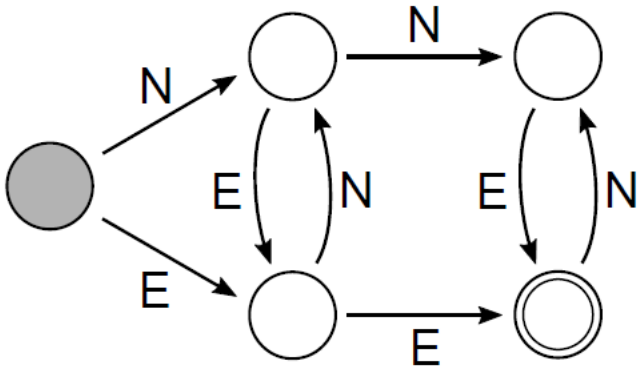
```
a = 4
b = 4
c = 3
print 0, a
print 1, b
print 2, c
for k in range(3, 200):
    d = (5*c - 3*b + 2*a)%5
    print k, d
    a = b
    b = c
    c = d
    if a == 4 and b == 4 and c == 3: break
```

L'ultimo valore stampato è $u_{126} = 3$.

Robot paths

(Italia, 2017)

[…] The robot is now directed by the diagram in the figure below. Complete the following statements choosing the correct completion.



initial state

eventually final state

1. Any allowed path
   - must end with E
   - only sometimes ends with E
   - cannot end with E
2. No allowed path contains
   - ENNE
   - ENEENE
   - NEENENNE
3. In any allowed path the difference between the number of E and the number of N is
   - equal to 2
   - equal to 3
   - at most 2

One can see that any allowed path has the following properties:

- it starts with either E or N (since from the filled circle two arrows start marked with E and N respectively);
- it ends with E (since the double circle is reached only by arrows marked by E);
- it alternates E and N except once;
- it necessarily contains a pair of consecutive equal steps (either EE or NN).

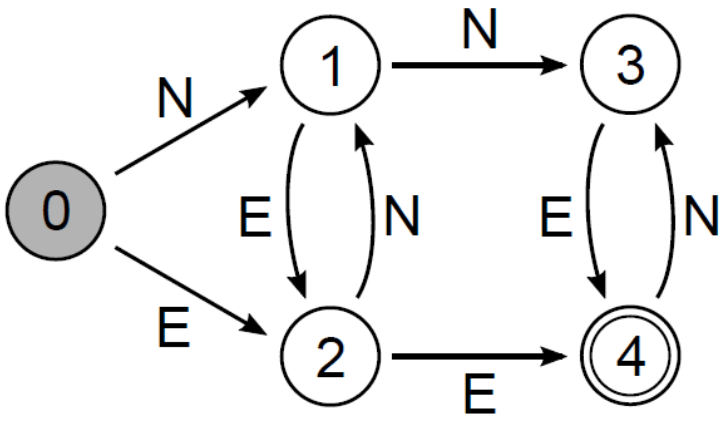
Thus the correct answers are the following.

1. Any allowed path **must end with E**.
2. No allowed path contains **NEENENNE**, since after executing two consecutive arrows marked with E, the double circle is reached and, from then on, E and N must always alternate.
3. In any allowed path the difference between the number of E and the number of N is **at most 2**: this is implied by the above properties, since in the most unbalanced case, the path starts and ends with E and contains EE, so it is of the form E...NEEN....E where dots stand for some repetitions (possibly none) of NE.

The diagrams presented in the tasks are the *state diagrams* of *finite-state automata* (FSA, aka finite-state machine, finite automaton, or state machine). FSA are used in informatics to describe systems that can be in a finite set of possible states and move from one state to another by means of a finite set of actions (the next state depends only on the current state and the executed action). Usually, the *initial state* and the *final states* of the system are also specified.

In the state diagram of a FSA, possible states are represented by circles (in particular, in the task the initial state is the filled circle and the final one is the double circle), transition of states by arrows, and actions by letters from any given alphabet (N and E in the task).

States can be numbered or named to distinguish them, as in the following figure for the FSA of the task:



In other words, the robot system can be in 5 possible states:

0. the robot has not executed any move yet;
1. the robot has executed a sequence of alternating N and E steps starting with N;
2. the robot has executed a sequence of alternating E and N steps starting with E;
3. the robot has executed exactly once two consecutive N steps, alterating N and E steps in the remaining part of its path;
4. the robot has executed exactly once two consecutive E steps, alternating N and E steps in the remaining part of its path;

The *behaviour* of the system is described by the *language accepted* by a FSA, i.e., the set of sequences that can be obtained by following the arrows in its diagram. For instance, the language accepted by the FSA in the figure describes all and only the allowed paths the robot may travel.

