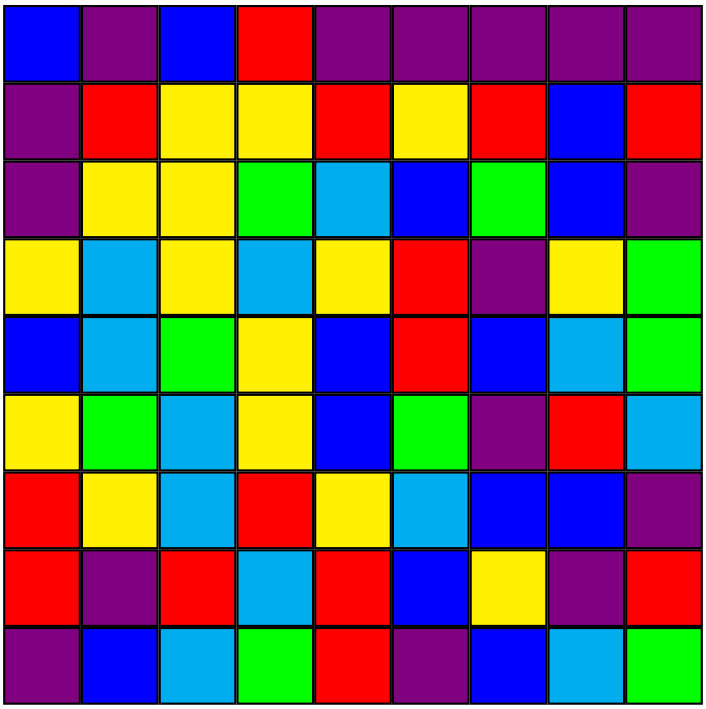
SQUADRA N.31

Il muro delle password

Questo quesito vi consentirà di scoprire la password con cui accedere al computer.

Giochiamo a Clickomania! Considerate questa parete iniziale:



e la sequenza di mosse (2,2) (3,1) (2,0) (2,2) (1,4) .

Come diventerà la parete dopo aver eseguito tutte le mosse?

Considerate in particolare la quarta riga dall'alto: **la vostra password è data dalla sequenza di lettere maiuscole corrispondenti alle iniziali dei colori dei mattoni che formano la quarta riga, partendo da sinistra.**

I colori dei mattoni sono: blu (B), rosso (R), giallo (G), fucsia (F), azzurro (A), verde (V). Nel caso in cui la riga contenga mattoni rimossi, usate N per indicare il nero di sfondo.

Quindi la vostra password è:

. . .

SQUADRA N.

Quesiti da svolgere esclusivamente su carta

Mini-robot

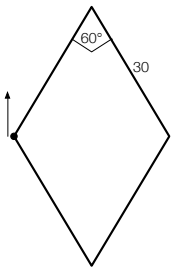
Difficoltà: facile
Abbiamo un mini-robot in grado di eseguire questi comandi:

- V: avanza di un passo;
- $S(g)$: ruota a sinistra di un angolo di g gradi;
- $D(g)$: ruota a destra di un angolo di g gradi.

Più comandi in sequenza sono separati dal simbolo $+$, mentre la ripetizione per n volte di un comando o una sequenza C di comandi può essere concisamente indicata da $n * (C)$. Ad esempio:

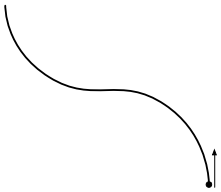
- la sequenza $V + S(30) + 5 * (V) + D(45)$ significa: (dalla posizione in cui ti trovi e nel verso in cui guardi) avanza di un passo, poi girati di 30 gradi a sinistra, poi avanza di 5 passi, infine girati di 45 gradi a destra;
- il comando $3 * (10 * (V) + D(60))$ significa: ripeti per 3 volte queste due azioni: avanza di 10 passi e poi girati di 60 gradi a destra.

Se il mini-robot si trova rivolto verso nord nella posizione indicata dal pallino nero nella figura qui sotto, quali comandi gli faranno percorrere la traiettoria romboidale, dove il lato del rombo è percorso in 30 passi? Attenzione: alla fine il mini-robot dovrà ritrovarsi nella stessa precisa posizione di partenza (rivolto a nord).



Se il mini-robot riceve il comando $180*(V+S(1))$, percorrerà una traiettoria "semicircolare", come indicato nella figura qui sotto, e alla fine sarà rivolto verso sud.

Quali comandi faranno invece percorrere al mini-robot la traiettoria sotto indicata?

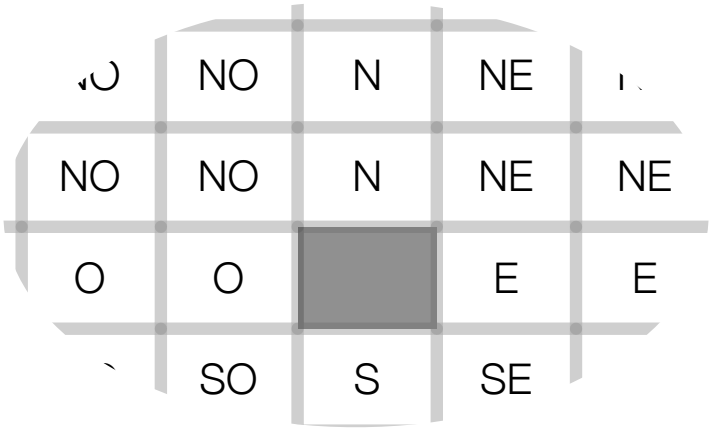


Caccia al tesoro

Difficoltà: media

C'è un tesoro nascosto da qualche parte in un'area quadrata, suddivisa in quadrati piú piccoli, come una scacchiera con n caselle per lato. Ad ogni vostro tentativo, voi indicate una casella e ottenete una tra queste nove risposte: Ok, N, O, S, E, NO, SO, SE, NE.

La risposta Ok significa che il tesoro è nascosto proprio nella casella da voi indicata; la risposta N significa che il tesoro si trova in una delle caselle a Nord, sulla stessa colonna di quella da voi indicata; la risposta O significa che il tesoro si trova in una delle caselle a Ovest, sulla stessa riga di quella da voi indicata; la risposta NO significa che il tesoro si trova in una delle caselle piú a Nord e piú a Ovest di quella da voi indicata. Analogamente per le altre risposte, dove S sta per Sud ed E sta per Est. Per maggior chiarezza, nella figura qui sotto è disegnata una porzione dell'area quadrata e voi avete indicato la casella in grigio; in ciascuna casella è riportata la risposta che voi otterreste se il tesoro fosse nascosto proprio in quella casella. (Naturalmente se il tesoro fosse nella casella in grigio, la risposta ottenuta sarebbe Ok.)



Si vuol sapere qual è la dimensione massima dell'area quadrata (in numero di caselle per lato) affinché possiate avere la certezza di indovinare dove si trova il tesoro al piú al quarto tentativo. Giustificare la risposta!

Attraversamento del fiume in notturna

Difficoltà: media

Quattro castori devono attraversare un ponte sul fiume, ed è notte fonda. Per fortuna hanno una torcia elettrica, ma il ponte può essere percorso da al piú due castori per volta, e chi attraversa il ponte deve avere con sé la torcia, mentre gli altri possono aspettare al buio. I quattro castori non sono ugualmente veloci: per attraversare il ponte, Alex impiega 1 minuto, Bob 2 minuti, Chris 5 minuti e Dan, che è il piú lento, 10 minuti. Qual è il tempo minimo necessario affinché tutti e quattro i castori si ritrovino sulla riva opposta del fiume? Giustificare la risposta!

Un vecchio calcolatore

Difficoltà: difficile
Il castoro Adriano ha trovato in soffitta un vecchio calcolatore, che dispone di tre soli *registri* (aree di memoria in ciascuna delle quali può essere memorizzato un numero), chiamati R_1, R_2 e R_3.
Per programmare questa macchina, è necessario codificare in una opportuna sequenza le operazioni che essa dovrà svolgere. Le operazioni che la macchina può compiere sono di sei tipi:

- `Zero(i)`: memorizza 0 nel registro R_i;
- `Inc(i)`: incrementa di 1 il contenuto del registro R_i;
- `Dec(i)`: decrementa di 1 il contenuto del registro R_i;
- `Store(i, j)`: copia il contenuto del registro R_j nel registro R_i;
- `Jump(i, j, n)`: se R_i e R_j contengono lo stesso valore, allora salta all'operazione numero n
- `JumpNeg(i, j, n)`: se R_i e R_j contengono valori diversi, allora salta all'operazione numero n

dove i e j indicano il numero di un registro (e quindi possono essere 1, 2 o 3) e n indica il numero d'ordine di un'operazione nella sequenza.
Ad esempio, il programma seguente scambia il contenuto dei registri R_1 e R_2:

```
1:  Store(3,2)
2:  Store(2,1)
3:  Store(1,3)
```

Il castoro Adriano memorizza due numeri nei registri R_1 e R_2 e vuole scrivere un programma che calcoli la loro somma, memorizzandola in R_1 al posto del primo addendo.
Sapreste aiutarlo?

