Crittografia LWF (`lwf`)

Luca e William devono sovente scambiarsi delle segretissime informazioni riguardo alle selezioni territoriali, sotto forma di numeri interi N. Per evitare di essere scoperti, hanno quindi deciso di inventare un nuovo codice crittografico, che hanno chiamato *codice Luca-William-Fibonacci* (LWF).

In questo codice, ogni numero intero N viene tradotto in una sequenza $s_0 s_1 \dots s_k$ di cifre binarie `'0'` e `'1'`, di cui l'ultima è un `'1'`, in maniera tale che:

$$N = \sum_{i=0}^{k} s_i \cdot F_i$$

dove F_i è il numero di Fibonacci i-esimo. Più informalmente, una cifra 1 in posizione i nella sequenza indica che il numero di Fibonacci i-esimo fa parte della somma che ricostruisce il numero N.

> ☞ La sequenza dei numeri di Fibonacci è definita in maniera *ricorsiva*: i primi due termini della sequenza sono $F_0 = 1$ e $F_1 = 1$, mentre ognuno dei successivi viene calcolato sommando i due precedenti $F_i = F_{i-1} + F_{i-2}$.

Per esempio, consideriamo la sequenza `1011001` di lunghezza $k = 7$. Visto che i primi 7 numeri di Fibonacci sono:

$$1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13$$

il numero N corrispondente è pari a $1 + 2 + 3 + 13 = 19$.

Luca ha già implementato l'algoritmo di decodifica (descritto come sopra), che da una sequenza di cifre binarie ricostruisce il numero N. Tuttavia William è ancora in alto mare con l'algoritmo di codifica, che dato un numero N dovrebbe produrre una sequenza di cifre binarie corrispondente. Implementalo tu!

Dati di input

Il file `input.txt` è composto da un'unica riga contenente l'unico intero N.

Dati di output

Il file `output.txt` deve essere composto da un'unica riga contenente una sequenza di cifre binarie che termina con `'1'` corrispondente ad N.

Assunzioni

- $1 \leq N \leq 1\,000\,000$.
- Potrebbero esserci più sequenze di cifre ugualmente valide.

Esempi di input/output

input.txt	output.txt
19	1011001
9	11101

Spiegazione

Il **primo caso di esempio** è quello discusso nel testo.

Nel **secondo caso di esempio**, 9 può essere ottenuto sia come $1+1+2+5$ (come nell'output di esempio), oppure come $1+3+5$ (`10011`) e $1+8$ (`100001`).

Soluzione

```cpp
// Soluzione di lwf
// Autore: Edoardo Morassutto
// Complessità: O(N)
// Breve spiegazione:
//   Calcolo i numeri di Fibonacci fino ad N (incluso) e dal piu grande
//   li prendo gready se ci stanno in N e diminuisco.

#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>

using namespace std;

int fib[50];

int main() {
    ifstream in("input.txt");
    ofstream out("output.txt");

    int N;
    in >> N;

    fib[0] = 1;
    fib[1] = 1;
    int i;
    for (i = 2; fib[i-1]+fib[i-2] <= N; i++) {
        fib[i] = fib[i-1]+fib[i-2];
    }
    i--;

    string s;
    while (i >= 0) {
        if (N >= fib[i]) {
            s += '1';
            N -= fib[i];
        } else {
            s += '0';
        }
        i--;
    }
    reverse(s.begin(), s.end());
    out << s << endl;
}
```

Appetito aracnide (`tecla`)

Ape Maya è rimasta intrappolata in un nodo della tela di Tecla, un ragno molto temuto tra le api dell'alveare. Tecla si affretta ad afferrarla ma, quando giunge su quel nodo, si accorge di non avere appetito, e dice "BLEAH". Va detto che l'appetito dei ragni è molto particolare: ogni volta che percorrono un filamento della loro rete, essi invertono lo stato del loro stomaco tra "SLURP" e "BLEAH". Tecla deve quindi farsi un giretto nella rete sperando di tornare da Maya in stato "SLURP".

La tela di Tecla è composta da N nodi (numerati da 0 a $N-1$) connessi tra loro da M filamenti. Tecla e Ape Maya all'inizio si trovano entrambe nel nodo 0, e ogni filamento può essere attraversato da Tecla in entrambe le direzioni. Aiuta Tecla ad individuare una passeggiata funzionale al buon appetito!