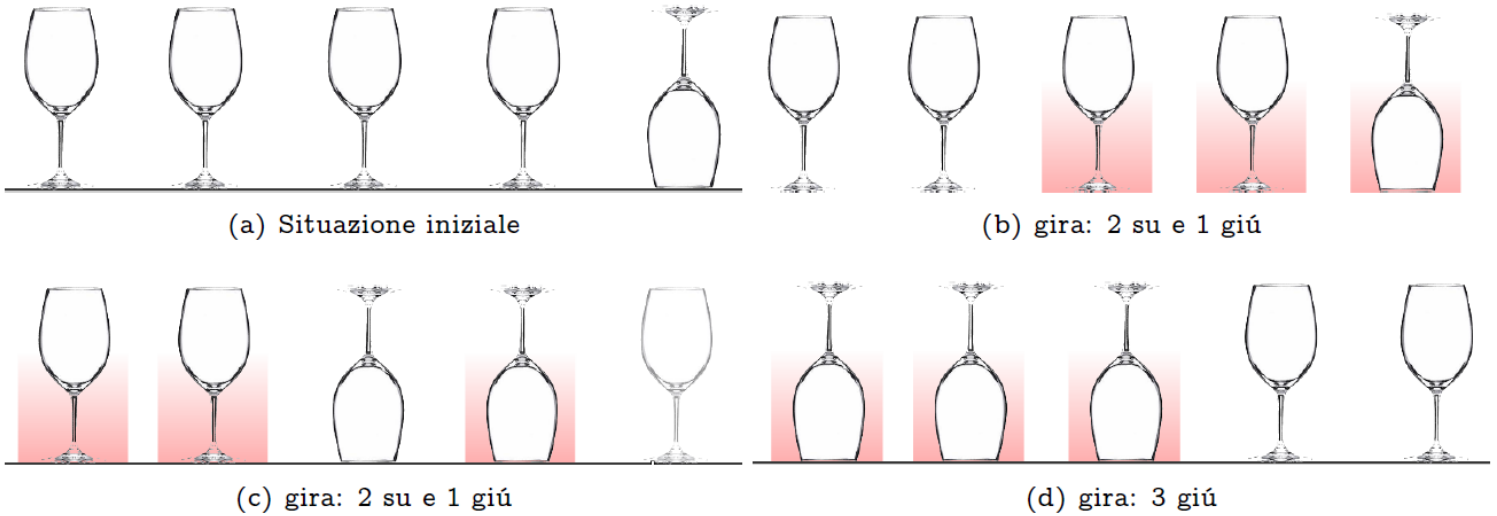
Bicchieri (max 5 punti) (Ucraina, 2012)

Sul tavolo ci sono 5 bicchieri vuoti. Uno dei bicchieri è girato a testa in giù e gli altri sono dritti. In una mossa, potete capovolgere esattamente tre diversi bicchieri (da testa in giù a dritti, o viceversa).

Qual è il numero minimo di mosse richieste per raddrizzare tutti i bicchieri?

Soluzione.

I bicchieri possono essere raddrizzati con un numero minimo di tre mosse, come mostra la Figura 5.2.



(a) Situazione iniziale

(b) gira: 2 su e 1 giú

(c) gira: 2 su e 1 giú

(d) gira: 3 giú

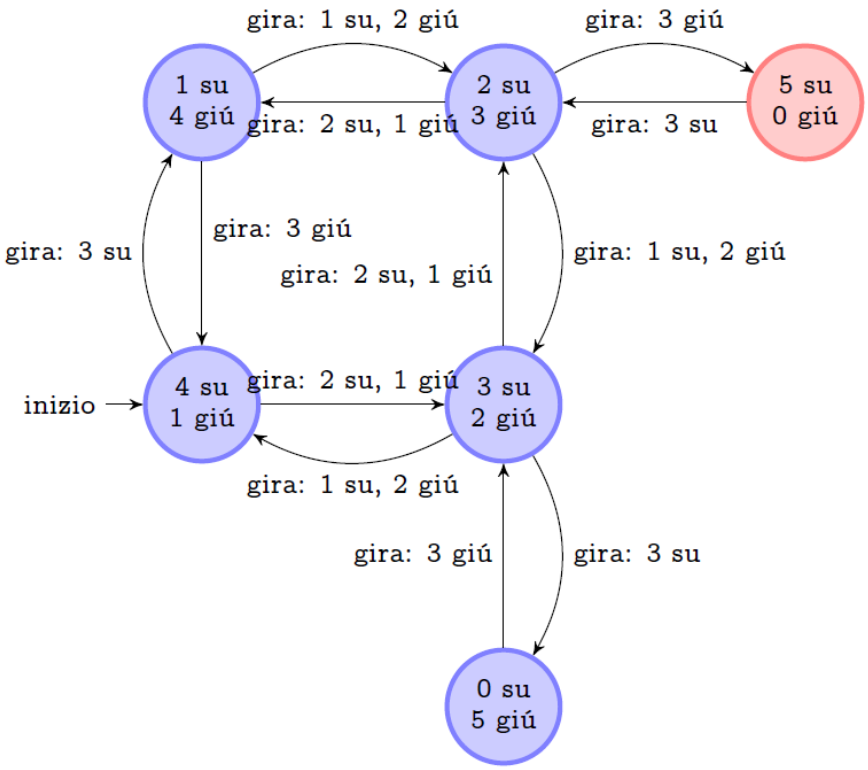
Figura 5.2: Mosse necessarie alla soluzione

Si osservi che il numero di mosse deve essere dispari. Infatti dopo la prima mossa i bicchieri a testa in giú saranno due o quattro; dopo la seconda mossa saranno un numero dispari (1,3,5). Ci vorranno quindi piú di due mosse e in generale un numero dispari di mosse per raddrizzare tutti i bicchieri. Abbiamo mostrato una soluzione con tre mosse, che deve quindi essere minima.