Quesiti da svolgere con l'aiuto del computer

Un programma affollato di eventi

Il parco di divertimento *Kangourandia* offre ai suoi visitatori un ricco calendario di eventi. Ogni giorno sono previsti film 4D, esibizioni degli *stunt-men* e dei tuffatori, commedie musicali, animazione dei bambini, parate delle *mascotte*, e altro ancora.
Avete solo un giorno a disposizione e volete assistere al massimo numero di eventi possibili. Naturalmente non è possibile assistere ad eventi che si svolgono in contemporanea (anche parzialmente).
Il vostro amico Ale Gridi, genio dell'informatica, vi ha suggerito di usare questa strategia per scegliere gli eventi cui assistere:

> Mettete in ordine gli eventi del programma, poi analizzateli uno alla volta rispettando quell'ordine: se l'evento in questione è il primo considerato oppure non si sovrappone agli altri eventi già scelti, allora assisterete all'evento, altrimenti lo scarterete e non lo prenderete più in considerazione.

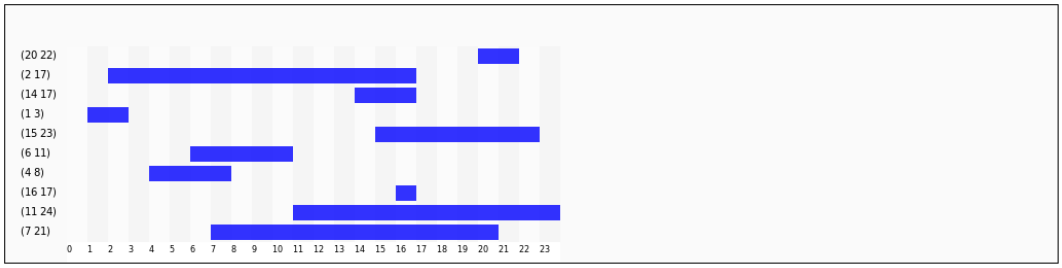
In questo modo, garantisce Ale Gridi, riuscirete ad assistere ad un numero di eventi che nessuno potrà battere.
L'unico problema è che non vi ricordate in che modo Ale Gridi vi ha suggerito di ordinare gli eventi e le possibilità sono molte!
Per individuare un buon criterio di ordinamento, avete a disposizione un programma che vi aiuterà a fare degli esperimenti. Per avviarlo lanciate il programma *Eventi*, vi comparirà una pagina con tre riquadri.

- Nel riquadro in alto potete inserire e modificare le informazioni relative agli orari degli eventi; per semplicità assumiamo che gli eventi inizino e finiscano tutti allo scoccare dell'ora. Per ogni evento, va indicata la coppia di ora di inizio e ora di fine nel formato `(inizio fine)`. Ad esempio la sequenza

  ```
  (20 22)(2 17)(14 17)(1 3)(15 23)(6 11)(4 8)(16 17)(11 24)(7 21)
  ```

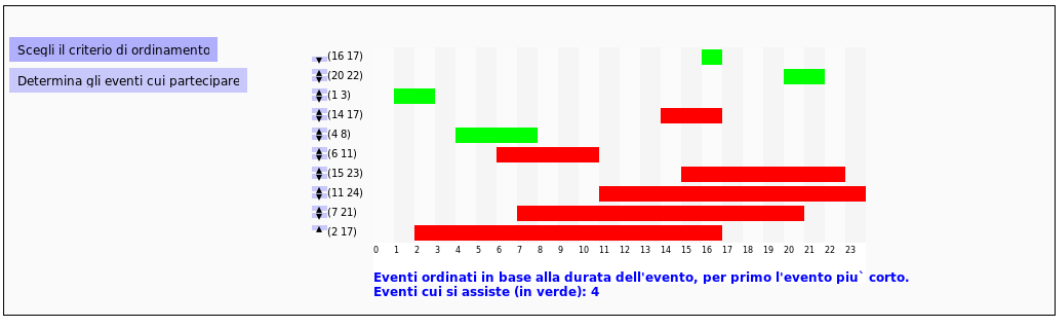
  rappresenta dieci eventi, uno che inizia alle 20 e finisce alle 22, uno che inizia alle 2 e finisce alle 17, uno che inizia alle 14 e finisce alle 17, e cosí via. Possono esserci piú eventi con lo stesso orario, in questo caso ci sarà una coppia `(inizio fine)` che si ripete nella sequenza.

- Nel riquadro centrale trovate una rappresentazione grafica degli eventi in base al loro orario.



- Nel riquadro in basso potete modificare l'ordine in cui vengono visualizzati gli eventi elencati nella sequenza del primo riquadro. Con le freccine a sinistra potete spostare verso l'alto o il basso ciascun evento. Potete inoltre scegliere un criterio di ordinamento e ottenere la visualizzazione degli eventi

nell'ordine corrispondente. Infine, cliccando su "determina gli eventi cui partecipare", potete applicare, agli eventi cosí ordinati, la strategia suggerita da Ale Gridi, ottenendo in verde l'elenco degli eventi cui assistere. Ad esempio, ordinando in base all'ora di inizio (per primo quello che inizia per primo) si otterebbe questa selezione di eventi:



Per ciascuno dei seguenti criteri di ordinamento, scrivete se è un criterio *adatto* oppure no, nel senso che permette di selezionare il maggior numero di eventi possibili cui assistere. Inoltre, per ciascuno dei criteri che considerate non adatti, fornite un esempio (più semplice è l'esempio, maggiore sarà il punteggio) che mostra perché il criterio non è adatto. Per ciascuno dei criteri che considerate adatti, illustrate, nell'ultimo riquadro di questo quesito, il ragionamento che vi ha portato a questa conclusione.

1. Criterio: in base all'ora d'inizio, per primo quello che inizia prima.

2. Criterio: in base all'ora di fine, per primo quello che finisce prima.

3. Criterio: in base all'ora d'inizio, per primo quello che inizia più tardi.

4. Criterio: in base all'ora di fine, per primo quello che finisce più tardi.

5. Criterio: in base alla durata dell'evento, per primo l'evento più corto.

6. Criterio: in base alla durata dell'evento, per primo l'evento più lungo.

7. Criterio: in base al numero di sovrapposizioni, per primo l'evento che ha meno sovrapposizioni con gli altri eventi.

8. Criterio: in base al numero di sovrapposizioni, per primo l'evento che ha più sovrapposizioni con gli altri eventi.

Illustrate in modo convincente che i criteri (o il criterio) che avete considerato adatti sono imbattibili.

Clickomania

Vi ricordate le regole di *Clickomania* che avete ricevuto ieri sera?
Vostro compito è realizzare un programma per giocare a Clickomania o, piú precisamente, un programma che aggiorni la parete di gioco in seguito a ogni mossa (click) del giocatore, utilizzando l'*ambiente di programmazione Blockly*.

Blockly

Blockly consente di *costruire dei programmi* a partire da blocchi colorati predefiniti, che si possono incastrare come in un puzzle.

Non c'è un solo programma che permette di giocare a Clickomania: i blocchi a disposizione possono essere combinati a formare tanti diversi programmi, tutti funzionanti. Tuttavia ciascun blocco ha un *costo* e il vostro punteggio sarà tanto maggiore quanto minore sarà la somma dei costi dei blocchi usati.

Una volta costruito un programma, si può iniziare il gioco cliccando sul bottone "Gioca". A questo punto, ad ogni click del giocatore (voi stessi, mentre testate il vostro programma!!), la parete verrà aggiornata in base alle istruzioni del programma costruito.

Blockly consente inoltre di vedere gli effetti dell'esecuzione del programma costruito, attraverso un'animazione, direttamente sulla parete di gioco, e permette di salvare fino a 4 programmi costruiti.

Interfaccia

Lanciate il programma *Clickomania*: vedrete una pagina composta di tre colonne.

- A sinistra trovate vari elementi.
  - La parete di mattoni.
  - Un cursore con ai lati una tartaruga e una lepre, per aumentare/diminuire la velocità dell'animazione.
  - Il bottone "Gioca" che consente di iniziare una partita e che si trasforma in "Reset" durante l'esecuzione del programma; cliccando su "Reset" è possibile iniziare una nuova partita.
  - Le icone relative a salvataggio/visualizzazione/caricamento dei programmi (o versioni di uno stesso programma): cliccando sul primo simbolo si può salvare il programma attualmente in costruzione, soffermandosi col mouse sul secondo simbolo si può visualizzare il programma salvato, cliccando sul terzo simbolo si può caricare il programma salvato in precedenza.
- Al centro trovate un menu di categorie tra cui scegliere i blocchi colorati; soffermandosi col mouse su un blocco si può leggere una descrizione di cosa fa il blocco.
- A destra trovate uno spazio bianco in cui potete trascinare i blocchi scelti dal menu, per costruire le istruzioni del vostro programma. Nell'angolo in basso a destra c'è il simbolo di un cestino nel quale possono essere trascinati i blocchi che non servono più.