Dati di input

Il file `input.txt` è composto da $M+1$ righe, contenenti:

- Riga 1: gli interi N ed M, il numero di nodi e di filamenti della tela.
- Riga 2..$M+1$: due interi separati da spazio u, v; dove u e v identificano i due nodi ai capi del filamento i-esimo.

Dati di output

Il file `output.txt` deve essere composto da due righe, contenenti:

- Riga 1: il numero di spostamenti L che Tecla deve compiere nella sua passeggiata.
- Riga 2: $L+1$ numeri separati da uno spazio, di cui il primo e l'ultimo devono essere 0 (nodo di partenza e di arrivo), e gli altri sono i nodi come visitati da Tecla nell'ordine (e possono avere ripetizioni).

Assunzioni

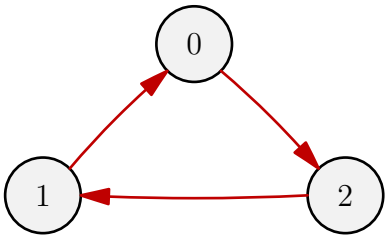
- $1 \leq M, N \leq 30$.
- In ogni filamento, $u \neq v$ e sono entrambi compresi tra 0 e $N-1$.
- Si garantisce l'esistenza di una soluzione: Ape Maya è spacciata!

Esempi di input/output

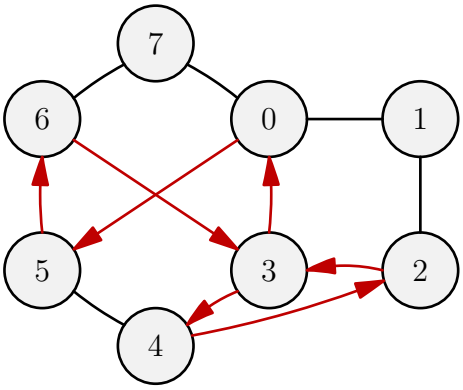
input.txt	output.txt
3 3 0 1 1 2 2 0	3 0 2 1 0
8 12 0 1 1 2 2 3 3 0 2 4 3 4 4 5 5 6 6 7 7 0 0 5 6 3	7 0 5 6 3 4 2 3 0

Spiegazione

Nel **primo caso di esempio**, la tela di Tecla è come nella figura seguente, dove il percorso da seguire è evidenziato in rosso:



Nel **secondo caso di esempio**, la tela e il percorso sono:



Soluzione

```
/* solutore oddcycle
   Romeo 2017-03-15
*/

#include <iostream>
#include <cassert>
#include <vector>

#define DEBUG 0

using namespace std;

const unsigned MAXN = 30;
int seen[MAXN];
int sex[MAXN];
int cycle[MAXN], posW = 0;

int N, M, L;
vector<int> adj[MAXN];

bool odd_cycle(int node, int mysex) {
  if ( seen[node] ) {
    if ( sex[node] != mysex ) {
      cycle[posW++] = node;
      L = posW;
      return true;
    }
    else
      return false;
  }
  seen[node] = 1;
  sex[node] = mysex;
  cycle[posW++] = node;
  for (int next: adj[node])
    if ( odd_cycle(next, 1-mysex) )
      return true;
  posW--;
  return false;
}

int main() {
    assert(freopen("input.txt", "r", stdin));
    assert(freopen("output.txt", "w", stdout));

    cin >> N >> M;

```

```
47      for (int i=0; i<M; i++) {
48          int a, b;
49          cin >> a >> b;
50          adj[a].push_back( b );
51          adj[b].push_back( a );
52      }
53
54      assert( odd_cycle(0, 0) );
55
56      int visitedTwice = cycle[L-1];
57      bool repeat = false;
58      for (int i=L-2; i >= 0; i--) {
59          if ( repeat )
60              cycle[L++] = cycle[i];
61          if ( cycle[i] == visitedTwice )
62              repeat = true;
63      }
64
65      cout << L-1 << endl;
66      for (int i=0; i<L; i++)
67        cout << cycle[i] << " ";
68      cout << endl;
69
70      return 0;
71  }
```

Sport intellettuali (scommessa)