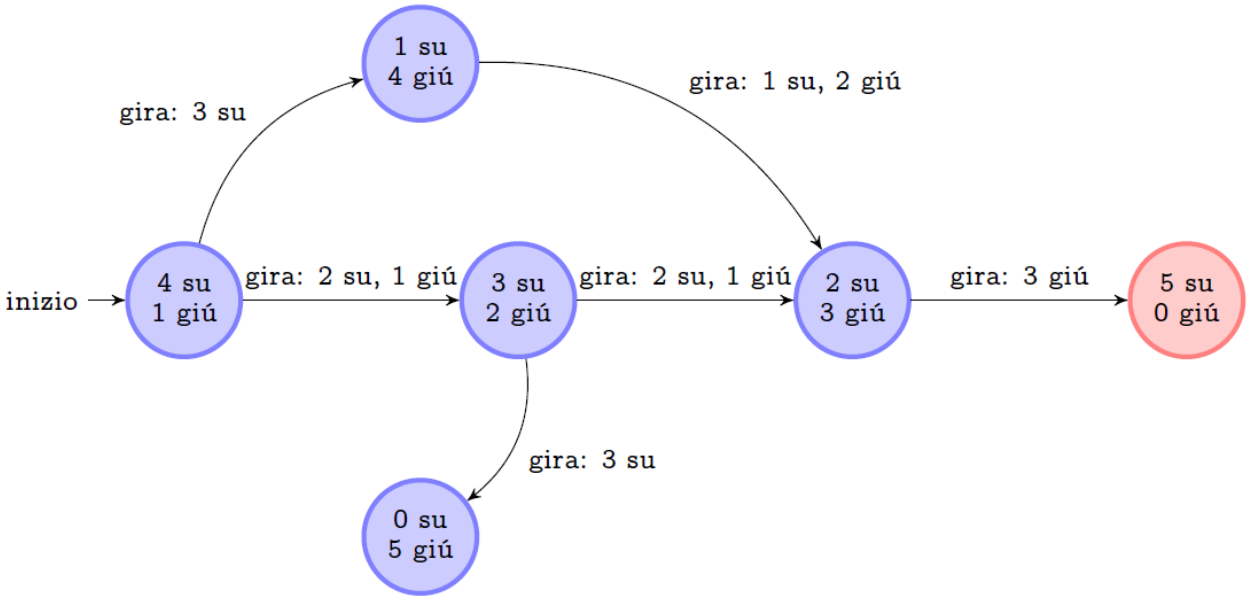
Anche questa è informatica! Possiamo visualizzare le trasformazioni come cambiamenti di stato in un *automa a stati finiti, DFA (Deterministic Finite Automaton).*

Per capire cosa succede nella figura a fianco:

- partite dal cerchio contrassegnato con la freccia "inizio",
- spostatevi da un cerchio a un altro in base alle parole sopra la freccia che li collega: quando la freccia dice "gira: X in su, Y in giú", vuol dire che X dei bicchieri che sono diritti devono essere capovolti, e Y dei bicchieri che sono a testa in giú devono essere raddrizzati;
- l'obiettivo è arrivare nello stato in rosso (lo stato contrassegnato "5 su, 0 giú").



In alternativa, poiché stiamo cercando il minimo numero di mosse, possiamo costruire un automa piú semplice in cui aggiungiamo un cerchio solo se non l'abbiamo già incontrato, come nella figura sotto.



In questa figura è facile vedere che il cammino minimo dal cerchio iniziale a quello finale è lungo tre passi e che di tali cammini ce ne sono due.

Eseguire un algoritmo, tenere traccia dello stato del "sistema" o delle "variabili", ragionare sulla parità e argomentare sulla correttezza di un algoritmo sono aspetti importanti dell'informatica.

Parole chiave e riferimenti: algoritmo, stato del sistema, parità, correttezza, automi a stati finiti

Un'insegna a LED

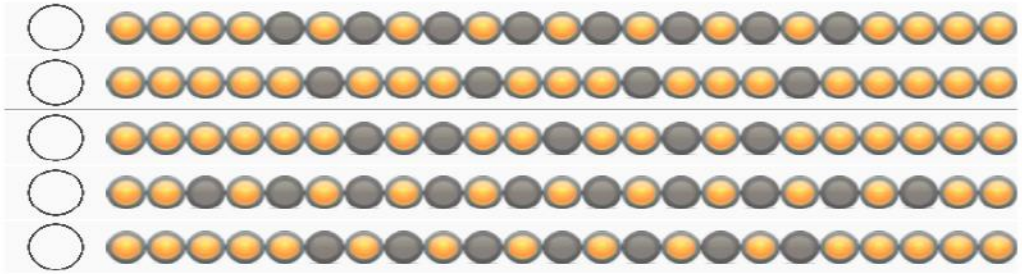
(Italia, 2014)

Un'insegna a LED è composta da una lunga linea di LED, ognuno dei quali può essere acceso oppure spento. All'insegna è collegato un pulsante, la cui pressione modifica lo stato di tutti i LED come indicato dalle regole seguenti:

- se un LED è spento, il LED viene acceso
- se un LED è acceso e i due LED adiacenti sono accesi, il LED rimane acceso
- se un LED è acceso e i due LED adiacenti sono spenti, il LED rimane acceso
- in tutti gli altri casi, il LED viene spento

Stamattina l'insegna aveva tutti i LED accesi tranne quello centrale, spento.

Quale figura rappresenta correttamente l'insegna dopo che il pulsante è stato premuto per sette volte?



Supponendo che l'insegna abbia almeno 500 LED, di cui soltanto uno al centro spento, immaginiamo di aver premuto il pulsante per 100 volte. Quali tra le seguenti affermazioni sono vere e quali false?