  Durante l'aggiornamento della parete in seguito ad una mossa (click), i blocchi in esecuzione saranno via via evidenziati. Per fare degli esperimenti, tra una mossa e l'altra potete modificare il programma; però come soluzione del quesito accetteremo soltanto un programma che non debba essere modificato a seconda della mossa!

Blocchi a disposizione

I blocchi disponibili sono raggruppati nelle seguenti categorie (alcuni blocchi sono presenti in piú di una categoria): Aggiornamento parete, Caselle, Colori, Infezione, Logica, Cicli, Matematica, Variabili.
Ciascun blocco ha un costo che si può visualizzare soffermandosi col mouse sopra il blocco stesso. L'elenco completo dei blocchi con i loro costi è disponibile nel foglio allegato.

Esempi

Il programma più semplice che gioca a Clickomania è il seguente. Ad ogni click, vengono rimossi tutti i mattoni che fanno parte dell'area di contagio attorno alla casella cliccata, e gli spazi vuoti che si vengono a creare vengono eliminati dalla caduta dei mattoni che si trovano sopra.

Vediamo ora qualche esempio un po' più complicato (non è detto che siano di qualche utilità!!):

In questo caso, se la casella cliccata si trova nella riga di indice 5, allora viene infettato il mattone nella casella di coordinate $(1, 4)$ e viene colorato di rosso il mattone nella casella cliccata.

In questo caso, viene considerata la casella più in alto tra quelle contenenti un mattone infetto, assieme alla casella cliccata; se le due caselle contengono un mattone dello stesso colore, allora verrà rimosso il mattone che si trova nella casella di coordinate $(5, 4)$ e cadrà l'intera pila di mattoni posizionata a partire dalla casella di coordinate $(4, 4)$.

In questo caso, se ci sono più di tre mattoni infetti, ne verranno rimossi tre a caso; se ce ne sono tre o meno, verranno rimossi tutti.

Paleografia informatica

Questa prova va svolta su carta. Se volete, potete usare il Web per fare delle ricerche.

Essendo nato nel 4016 aveva ormai quasi quarant'anni, e da tutti era considerato un archeologo esperto, uno dei luminari nel suo campo, la *paleografia informatica*. Eppure non aveva mai partecipato prima di allora a una campagna di scavi che lo emozionasse tanto: sentiva che quello che stavano scoprendo avrebbe gettato una nuova luce sul passato dell'umanità e sui modi misteriosi con cui gli uomini, piú di due millenni prima, interagivano con i computer. Ben prima dell'avvento della programmazione olistica, prima del trasferimento osmotico della conoscenza, prima della costruzione della PCA (Pura Coscienza Algoritmica), gli umani usavano i computer in modo estremamente rudimentale, programmandone il funzionamento direttamente, e lo facevano attraverso misteriosi linguaggi (li chiamavano **"linguaggi di programmazione"**, appunto) il cui significato e la cui sintassi erano ancora in gran parte avvolti nel mistero.
Stava pensando a tutto questo quando, con la mano tremante, si avviava ad esaminare il contenuto della memoria USB trovata il giorno prima, alla fine di una lunga giornata sul campo, sotto il sole cocente dell'antica Silicon Valley, in California: stavano scavando a Mountain View, in quella che era stata la sede di un opificio informatico nato all'inizio del terzo millennio e chiamato Gogol. Era in quell'area che, il giorno prima, avevano recuperato questo piccolo oggetto metallico. L'aveva riconosciuto subito, ne aveva già visti centinaia: era una memoria USB, forse posseduta da qualche dipendente di Gogol che la usava per trasportare i programmi da un computer all'altro, o magari per lavorarci da casa, durante la notte. Quella memoria conteneva con ogni probabilità del codice sorgente di qualche *programma informatico*.
Ora l'oggetto metallico era nel suo laboratorio; maneggiandolo con grande cautela lo aveva inserito nel lettore speciale (a prova di polvere e di contaminazione) che lui stesso aveva costruito negli anni e che gli permetteva di analizzare il contenuto di memorie di quel periodo. Dopo qualche istante, nello schermo olografico, iniziarono a comparire i piccoli involti che rappresentavano i "*file*" (cosí, nell'antichità, erano chiamati i contenitori dei codici sorgente dei programmi). Era bellissimo: c'erano decine di programmi tutti a sua disposizione, e aprendoli avrebbe potuto scoprire che cosa facevano e forse capire meglio i linguaggi in cui erano scritti e di cui ancora nessuno aveva compreso quasi nulla.
Prese in mano il pacchetto olografico di uno dei *file*, e ne svolse i lembi per aprirlo e vederne il contenuto. Quello che segue fu ciò che apparve, nella penombra della stanza.
Sapete identificare a) il linguaggio di programmazione e b) il contesto storico o informatico in cui sono stati creati i seguenti frammenti di codice?

1. Ecco un frammento:

```
/*
 * You are not expected to understand this.
 */
if(rp->p_flag & SSWAP) {
        rp->p_flag &= ~SSWAP;
        aretu(u.u_ssav);
}
```

   Di che si tratta?

2. Ecco un altro frammento:

```
      INTEGER A,B,C
      READ(5,501) A,B,C
  501 FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C))
      WRITE(6,601) A,B,C,AREA
  601 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,12HSQUARE UNITS)
      STOP
      END
```

Di che si tratta?

3. Ecco un altro frammento:

```
s''$/=\2048;while(<>){G=29;R=142;if((@a=unqT="C*",_)[20]&48){D=89;_=unqb24,qT,@
b=map{ord qB8,unqb8,qT,_^$a[--D]}@INC;s/...$/1$&/;Q=unqV,qb25,_;H=73;O=$b[4]<<9
|256|$b[3];Q=Q>>8^(P=(E=255)&(Q>>12^Q>>4^Q/8^Q))<<17,O=O>>8^(E&(F=(S=O>>14&7^O)
^S*8^S<<6))<<9,_=(map{U=_%16orE^=R^=110&(S=(unqT,"\xb\ntd\xbz\x14d")[_/16%8]);E
^=(72,@z=(64,72,G^=12*(U-2?0:S&17)),H^=_%64?12:0,@z)[_%8]}(16..271))[_]^((D>>=8
)+=P+(~F&E))for@a[128..$#a]}print+qT,@a}';s/[D-HO-U_]/\$$&/g;s/q/pack+/g;eval
```

Di che si tratta?

4. Ecco un altro frammento:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.
>---.+++++++..+++.>>.<-.<.+++.------.--------.>>+.>++.
```

Di che si tratta?

5. Ecco un altro frammento:

```
10 Print Chr$(205.5+RND(1)); : GOTO 10
```

Di che si tratta?

6. Ecco un altro frammento:

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
```

```
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Di che si tratta?

7. Ecco un altro frammento:

```
primes = sieve [2..]
sieve (p : xs) = p : sieve [x | x <- xs, x `mod` p > 0]
```

Di che si tratta?

8. Ecco un altro frammento:

```
((lambda (x)
       (list x (list (quote quote) x)))
     (quote
        (lambda (x)
          (list x (list (quote quote) x)))))
```

Di che si tratta?

9. Ecco un altro frammento:

```
mortal(X) :-
  human(X).
human(socrate).
?- mortal(socrate).
yes
```

Di che si tratta?

10. Ecco un altro frammento:

```
tell application "Finder"
    make new folder at desktop
end tell
```

Di che si tratta?

11. Ecco un altro frammento:

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!"
KTHXBYE
```

Di che si tratta?

12. Ecco un altro frammento:

```
keeplooping=1;
while [[ $keeplooping -eq 1 ]] ; do
  read quitnow
  if [[ "$quitnow" = "yes" ]] ; then
    keeplooping=0
  fi
  if [[ "$quitnow" = "q" ]] ; then
    break;
  fi
done
```

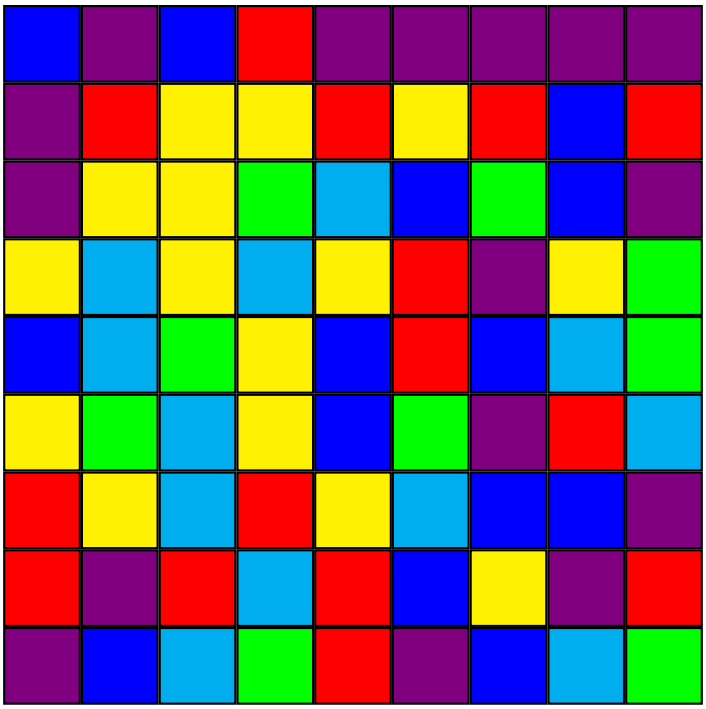
Di che si tratta?

SQUADRA N.31

Il muro delle password

Questo quesito vi consentirà di scoprire la password con cui accedere al computer.

Giochiamo a Clickomania! Considerate questa parete iniziale:



e la sequenza di mosse (2,2) (3,1) (2,0) (2,2) (1,4) .

Come diventerà la parete dopo aver eseguito tutte le mosse?

Considerate in particolare la quarta riga dall'alto: **la vostra password è data dalla sequenza di lettere maiuscole corrispondenti alle iniziali dei colori dei mattoni che formano la quarta riga, partendo da sinistra.**

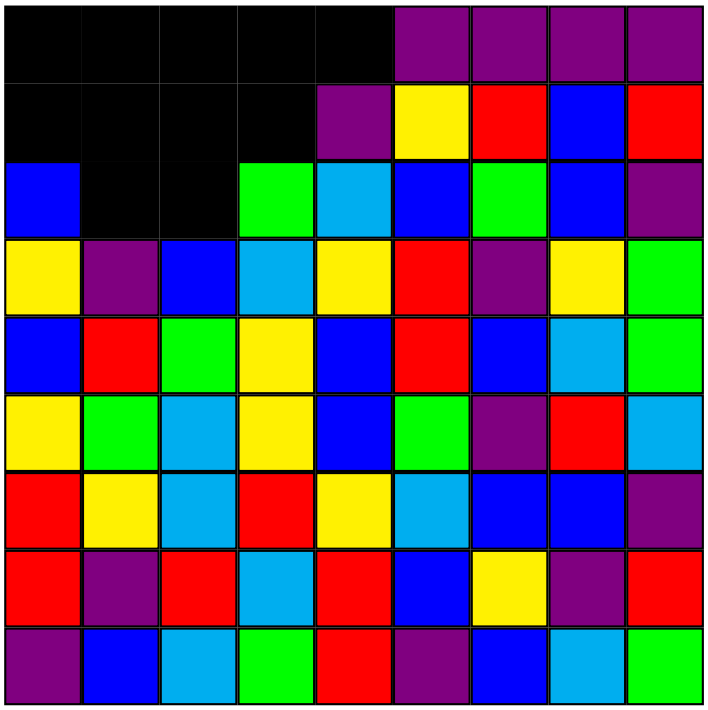
I colori dei mattoni sono: blu (B), rosso (R), giallo (G), fucsia (F), azzurro (A), verde (V). Nel caso in cui la riga contenga mattoni rimossi, usate N per indicare il nero di sfondo.

Quindi la vostra password è:

GFBAGRFGV

Kangourou dell'Informatica 2014
Il muro delle password

SQUADRA N.31



SQUADRA N.

Quesiti da svolgere esclusivamente su carta

Mini-robot

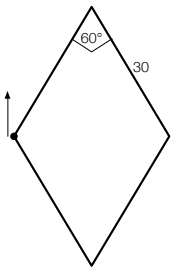
Difficoltà: facile
Abbiamo un mini-robot in grado di eseguire questi comandi:

- V: avanza di un passo;
- $S(g)$: ruota a sinistra di un angolo di g gradi;
- $D(g)$: ruota a destra di un angolo di g gradi.

Più comandi in sequenza sono separati dal simbolo $+$, mentre la ripetizione per n volte di un comando o una sequenza C di comandi può essere concisamente indicata da $n * (C)$. Ad esempio:

- la sequenza $V + S(30) + 5 * (V) + D(45)$ significa: (dalla posizione in cui ti trovi e nel verso in cui guardi) avanza di un passo, poi girati di 30 gradi a sinistra, poi avanza di 5 passi, infine girati di 45 gradi a destra;
- il comando $3 * (10 * (V) + D(60))$ significa: ripeti per 3 volte queste due azioni: avanza di 10 passi e poi girati di 60 gradi a destra.

Se il mini-robot si trova rivolto verso nord nella posizione indicata dal pallino nero nella figura qui sotto, quali comandi gli faranno percorrere la traiettoria romboidale, dove il lato del rombo è percorso in 30 passi? Attenzione: alla fine il mini-robot dovrà ritrovarsi nella stessa precisa posizione di partenza (rivolto a nord).



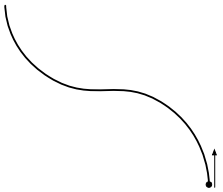
$$D(30) + 30 * V + D(120) + 30 * V + D(60) + 30 * V + D(120) + 30 * V + D(30)$$

oppure, più sinteticamente:

$$2 * (D(30) + 30 * V + D(120) + 30 * V + D(30))$$

Se il mini-robot riceve il comando $180*(V+S(1))$, percorrerà una traiettoria "semicircolare", come indicato nella figura qui sotto, e alla fine sarà rivolto verso sud.

Quali comandi faranno invece percorrere al mini-robot la traiettoria sotto indicata?



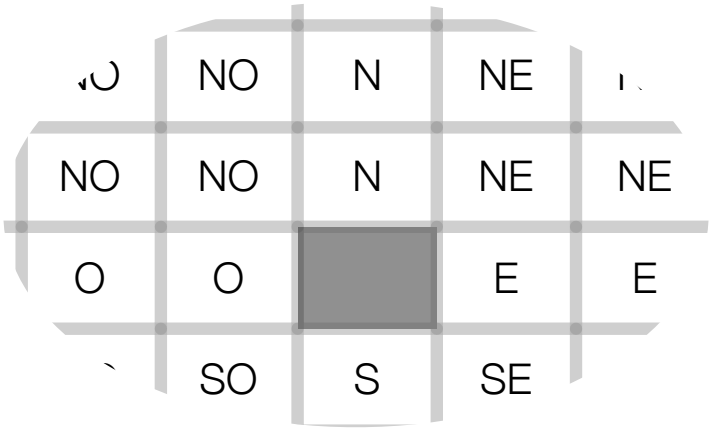
$90 * (V + S(1)) + 90 * (V + D(1))$

Caccia al tesoro

Difficoltà: media

C'è un tesoro nascosto da qualche parte in un'area quadrata, suddivisa in quadrati piú piccoli, come una scacchiera con n caselle per lato. Ad ogni vostro tentativo, voi indicate una casella e ottenete una tra queste nove risposte: Ok, N, O, S, E, NO, SO, SE, NE.

La risposta Ok significa che il tesoro è nascosto proprio nella casella da voi indicata; la risposta N significa che il tesoro si trova in una delle caselle a Nord, sulla stessa colonna di quella da voi indicata; la risposta O significa che il tesoro si trova in una delle caselle a Ovest, sulla stessa riga di quella da voi indicata; la risposta NO significa che il tesoro si trova in una delle caselle piú a Nord e piú a Ovest di quella da voi indicata. Analogamente per le altre risposte, dove S sta per Sud ed E sta per Est. Per maggior chiarezza, nella figura qui sotto è disegnata una porzione dell'area quadrata e voi avete indicato la casella in grigio; in ciascuna casella è riportata la risposta che voi otterreste se il tesoro fosse nascosto proprio in quella casella. (Naturalmente se il tesoro fosse nella casella in grigio, la risposta ottenuta sarebbe Ok.)



Si vuol sapere qual è la dimensione massima dell'area quadrata (in numero di caselle per lato) affinché possiate avere la certezza di indovinare dove si trova il tesoro al piú al quarto tentativo. Giustificare la risposta!

15. Si tratta di un'immediata estensione al caso bidimensionale del procedimento di "ricerca binaria". È stata ritenuta accettabile anche la risposta 16, considerando i quattro tentativi come quelli necessari ad eliminare l'incertezza piuttosto che ad identificare il tesoro.