V F Non ci saranno due LED vicini entrambi spenti

V F Il numero di LED spenti sarà dispari

V F Ci sarà almeno un LED che da acceso è diventato spento l'ultima volta che ho premuto il pulsante

Soluzione.
Lo stato dell'insegna corretto è il primo dall'alto. L'unica falsa è la seconda affermazione.

Approfondimenti. Quello proposto è uno dei 256 *automi cellulari* (a tempo discreto) *elementari:* essi sono unidimensionali, cioè costituiti da una fila di celle, infinitamente lunga; due soltanto sono gli stati possibili per una cella (0 = acceso, 1 = spento), e lo stato di una cella al tempo $i+1$ è stabilito dallo stato della cella stessa e delle due adiacenti (una a destra e una a sinistra) al tempo i. Adottando la notazione di Stephen Wolfram, che ha studiato le sorprendenti proprietà di questi oggetti, il nome dell'automa elementare (o dell'insieme di regole, o piú semplicemente della *regola*, che lo governa) è quel numero decimale che, espresso in notazione binaria usando 8 cifre, fornisce la sequenza di regole di evoluzione nell'ordine discendente da 111 a 000. Ad esempio, l'automa (o regola) 18 (che in base 2 con 8 bit è 00010010, ed è proprio quello proposto nel quesito) corrisponde alla seguente tabella:

configurazione di tre celle adiacenti	111	110	101	100	011	010	001	000
nuovo stato per la cella centrale	0	0	0	1	0	0	1	0

Applicandolo alla configurazione iniziale con una sola cella "centrale" spenta e procedendo per parecchi passi, disegnando man mano ciascuna configurazione successiva sotto la precedente, risulterà chiaro il suo legame con il famoso *triangolo di Sierpinski*, o — se preferite — con la dislocazione dei numeri dispari nel *triangolo di Tartaglia.* In particolare, dopo un numero di passi dato da una potenza di 2 meno uno, si presenterà un'unica sequenza alternata di celle, una spenta una accesa. (Ci sono altre sette regole che producono la stessa identica successione della "regola 18": sapreste trovarle?)

Del piú celebre automa cellulare (bidimensionale), *Life*, ideato da John H. Conway verso la fine degli anni '60, parlammo già nel libretto del 2010, alla soluzione del quesito "Salvaschermo lampeggiante". Studi sistematici e approfonditi sugli automi cellulari sono stati compiuti da Stephen Wolfram a partire dagli anni '80. Uno dei suoi primi lavori, in cui ne propose già una classificazione, fu discusso in un seminario tenuto nel 1983 al Los Alamos National Laboratory (l'originale si trova in rete: `http://library.lanl.gov/cgi-bin/getfile?09-01.pdf`), mentre una gigantesca raccolta di risultati e scoperte è presentata nel suo libro *A New Kind of Science* (Wolfram Media, 2002). In rete, partendo dal sito web `http://www.stephenwolfram.com/`, si trova parecchio di questo materiale; inoltre, il sito `http://tones.wolfram.com/generate/` offre l'opportunità di trasformare una sequenza di passi di un automa in note musicali, generando cosí delle melodie. Consigliamo anche il ricchissimo sito "CelLab", assai ben curato da Rudy Rucker e John Walker (`http://www.fourmilab.ch/cellab/`), contenente storia, teoria e applicazioni a non finire con le quali ci si può divertire, ottenendo spesso risultati strabilianti.

Ricordiamo infine che certi *automi cellulari* rivestono pure un particolare interesse per l'informatica teorica, poiché hanno le potenzialità di un computer universale: è stata infatti dimostrata l'esistenza di automi cellulari universali, anche semplici come la "regola 110" e *Life*, capaci di emulare il comportamento di qualsiasi altro automa cellulare e di qualsiasi macchina di Turing!

Parole chiave: Automi cellulari.

Gattaca (Francia, 2014)

Ognuna delle carte che vedete qui sotto contiene una lettera fra **a, c, g, t** che rappresenta una delle basi che costituiscono il DNA dei castori.

Da qualche parte nel DNA si nasconde la sequenza **gattaca**, scritta da sinistra a destra senza interruzioni.

Nell'intento di scoprire dove si trova, il canguro Angelo ha già voltato (vedi figura) cinque carte, ma ha operato un po' a caso. Volendo aiutarlo a minimizzare il numero di carte da scoprire, cliccate sette carte che certamente non possono far parte della sequenza cercata.

Soluzione.
Una soluzione consiste nel cliccare 7 delle 11 carte colorate in giallo nella figura.

Approfondimenti. Notate che, nel peggiore dei casi, il canguro Angelo dovrà ancora voltare ben 13 delle 14 carte "bianche" per scoprire dove si nasconde la sequenza cercata (o anche per concludere che non c'è, nell'eventualità che appunto non vi sia): ad esempio, supponete che, subito dopo la seconda a scoperta, inizi una sequenza che differisca da `gattaca` per una sola lettera — proprio quella che voi scoprirete per settima! — mentre la `t` già scoperta sia la prima `t` di `gattaca`...