Attraversamento del fiume in notturna

Difficoltà: media

Quattro castori devono attraversare un ponte sul fiume, ed è notte fonda. Per fortuna hanno una torcia elettrica, ma il ponte può essere percorso da al piú due castori per volta, e chi attraversa il ponte deve avere con sé la torcia, mentre gli altri possono aspettare al buio. I quattro castori non sono ugualmente veloci: per attraversare il ponte, Alex impiega 1 minuto, Bob 2 minuti, Chris 5 minuti e Dan, che è il piú lento, 10 minuti. Qual è il tempo minimo necessario affinché tutti e quattro i castori si ritrovino sulla riva opposta del fiume? Giustificare la risposta!

17 minuti.

Se ogni volta fosse Alex, il castoro piú veloce, ad accompagnare uno dei suoi compagni sull'altra sponda del fiume, per tornare poi indietro da solo a prenderne un altro, allora il tempo complessivo sarebbe di 19 minuti. Ma c'è un'altra possibilità, che permette di risparmiare due minuti:

- attraversano Alex e Bob, impiegando 2 minuti;
- Alex torna indietro, impiegando un minuto;
- attraversano Chris e Dan, impiegando 10 minuti;
- Bob torna indietro, impiegando 2 minuti;
- attraversano Alex e Bob, impiegando 2 minuti.

Un vecchio calcolatore

Difficoltà: difficile
Il castoro Adriano ha trovato in soffitta un vecchio calcolatore, che dispone di tre soli *registri* (aree di memoria in ciascuna delle quali può essere memorizzato un numero), chiamati R_1, R_2 e R_3.
Per programmare questa macchina, è necessario codificare in una opportuna sequenza le operazioni che essa dovrà svolgere. Le operazioni che la macchina può compiere sono di sei tipi:

- `Zero(i)`: memorizza 0 nel registro R_i;
- `Inc(i)`: incrementa di 1 il contenuto del registro R_i;
- `Dec(i)`: decrementa di 1 il contenuto del registro R_i;
- `Store(i, j)`: copia il contenuto del registro R_j nel registro R_i;
- `Jump(i, j, n)`: se R_i e R_j contengono lo stesso valore, allora salta all'operazione numero n
- `JumpNeg(i, j, n)`: se R_i e R_j contengono valori diversi, allora salta all'operazione numero n

dove i e j indicano il numero di un registro (e quindi possono essere 1, 2 o 3) e n indica il numero d'ordine di un'operazione nella sequenza.
Ad esempio, il programma seguente scambia il contenuto dei registri R_1 e R_2:

```
1:  Store(3,2)
2:  Store(2,1)
3:  Store(1,3)
```

Il castoro Adriano memorizza due numeri nei registri R_1 e R_2 e vuole scrivere un programma che calcoli la loro somma, memorizzandola in R_1 al posto del primo addendo.
Sapreste aiutarlo?

```
1:  Zero(3)
2:  Jump(1, 3, 6)
3:  Inc(2)
4:  Dec(1)
5:  JumpNeg(1,3,3)
6:  Store(1,2)
```

Quesiti da svolgere con l'aiuto del computer

Un programma affollato di eventi

Il parco di divertimento *Kangourandia* offre ai suoi visitatori un ricco calendario di eventi. Ogni giorno sono previsti film 4D, esibizioni degli *stunt-men* e dei tuffatori, commedie musicali, animazione dei bambini, parate delle *mascotte*, e altro ancora.
Avete solo un giorno a disposizione e volete assistere al massimo numero di eventi possibili. Naturalmente non è possibile assistere ad eventi che si svolgono in contemporanea (anche parzialmente).
Il vostro amico Ale Gridi, genio dell'informatica, vi ha suggerito di usare questa strategia per scegliere gli eventi cui assistere:

> Mettete in ordine gli eventi del programma, poi analizzateli uno alla volta rispettando quell'ordine: se l'evento in questione è il primo considerato oppure non si sovrappone agli altri eventi già scelti, allora assisterete all'evento, altrimenti lo scarterete e non lo prenderete più in considerazione.

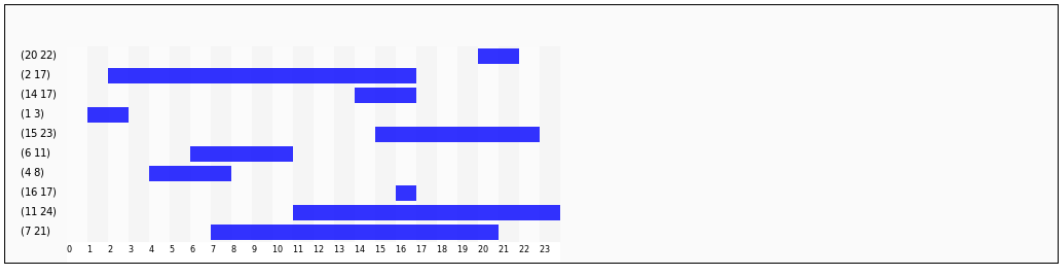
In questo modo, garantisce Ale Gridi, riuscirete ad assistere ad un numero di eventi che nessuno potrà battere.
L'unico problema è che non vi ricordate in che modo Ale Gridi vi ha suggerito di ordinare gli eventi e le possibilità sono molte!
Per individuare un buon criterio di ordinamento, avete a disposizione un programma che vi aiuterà a fare degli esperimenti. Per avviarlo lanciate il programma *Eventi*, vi comparirà una pagina con tre riquadri.

- Nel riquadro in alto potete inserire e modificare le informazioni relative agli orari degli eventi; per semplicità assumiamo che gli eventi inizino e finiscano tutti allo scoccare dell'ora. Per ogni evento, va indicata la coppia di ora di inizio e ora di fine nel formato `(inizio fine)`. Ad esempio la sequenza

  ```
  (20 22)(2 17)(14 17)(1 3)(15 23)(6 11)(4 8)(16 17)(11 24)(7 21)
  ```

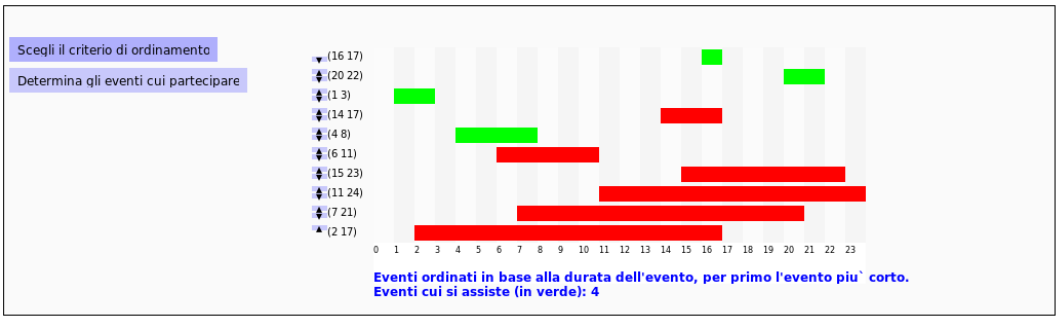
  rappresenta dieci eventi, uno che inizia alle 20 e finisce alle 22, uno che inizia alle 2 e finisce alle 17, uno che inizia alle 14 e finisce alle 17, e cosí via. Possono esserci piú eventi con lo stesso orario, in questo caso ci sarà una coppia `(inizio fine)` che si ripete nella sequenza.

- Nel riquadro centrale trovate una rappresentazione grafica degli eventi in base al loro orario.



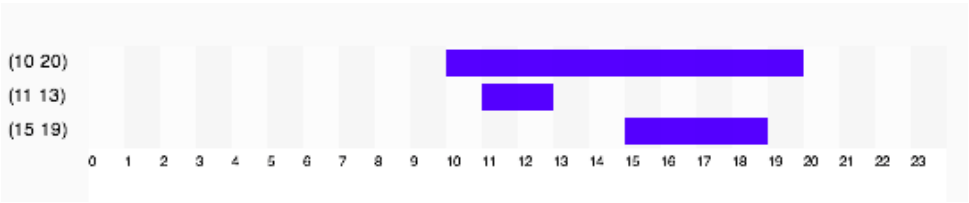
- Nel riquadro in basso potete modificare l'ordine in cui vengono visualizzati gli eventi elencati nella sequenza del primo riquadro. Con le freccine a sinistra potete spostare verso l'alto o il basso ciascun evento. Potete inoltre scegliere un criterio di ordinamento e ottenere la visualizzazione degli eventi

nell'ordine corrispondente. Infine, cliccando su "determina gli eventi cui partecipare", potete applicare, agli eventi cosí ordinati, la strategia suggerita da Ale Gridi, ottenendo in verde l'elenco degli eventi cui assistere. Ad esempio, ordinando in base all'ora di inizio (per primo quello che inizia per primo) si otterebbe questa selezione di eventi:



Per ciascuno dei seguenti criteri di ordinamento, scrivete se è un criterio *adatto* oppure no, nel senso che permette di selezionare il maggior numero di eventi possibili cui assistere. Inoltre, per ciascuno dei criteri che considerate non adatti, fornite un esempio (più semplice è l'esempio, maggiore sarà il punteggio) che mostra perché il criterio non è adatto. Per ciascuno dei criteri che considerate adatti, illustrate, nell'ultimo riquadro di questo quesito, il ragionamento che vi ha portato a questa conclusione.