Verso la fine degli anni '70 del secolo scorso, furono ideati alcuni algoritmi particolarmente efficienti per la ricerca di una sequenza di caratteri o *pattern*, di lunghezza m, in un testo, di lunghezza n (con $n \geqslant m$), in alternativa al metodo di forza bruta (che, nel caso peggiore, richiede un numero di confronti fra caratteri di ordine $n \cdot m$). L'algoritmo di Knuth, Morris e Pratt e quello di Boyer e Moore, pubblicati entrambi nel 1977, oltre alla memorizzazione del testo e del *pattern*, richiedono la preparazione e l'uso di una sequenza ausiliaria, di lunghezza rispettivamente pari a m e al numero dei diversi caratteri utilizzati. Il primo ha il vantaggio di procedere sempre in avanti nel testo (dunque è particolarmente adatto al caso in cui il testo è memorizzato in una struttura di dati ad accesso sequenziale) e può rivelarsi assai piú veloce del metodo di forza bruta quando il *pattern* e il testo sono auto-ripetitivi, come potrebbe accadere nel caso del quesito qui proposto. Il secondo algoritmo, invece, torna indietro nel testo, ma è spesso piú efficiente: infatti, il numero di confronti *atteso* è di ordine n/m, purché m sia abbastanza piccolo rispetto al numero dei diversi caratteri utilizzati, oltre che a n (e questo non è il nostro caso: l'alfabeto è di soli 4 caratteri e $m = 7$).

Nel 1980, M. O. Rabin e R. M. Karp escogitarono un algoritmo piuttosto semplice che, *con buona probabilità*, richiede un tempo proporzionale a $m + n$ (che è un limite inferiore per il problema in sé) e, inoltre, può essere facilmente esteso al caso bidimensionale, per l'elaborazione di immagini.

Parole chiave: Ricerca di *pattern*.

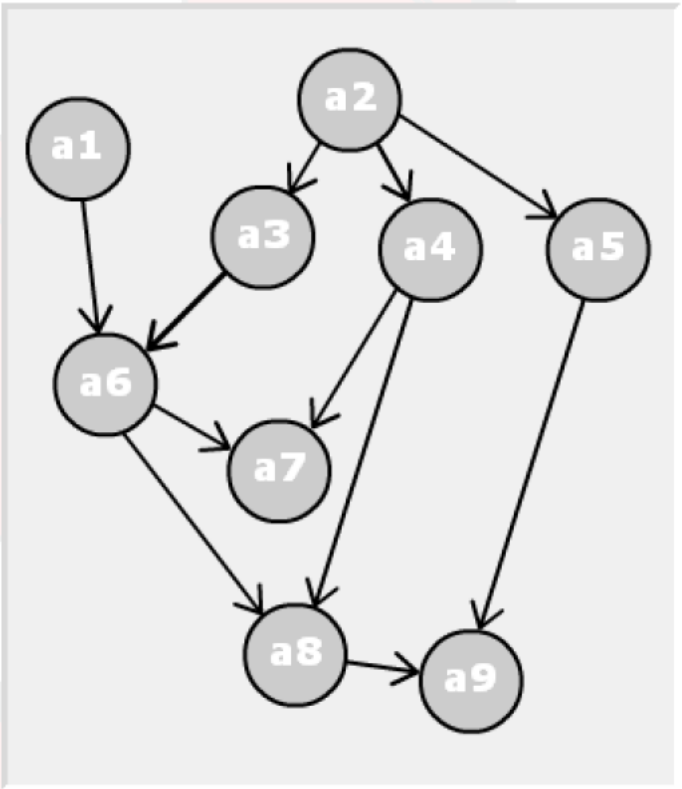
Lavori di manutenzione (6 punti)

Aldo, Bruno e Carlo stanno pianificando i lavori di manutenzione del loro "rifugio"... Come tecnici provetti, desiderano ultimare tutti i compiti nel più breve lasso di tempo. Così individuano ed elencano le diverse attività da svolgere, con le rispettive durate previste (in ore di lavoro) e le precedenze che devono essere rispettate.

Tralasciando qui le descrizioni dei lavori che costituiscono ciascuna attività, il loro piano dovrà rispettare questo schema:

Attività	Durata
a1	2
a2	8
a3	5
a4	9
a5	7
a6	3
a7	5
a8	4
a9	3

Il grafo disegnato qui a destra stabilisce le precedenze tra le attività: a1 e a2 possono cominciare subito; a2 è preliminare ad a3, a4 e a5, cioè queste potranno cominciare soltanto quando a2 sarà terminata; a6 potrà cominciare soltanto quando saranno terminate entrambe le attività a1 e a3, e così via. Ciascuna attività sarà svolta da uno solo dei tre ragazzi; anche se colui che si prende carico di una certa attività fosse aiutato da qualcuno dei suoi amici, la durata di quell'attività non cambierebbe...

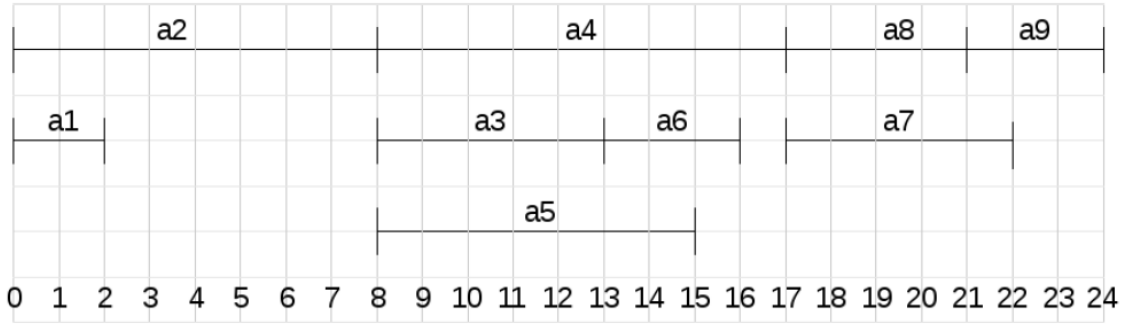


In quante ore, come minimo, potranno essere completate tutte le attività previste?