1. Criterio: in base all'ora d'inizio, per primo quello che inizia prima.



In questo caso l'algoritmo di Ale Gridi sceglierebbe l'evento che inizia alle 10 e finisce alle 20, invece si potrebbe assistere ad entrambi gli altri eventi.

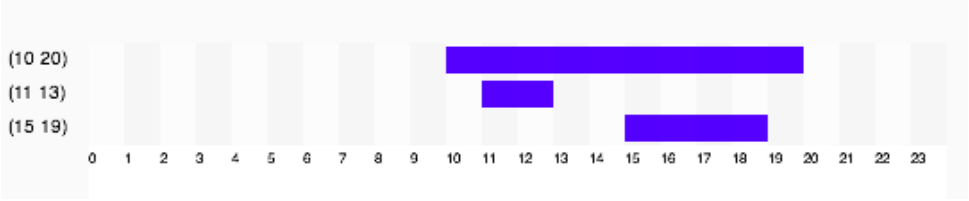
2. Criterio: in base all'ora di fine, per primo quello che finisce prima.

Questo è un criterio adatto!

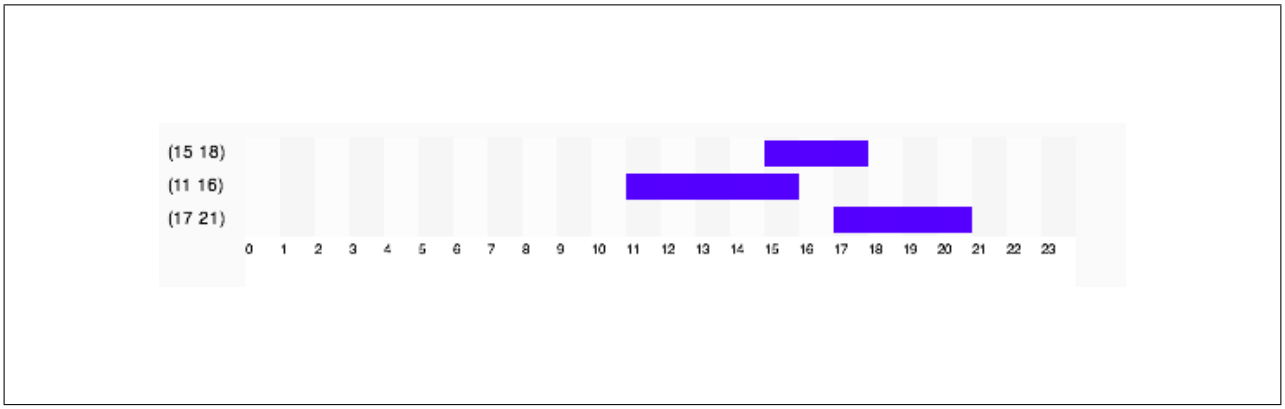
3. Criterio: in base all'ora d'inizio, per primo quello che inizia più tardi.

Questo è un criterio adatto!

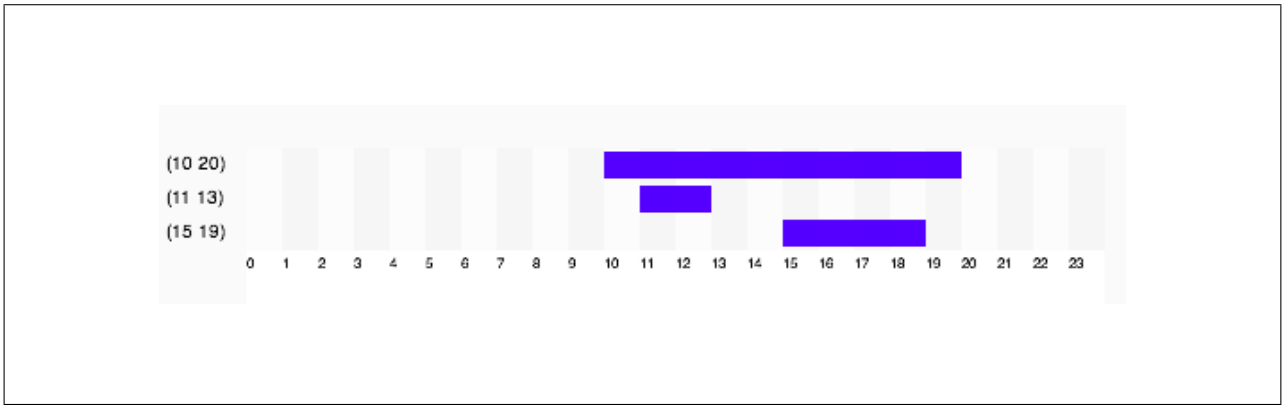
4. Criterio: in base all'ora di fine, per primo quello che finisce più tardi.



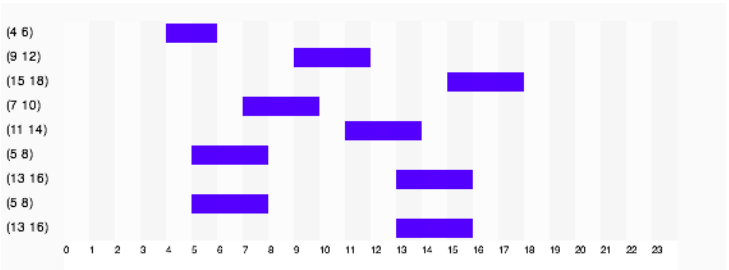
5. Criterio: in base alla durata dell'evento, per primo l'evento più corto.



6. Criterio: in base alla durata dell'evento, per primo l'evento più lungo.

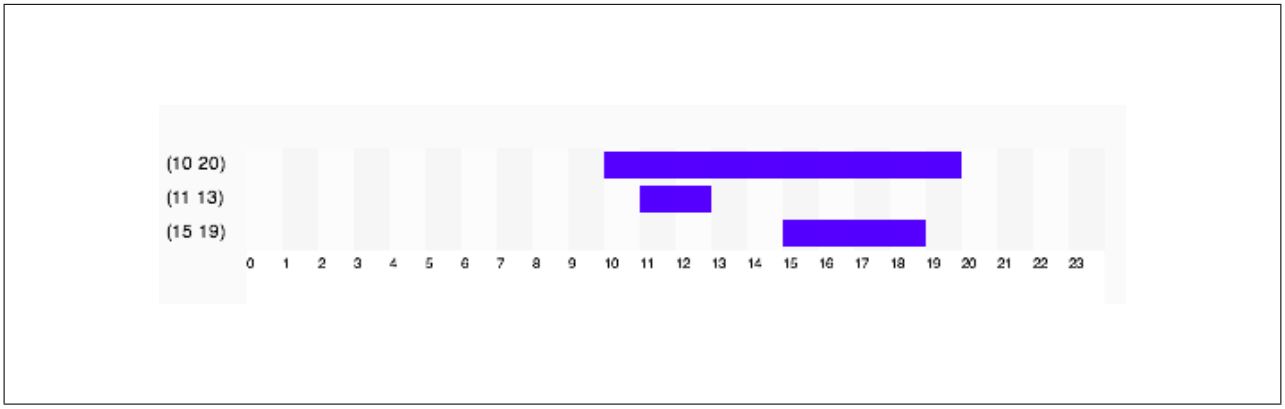


7. Criterio: in base al numero di sovrapposizioni, per primo l'evento che ha meno sovrapposizioni con gli altri eventi.



In questo caso la strategia di Ale Gridi sceglierebbe gli eventi con orari (4,6), (9,12) e (15,18). Tuttavia, invece di assistere all'evento (9,12), si potrebbe assistere agli eventi (7,10) e (11,14), per un totale di 4 eventi al posto di 3.

8. Criterio: in base al numero di sovrapposizioni, per primo l'evento che ha più sovrapposizioni con gli altri eventi.



Illustrate in modo convincente che i criteri (o il criterio) che avete considerato adatti sono imbattibili.