- 21
- 23
- 24
- 34
- 46

Nota: ai fini del punteggio conta solo la scelta del numero di ore; per aiutarti a trovare la soluzione, puoi associare un colore a ciascuno dei tre amici, quindi colorare le attività nel grafo (ad ogni clic cambierà il colore assegnato).

Soluzione. Le attività previste potranno essere completate in 24 ore come minimo. C'è infatti un *percorso critico*, che stabilisce un *limite inferiore* alla durata dei lavori: in questo caso è costituito dalle attività a2, a4, a8 e a9, che complessivamente occupano 24 ore. Essendo tre i ragazzi, ce la possono fare in questo tempo, come mostra il seguente *diagramma di Gantt*, che riporta la collocazione temporale nel caso in cui ciascuna attività sia fatta iniziare il piú presto possibile:



Si noti che questo diagramma ha il minimo numero di righe, ma non rispecchia necessariamente una ripartizione delle attività tra i ragazzi, ossia non è detto che a ciascuna riga corrisponda un nome: qui si riesce a distribuire equamente il lavoro, nel modo migliore senza "spezzare" le attività, assegnando ad esempio ad Aldo a1, a4 e a7 (per un totale di 16 ore); a Bruno a2 e a5 (15 ore); a Carlo a3, a6, a8 e a9 (15 ore)... ma questo sarebbe stato un problema piú difficile! Il problema specifico che è stato proposto non è particolarmente arduo, anche se comporta un po' di calcoli: qui, infatti, bastava trovare il percorso piú "costoso" (in termini di tempo) da una delle attività che non sono precedute da altre (a1 e a2) a una di quelle che non sono seguite da altre (a7 e a9), ed è facile accorgersi che tale percorso è costituito da a2, a4, a8 e a9. Dunque, il periodo di tempo minimo non può essere inferiore a 24 ore. Provando poi a disporre le altre attività in modo da rispettare i vincoli di precedenza ma anche da sfruttare ed evidenziare il parallelismo (almeno di a2 con a1 e di a4 con a3 e a5), si arriva senza eccessiva difficoltà a un diagramma di tre righe come quello sopra riportato.

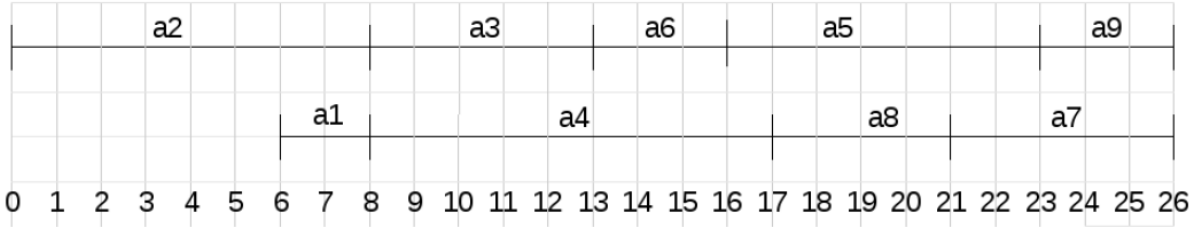
Contesto informatico e riferimenti. Il quesito proposto è un problema di pianificazione (della produzione o di un progetto: in inglese è chiamato *project scheduling*) che consiste nell'ordinare in senso temporale un insieme di attività, in modo da minimizzare il tempo di completamento rispettando la relazione di precedenza tra le attività stesse. Tale relazione è rappresentata dagli archi del grafo orientato (che, chiaramente, deve essere privo di cicli) dato nella formulazione del problema.

Tecniche, dette *reticolari*, per affrontare problemi di questo tipo — di particolare interesse aziendale — furono sviluppate a partire dalla seconda metà degli anni '50. Se si ipotizza che le *risorse* siano *illimitate*, allora la questione è *trattabile* (dal punto di vista computazionale). Questa ipotesi non rientra nel problema proposto, dove i ragazzi sono soltanto tre; tuttavia, anche supponendo di poter reclutare in aiuto quanti ragazzi si vogliono, il tempo totale resterebbe di 24 ore — per cui si vede che tre ragazzi bastano per eseguire tutti i lavori in questo lasso di tempo. (Una piccola ammenda: nella realtà dei fatti, trattandosi di lavori di manutenzione, non è affatto plausibile ipotizzare che la durata di un'attività non cambi a fronte di un aiuto; tuttavia qui si voleva proporre un problema con determinate caratteristiche, e con tempi prefissati e certi.)

Si possono fare alcune interessanti osservazioni, rimanendo nel limite inferiore di tempo (ammesso che certe attività non si prolunghino):

- a1 può slittare in avanti (se ci fosse un quarto ragazzo, potrebbe addirittura sovrapporsi ad a3: l'importante, infatti, è che finisca entro l'inizio di a6);
- d'altra parte, anche la coppia a3–a6 o soltanto a6 possono slittare in avanti di un'ora;
- e infine a5 può slittare in avanti fino a sei ore, o chi la svolge se la potrebbe prendere piú comoda!