Ordinare gli eventi in base all'orario di fine, per primo quello che finisce prima, fornisce un criterio adatto, nel senso che garantisce sempre di assistere al massimo numero di eventi possibile. Per dimostrarlo, fissiamo un qualsiasi insieme di eventi. Immaginiamo che sia n il numero massimo di eventi cui si può assistere; che $\mathcal{E} = \{E_1, E_2, \ldots, E_n\}$ sia una qualunque soluzione (ovvero un insieme di eventi che non si sovrappongono) con n eventi; e che $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$ sia l'elenco degli eventi scelti invece usando la strategia di Ale Gridi in base all'ora di fine (per primo quello che finisce prima). Osservate che $m \leq n$ visto che si può assistere al piú a n eventi. Per comodità, sia in $\mathcal{E}$ che in $\mathcal{G}$ mettiamo gli eventi in ordine in base all'orario di fine, ovvero E_1 finisce prima di E_2, E_2 finisce prima di E_3, e cosí via, e lo stesso per $G_1, G_2, \ldots$. Allora naturalmente si ha che G_1 finisce prima (o assieme a) E_1; infatti se cosí non fosse, la strategia di Ale Gridi avrebbe portato a scegliere E_1 invece di G_1. Di conseguenza, E_2 inizia dopo che G_1 è finito. Quindi anche G_2 deve finire prima (o assieme a) E_2, altrimenti la strategia di Ale Gridi avrebbe portato a scegliere E_2 invece di G_2. E cosí via, fino ad arrivare a G_m che deve finire prima (o assieme a) E_m. Se m fosse piú piccolo di n, allora la strategia di Ale Gridi porterebbe ad aggiungere a $\mathcal{G}$ anche $E_{m+1}, E_{m+2} \ldots E_n$, e siccome questo non è il caso, allora possiamo concludere che m è proprio uguale a n, ovvero anche $\mathcal{G}$ contiene il numero massimo di eventi cui si può assistere.
Simmetricamente, anche ordinare gli eventi in base all'orario di inizio, per primo quello che inizia dopo, fornisce un criterio adatto; per dimostrarlo si può ragionare come per il criterio precedente, ma partendo dalla fine invece che dall'inizio.

Clickomania

Vi ricordate le regole di *Clickomania* che avete ricevuto ieri sera?
Vostro compito è realizzare un programma per giocare a Clickomania o, piú precisamente, un programma che aggiorni la parete di gioco in seguito a ogni mossa (click) del giocatore, utilizzando l'*ambiente di programmazione Blockly*.

Blockly

Blockly consente di *costruire dei programmi* a partire da blocchi colorati predefiniti, che si possono incastrare come in un puzzle.

Non c'è un solo programma che permette di giocare a Clickomania: i blocchi a disposizione possono essere combinati a formare tanti diversi programmi, tutti funzionanti. Tuttavia ciascun blocco ha un *costo* e il vostro punteggio sarà tanto maggiore quanto minore sarà la somma dei costi dei blocchi usati.

Una volta costruito un programma, si può iniziare il gioco cliccando sul bottone "Gioca". A questo punto, ad ogni click del giocatore (voi stessi, mentre testate il vostro programma!!), la parete verrà aggiornata in base alle istruzioni del programma costruito.

Blockly consente inoltre di vedere gli effetti dell'esecuzione del programma costruito, attraverso un'animazione, direttamente sulla parete di gioco, e permette di salvare fino a 4 programmi costruiti.

Interfaccia

Lanciate il programma *Clickomania*: vedrete una pagina composta di tre colonne.

- A sinistra trovate vari elementi.
  - La parete di mattoni.
  - Un cursore con ai lati una tartaruga e una lepre, per aumentare/diminuire la velocità dell'animazione.
  - Il bottone "Gioca" che consente di iniziare una partita e che si trasforma in "Reset" durante l'esecuzione del programma; cliccando su "Reset" è possibile iniziare una nuova partita.
  - Le icone relative a salvataggio/visualizzazione/caricamento dei programmi (o versioni di uno stesso programma): cliccando sul primo simbolo si può salvare il programma attualmente in costruzione, soffermandosi col mouse sul secondo simbolo si può visualizzare il programma salvato, cliccando sul terzo simbolo si può caricare il programma salvato in precedenza.
- Al centro trovate un menu di categorie tra cui scegliere i blocchi colorati; soffermandosi col mouse su un blocco si può leggere una descrizione di cosa fa il blocco.
- A destra trovate uno spazio bianco in cui potete trascinare i blocchi scelti dal menu, per costruire le istruzioni del vostro programma. Nell'angolo in basso a destra c'è il simbolo di un cestino nel quale possono essere trascinati i blocchi che non servono più.

  Durante l'aggiornamento della parete in seguito ad una mossa (click), i blocchi in esecuzione saranno via via evidenziati. Per fare degli esperimenti, tra una mossa e l'altra potete modificare il programma; però come soluzione del quesito accetteremo soltanto un programma che non debba essere modificato a seconda della mossa!

Blocchi a disposizione

I blocchi disponibili sono raggruppati nelle seguenti categorie (alcuni blocchi sono presenti in piú di una categoria): Aggiornamento parete, Caselle, Colori, Infezione, Logica, Cicli, Matematica, Variabili.
Ciascun blocco ha un costo che si può visualizzare soffermandosi col mouse sopra il blocco stesso. L'elenco completo dei blocchi con i loro costi è disponibile nel foglio allegato.

Esempi

Il programma più semplice che gioca a Clickomania è il seguente. Ad ogni click, vengono rimossi tutti i mattoni che fanno parte dell'area di contagio attorno alla casella cliccata, e gli spazi vuoti che si vengono a creare vengono eliminati dalla caduta dei mattoni che si trovano sopra.

Vediamo ora qualche esempio un po' più complicato (non è detto che siano di qualche utilità!!):

In questo caso, se la casella cliccata si trova nella riga di indice 5, allora viene infettato il mattone nella casella di coordinate $(1, 4)$ e viene colorato di rosso il mattone nella casella cliccata.

In questo caso, viene considerata la casella più in alto tra quelle contenenti un mattone infetto, assieme alla casella cliccata; se le due caselle contengono un mattone dello stesso colore, allora verrà rimosso il mattone che si trova nella casella di coordinate $(5, 4)$ e cadrà l'intera pila di mattoni posizionata a partire dalla casella di coordinate $(4, 4)$.

In questo caso, se ci sono più di tre mattoni infetti, ne verranno rimossi tre a caso; se ce ne sono tre o meno, verranno rimossi tutti.

Paleografia informatica

Questa prova va svolta su carta. Se volete, potete usare il Web per fare delle ricerche.

Essendo nato nel 4016 aveva ormai quasi quarant'anni, e da tutti era considerato un archeologo esperto, uno dei luminari nel suo campo, la *paleografia informatica*. Eppure non aveva mai partecipato prima di allora a una campagna di scavi che lo emozionasse tanto: sentiva che quello che stavano scoprendo avrebbe gettato una nuova luce sul passato dell'umanità e sui modi misteriosi con cui gli uomini, piú di due millenni prima, interagivano con i computer. Ben prima dell'avvento della programmazione olistica, prima del trasferimento osmotico della conoscenza, prima della costruzione della PCA (Pura Coscienza Algoritmica), gli umani usavano i computer in modo estremamente rudimentale, programmandone il funzionamento direttamente, e lo facevano attraverso misteriosi linguaggi (li chiamavano **"linguaggi di programmazione"**, appunto) il cui significato e la cui sintassi erano ancora in gran parte avvolti nel mistero.
Stava pensando a tutto questo quando, con la mano tremante, si avviava ad esaminare il contenuto della memoria USB trovata il giorno prima, alla fine di una lunga giornata sul campo, sotto il sole cocente dell'antica Silicon Valley, in California: stavano scavando a Mountain View, in quella che era stata la sede di un opificio informatico nato all'inizio del terzo millennio e chiamato Gogol. Era in quell'area che, il giorno prima, avevano recuperato questo piccolo oggetto metallico. L'aveva riconosciuto subito, ne aveva già visti centinaia: era una memoria USB, forse posseduta da qualche dipendente di Gogol che la usava per trasportare i programmi da un computer all'altro, o magari per lavorarci da casa, durante la notte. Quella memoria conteneva con ogni probabilità del codice sorgente di qualche *programma informatico*.
Ora l'oggetto metallico era nel suo laboratorio; maneggiandolo con grande cautela lo aveva inserito nel lettore speciale (a prova di polvere e di contaminazione) che lui stesso aveva costruito negli anni e che gli permetteva di analizzare il contenuto di memorie di quel periodo. Dopo qualche istante, nello schermo olografico, iniziarono a comparire i piccoli involti che rappresentavano i "*file*" (cosí, nell'antichità, erano chiamati i contenitori dei codici sorgente dei programmi). Era bellissimo: c'erano decine di programmi tutti a sua disposizione, e aprendoli avrebbe potuto scoprire che cosa facevano e forse capire meglio i linguaggi in cui erano scritti e di cui ancora nessuno aveva compreso quasi nulla.
Prese in mano il pacchetto olografico di uno dei *file*, e ne svolse i lembi per aprirlo e vederne il contenuto. Quello che segue fu ciò che apparve, nella penombra della stanza.
Sapete identificare a) il linguaggio di programmazione e b) il contesto storico o informatico in cui sono stati creati i seguenti frammenti di codice?