Assai piú difficile è l'analogo problema nell'ipotesi di *risorse limitate*, quando è davvero restrittiva. Si possono seguire criteri *euristici*, ma sostanzialmente — se si vuole la soluzione ottima — bisogna cercarla tra tutte le possibili. Ad esempio, se i ragazzi fossero soltanto due, Aldo e Bruno, di quanto si allungherebbe la durata dei lavori? Di sole *due* ore, ed ecco la soluzione ottima (dove a1, oltre ad anticipare, può scambiarsi con a2):

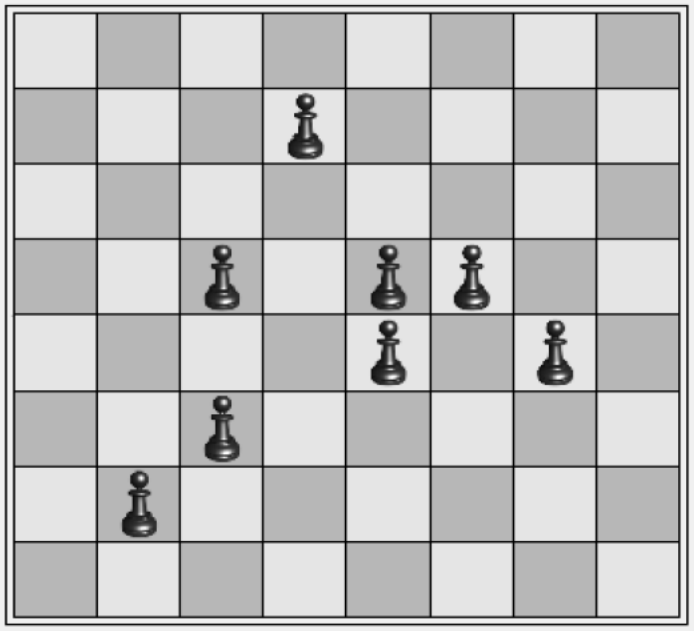


Qui una riga del diagramma dovrà riferirsi ad Aldo, l'altra a Bruno. Questo diagramma si può ottenere partendo dall'istante finale (sconosciuto) e assegnando a ritroso l'attività piú lunga, compatibilmente con i vincoli di precedenza, al primo ragazzo libero, che se ne prenderà carico. Ovviamente questo criterio non assicura la soluzione ottima! Adottando invece l'algoritmo che parte dall'istante iniziale e, procedendo in avanti, assegna ancora l'attività piú lunga (come sopra), per questo specifico problema si troverebbe una soluzione peggiore: 30 ore complessive; altri criteri portano a durate intermedie. Per ogni regola che può essere stabilita nell'assegnare i vari compiti, si possono trovare dei casi in cui darà il risultato migliore, ma pure altri casi in cui darà il peggiore! E, quando la dimensione del problema aumenta, la ricerca della soluzione ottima diviene presto intrattabile: ricordate i problemi NP-completi che abbiamo incontrato lo scorso anno, come quello del commesso viaggiatore?

Parole chiave: *project scheduling*, percorso critico, diagramma di Gantt, risorse, criteri euristici.

Fianco... sinist? (5 punti) (Italia, 2014)

Avete una scacchiera con 8 pedine disposte come in figura. Lo scopo del gioco è spostare tutte le pedine su uno stesso bordo della scacchiera, potendole muovere solo lungo le traverse e le colonne (quindi non in diagonale e ovviamente non su caselle occupate).



Seleziona una casella con una pedina

Reset

La scacchiera serve solo per sperimentare. Qual è il bordo più vicino, cioè raggiungibile attraversando il numero minimo di caselle?

Nord	Est	Sud	Ovest
○	○	○	○

In quante mosse si può raggiungere?

Soluzione.
Il bordo raggiungibile attraversando il numero minimo di caselle si trova a 'Sud' e per raggiungerlo servono 32 mosse.

Anche questa è informatica! Come si può calcolare il numero minimo di caselle da attraversare per spostare tutte e otto le pedine su un bordo (prefissato) della scacchiera? In modo piuttosto informale, descriviamo una delle procedure (un *algoritmo*) che rispondono efficientemente a questa domanda, in generale (qualunque sia cioè la dimensione della scacchiera $n \times n$ e ovunque siano sistemate le n pedine), senza tuttavia dimostrarne la correttezza, ossia che il risultato da essa fornito è davvero il migliore possibile.

Supponiamo, ad esempio, di voler allineare le otto pedine dello schema proposto sul bordo inferiore ('Sud') della scacchiera. Numeriamole per colonne (se volessimo allinearle sul bordo sinistro o sul destro, le dovremmo numerare per traverse), procedendo ad esempio da sinistra a destra: la pedina 1 è quella in seconda colonna; in terza colonna ne troviamo due: ad esempio, diciamo che la pedina 2 è quella piú in basso, l'altra sarà la 3 (ma potremmo anche dire il viceversa, grazie alle proprietà della "geometria Manhattan"!); la pedina 4 è quella in quarta colonna; in quinta colonna ne troviamo di nuovo due: la pedina 5 è quella piú in alto, l'altra è la 6 (ma potremmo anche scambiare i due numeri); infine, in sesta e in settima colonna ci sono le pedine 7 e 8, rispettivamente.

A questo punto, basterà calcolare la "distanza Manhattan" tra la pedina i e la prima casella in basso della i-esima colonna (semplicemente contando il numero di caselle che le separano ortogonalmente), per $i = 1, \ldots, 8$, e infine sommare tali distanze. Otteniamo cosí: $2 + 3 + 4 + 6 + 4 + 4 + 5 + 4 = 32$.

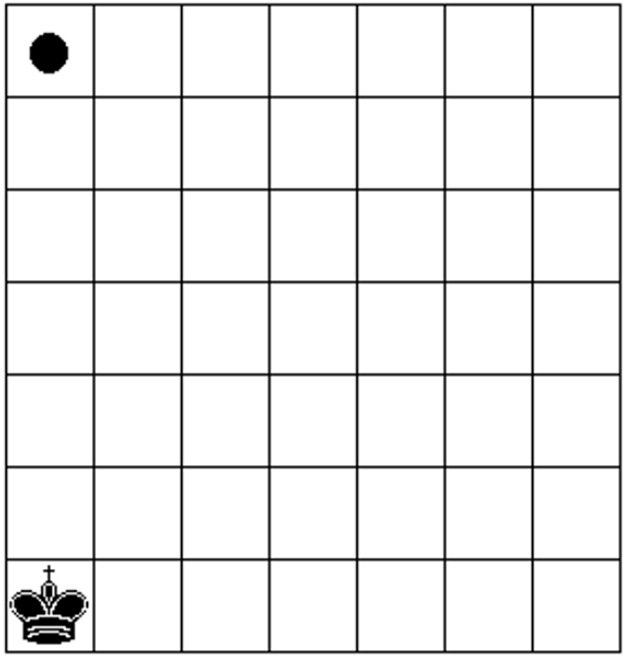
Il bello è che, qualora volessimo muovere realmente le pedine anziché calcolare soltanto il numero complessivo di caselle da attraversare, potremmo sempre riordinare gli spostamenti da effettuare, in modo che non si ostacolino a vicenda! Ad esempio, nel nostro caso, potremo muovere le pedine in questo ordine (ciascuna nella casella in basso della corrispondente colonna): 1, 2, 3, 4, 8, 7, 6, 5.

Se, partendo dalla stessa configurazione iniziale, ripetiamo ora il procedimento per gli altri tre bordi: Nord, Ovest, Est (negli ultimi due casi, per numerare le traverse e le pedine, possiamo partire dal basso o dall'alto arbitrariamente e, quando vi sono piú pedine sulla medesima traversa, l'ordine in cui le consideriamo è ancora indifferente), troviamo: 34, 34, 36, rispettivamente.

Parole chiave e riferimenti: Algoritmo, geometria Manhattan.

Quesito 13 – I cammini del Re… dall'angolo

Un Re si trova nella casa d'angolo in basso a sinistra di una scacchiera 7×7, come mostrato in figura. Supponiamo che il Re possa spostarsi di una casa verso l'alto, in verticale o in diagonale (a sinistra o a destra), naturalmente senza uscire dalla scacchiera.



Quanti sono i diversi percorsi che il Re può seguire per arrivare nella casa segnata in alto?
Nota: rappresentando un percorso come sequenza di spostamenti (ciascuno dei quali è o in verticale o in diagonale a sinistra o in diagonale a destra), nell'ordine in cui sono eseguiti, si intende che due percorsi siano diversi quando non sono rappresentati dalla stessa sequenza di spostamenti.

Su una scacchiera $n \times n$, tutti i percorsi che conducono il Re a destinazione hanno lunghezza $n - 1$, e i passi in diagonale a destra devono essere tanti quanti i passi in diagonale a sinistra, ma in ogni momento il numero dei passi verso sinistra non può essere maggiore del numero dei passi fatti verso destrala soluzione di questo non semplice problema può essere associata alla *successione di Motzkin* (*cfr*. OEIS A001006, *Motzkin numbers*), ricavabile mediante una *relazione di ricorrenza* a tre variabili (lineare, a coefficienti costanti), come mostrato nel seguente programma in Python:

```
def p(i, j, k):
    if i < 0 or j < 0 or k < 0:
        return 0
    if i == 0 and j == 0 and k == 0:
        return 1
    return p(i, j+1, k-1) + p(i-1, j, k-1) + p(i+1, j-1, k-1)

for k in range(12):
    s = 0
    for i in range(k+1):
        for j in range(k+1):
            s += p(i, j, k)
    print s,
```

L'esecuzione di questo programma stampa i primi 12 numeri di tale successione:

```
1   1   2   4   9   21   51   127   323   835   2188   5798
```

Il settimo è 51, la risposta al quesito proposto. Si noti che la funzione `p`, definita in forma ricorsiva, chiama sé stessa tre volte, con `k` decrementato di una unità; ci si aspetta quindi che il tempo di esecuzione all'incirca triplichi ogni volta che il terzo argomento aumenta di una unità: una crescita esponenziale, dunque, come avrete notato dall'evidente rallentamento delle stampe in output…
Gli stessi risultati si ottengono mediante una più efficiente procedura iterativa:

```
A = [1, 1]
B = [0, 1]
k = 2
print 1, 1,
for count in range(10):
    s = 0
    for i in range(k):
        s += A[i]*B[i]
    B = B + [A[0]]     # qui + concatena due liste
    A = [s+A[0]] + A   # il primo + addiziona, il secondo concatena
    k += 1
    print A[0],
```