1. Ecco un frammento:

```
/*
 * You are not expected to understand this.
 */
if(rp->p_flag & SSWAP) {
        rp->p_flag &= ~SSWAP;
        aretu(u.u_ssav);
}
```

   Di che si tratta?

Linguaggio di programmazione: C. Si tratta di un frammento del codice sorgente di UNIX, il sistema operativo programmato da Ken Thompson. Il commento si riferisce a una parte dello *scheduler* (il componente che permette di avere piú di un programma in esecuzione) particolarmente ostica da comprendere per chi non fosse totalmente consapevole di tutti i dettagli implementativi.

2. Ecco un altro frammento:

```
      INTEGER A,B,C
      READ(5,501) A,B,C
  501 FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C))
      WRITE(6,601) A,B,C,AREA
  601 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,12HSQUARE UNITS)
      STOP
      END
```

Di che si tratta?

Linguaggio di programmazione: Fortran. Un tipico esempio di programmazione di una ben nota formula matematica: il calcolo dell'area di un triangolo secondo il metodo di Erone.

3. Ecco un altro frammento:

```
s''$/=\2048;while(<>){G=29;R=142;if((@a=unqT="C*",_)[20]&48){D=89;_=unqb24,qT,@
b=map{ord qB8,unqb8,qT,_^$a[--D]}@INC;s/...$/1$&/;Q=unqV,qb25,_;H=73;O=$b[4]<<9
|256|$b[3];Q=Q>>8^(P=(E=255)&(Q>>12^Q>>4^Q/8^Q))<<17,O=O>>8^(E&(F=(S=O>>14&7^O)
^S*8^S<<6))<<9,_=(map{U=_%16orE^=R^=110&(S=(unqT,"\xb\ntd\xbz\x14d")[_/16%8]);E
^=(72,@z=(64,72,G^=12*(U-2?0:S&17)),H^=_%64?12:0,@z)[_%8]}(16..271))[_]^((D>>=8
)+=P+(~F&E))for@a[128..$#a]}print+qT,@a}';s/[D-HO-U_]/\$$&/g;s/q/pack+/g;eval
```

Di che si tratta?

Linguaggio di programmazione: Perl. Si tratta di una versione Perl molto compatta (e perciò sufficientemente breve da poter essere stampata su una maglietta) del programma capace di decrittare il contenuto di un DVD video in cui è stato usato il Content-Scrambling System (CSS), un sistema che — nato per impedire le copie illegali — finisce talvolta per impedire anche gli usi legali: per esempio la visione di un DVD acquistato in una regione commerciale differente. All'inizio degli anni 2000, il programma subí molte restrizioni alla sua diffusione: per superarle si pubblico il codice in forme tutelate dalla libertà di espressione come i libri o... le magliette.

4. Ecco un altro frammento:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.
>---.+++++++..+++.>>.<-.<.+++.------.--------.>>+.>++.
```

Di che si tratta?

Linguaggio di programmazione: Brainfuck. Brainfuck è un linguaggio di programmazione inventato nel 1993 da Urban Müller come esempio di linguaggio difficilmente comprensibile ai lettori umani e pure totalmente equivalente a quelli tradizionali. Il programma ritrovato dal nostro paleografo stampa il messaggio 'Hello World!'.

5. Ecco un altro frammento:

```
10 Print Chr$(205.5+RND(1)); : GOTO 10
```

Di che si tratta?

Linguaggio di programmazione: Basic. Il Basic è un linguaggio di programmazione inventato nel 1964 da John Kemeny e Thomas Kurtz per diffondere la conoscenza della programmazione presso un pubblico meno specialistico di quello cui all'epoca veniva insegnato a programmare in Fortran o Cobol. Divenne molto popolare quando fu installato sui primi *personal computer*, in particolare l'Altair (il primo prodotto Microsoft) e l'Apple II. Il programma trovato dal nostro paleografo era un classico dei primi passi nella programmazione del Commodore 64: l'effetto è la stampa di griglie casuali che tassellano tutto lo schermo.

6. Ecco un altro frammento:

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
```

```
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Di che si tratta?

Linguaggio di programmazione: C. Si tratta di un disastroso "bug" (errore di programmazione, dei i due goto fail di seguito il secondo viene *sempre* eseguito!) che provocò nel 2014 parecchi grattacapi per chi cura la sicurezza dei sistemi informatici: infatti in particolari situazioni ciò permette a un attaccante di evitare il controllo dei certificati normalmente necessario per iniziare transazioni potenzialmente pericolose.

7. Ecco un altro frammento:

```
primes = sieve [2..]
sieve (p : xs) = p : sieve [x | x <- xs, x `mod` p > 0]
```

Di che si tratta?

Linguaggio di programmazione: Haskell. Haskell (dal nome del logico Haskell Curry) è un linguaggio di programmazione funzionale in cui la scrittura dei programmi è molto vicina alla loro espressione in linguaggio matematico. Il programma denota una specie di "Crivello di Eratostene" per identificare i numeri primi, quei numeri, cioè, divisibili solo per 1 e per sè stessi e che possono essere trovati eliminando dall'insieme degli interi maggiori di 2 tutti quelli divisibili per i numeri primi trovati fino a quel momento (dei quali 2 è il primo).

8. Ecco un altro frammento:

```
((lambda (x)
       (list x (list (quote quote) x)))
     (quote
        (lambda (x)
          (list x (list (quote quote) x)))))
```

Di che si tratta?

Linguaggio di programmazione: Lisp. Il Lisp è uno dei primi linguaggi di programmazione (il primo fu il Fortran), definito nel 1958 da John McCarthy. Fra i motivi che lo rendono particolarmente interessante c'è il fatto che la forma in cui i programmi vengono scritti è la stessa che usa l'interprete o il compilatore per realizzarne l'esecuzione. Questa sua natura "riflessiva" lo rende particolarmente adatto per scrivere programmi, come quello ritrovato dal nostro paleografo, che producono sè stessi, i cosiddetti *quine*, dal nome del filosofo Willard Van Orman Quine.

9. Ecco un altro frammento:

```
mortal(X) :-
  human(X).
human(socrate).
?- mortal(socrate).
yes
```

Di che si tratta?

Linguaggio di programmazione: Prolog. Il Prolog è un linguaggio di programmazione basato sulla logica del primo ordine (quella che contiene solo predicati logici e quantificatori): il programma rappresenta l'inferenza possibile tramite il sillogismo "Tutti gli uomini sono mortali; Socrate è un uomo; ergo: Socrate è mortale."

10. Ecco un altro frammento:

```
tell application "Finder"
    make new folder at desktop
end tell
```

Di che si tratta?

Linguaggio di programmazione: Applescript. Applescript è un linguaggio di programmazione creato con lo scopo di di controllare le applicazioni al fine automatizzare delle operazioni ripetitive all'interno del sistema operativo Mac OS. Una delle caratteristiche di questo linguaggio è quella di essere facilmente leggibile in quanto i programmi sono scritti utilizzando una formulazione abbastanza simile a quella delle frasi del linguaggio naturale: il programma mostrato crea una nuova cartella all'interno della scrivania.

11. Ecco un altro frammento:

```
HAI
CAN HAS STDIO?
VISIBLE "HAI WORLD!"
KTHXBYE
```

Di che si tratta?

Linguaggio di programmazione: LOLCODE. LOLCODE fa parte dei cosiddetti "linguaggi esoterici", in cui la sintassi è pensata per ottenere dei programmi particolarmente poco facili da leggere e interpretare da parte delle persone (e per questo motivo anche poco utilizzati in pratica). In particolare, le istruzioni di LOLCODE sono ottenute storpiando parole del linguaggio naturale: per esempio, nel listato mostrato sopra (stampa il messaggio "HAI WORLD!") si può notare l'istruzione KTHXBYE, usata per terminare un programma, che corrisponde a "OK, thanks, bye".

12. Ecco un altro frammento:

```
keeplooping=1;
while [[ $keeplooping -eq 1 ]] ; do
  read quitnow
  if [[ "$quitnow" = "yes" ]] ; then
    keeplooping=0
  fi
  if [[ "$quitnow" = "q" ]] ; then
    break;
  fi
done
```

Di che si tratta?

Linguaggio di programmazione: Shell script. Nel contesto dei sistemi operativi derivati da Unix, il termine *shell* indica un interprete di comandi tipicamente utilizzato in modo interattivo tramite un terminale. Lo stesso interprete serve però anche a eseguire dei veri e propri programmi che tipicamente aiutano ad automatizzare operazioni ripetitive. Il frammento di programma mostrato serve a chiedere all'utente di immettere del testo, ripetendo la richiesta fino a che il testo immesso non è uguale a "yes" oppure a "q".