Romeo è un grande appassionato di sport intellettuali, e adora ritrovarsi con gli amici per seguire le competizioni internazionali più avvincenti di questo tipo. Di recente, il gruppo di amici si è appassionato a uno sport molto particolare. In questo gioco, un mazzo di carte numerate da 0 a $N-1$ (dove N è dispari) viene prima mescolato, e poi le carte vengono affiancate in linea retta sul tavolo. Ai telespettatori, per aumentare la suspence, vengono mostrati i numeri delle carte $C_0, C_1, \ldots, C_i, \ldots, C_{N-1}$ nell'ordine così ottenuto. A questo punto i giocatori[1] possono scoprire due carte disposte consecutivamente sul tavolo, e prenderle nel solo caso in cui queste due carte *abbiano somma dispari*. Se queste carte vengono prese, le altre vengono aggiustate quanto basta per riempire il buco lasciato libero. Il gioco prosegue quindi a questo modo finché nessun giocatore può più prendere carte.

Romeo e i suoi amici, per sentirsi più partecipi, hanno oggi deciso di fare un "gioco nel gioco": all'inizio della partita, scommettono su quali carte pensano rimarranno sul tavolo una volta finita la partita. Aiuta Romeo, determinando quali carte *potrebbero* rimanere sul tavolo alla fine del gioco!

> ☞ Una carta *potrebbe* rimanere sul tavolo a fine gioco, se esiste una sequenza di mosse (rimozioni di coppie di carte consecutive con somma dispari) tale per cui dopo di esse nessuna altra mossa è possibile (il gioco e finito) e la carta suddetta è ancora sul tavolo.

Dati di input

Il file `input.txt` è composto da 2 righe, contenenti:

- Riga 1: l'unico intero N.
- Riga 2: gli N interi C_i separati da spazio, nell'ordine in cui sono disposti sul tavolo.

Dati di output

Il file `output.txt` deve essere composto da due righe, contenenti:

- Riga 1: il numero di diverse carte K che *potrebbero* rimanere sul tavolo a fine partita.
- Riga 2: i K interi che identificano le carte che *potrebbero* rimanere sul tavolo a fine partita.

Assunzioni

- $1 \leq N \leq 100$.
- N è sempre un numero dispari.
- $0 \leq C_i \leq N-1$ per ogni $i = 0 \ldots N-1$.
- Ogni numero tra 0 e $N-1$ compare esattamente una volta nella sequenza dei C_i.

[1] Seguendo un elaborato ordine di gioco che non rientra nei margini di questo problema.

Esempi di input/output

input.txt	output.txt
3 1 2 0	1 0
11 1 0 2 6 4 5 3 9 8 10 7	2 2 8

Spiegazione

Nel **primo caso di esempio**, l'unica mossa possibile è eliminare le carte 1 e 2 per cui rimane sul tavolo necessariamente la carta 0.

Nel **secondo caso di esempio** sono invece possibili diverse sequenze di mosse. Una delle sequenze che lasciano la carta 2 è la seguente:

1 0 2 6 ~~4 5~~ 3 9 8 10 7
~~1 0~~ 2 6 3 9 8 10 7
2 ~~6 3~~ 9 8 10 7
2 9 8 ~~10 7~~
2 ~~9 8~~
2

Una delle sequenze di mosse che lasciano la carta 8 è la seguente:

~~1 0~~ 2 6 4 5 3 9 8 10 7
2 6 ~~4 5~~ 3 9 8 10 7
2 6 3 9 8 ~~10 7~~
2 ~~6 3~~ 9 8
~~2 9~~ 8
8

Non esistono invece sequenze di mosse che lasciano alcuna delle altre carte.

Soluzione

```cpp
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

const int MAX_N = 150;

int nums[MAX_N];
int parity[MAX_N];
// unibili finendo ad i-1 se prefix[i] == 0
int prefix[MAX_N];
// unibili partendo da i+1 se suffix[i] == 0
int suffix[MAX_N];

int main() {
    ifstream in("input.txt");
    ofstream out("output.txt");
    int N;
    in >> N;

    for (int i = 0; i < N; i++) {
        in >> nums[i];
        parity[i] = (nums[i] % 2 == 0) ? 1 : -1;
    }

    for (int i = 0; i < N-1; i++)
        prefix[i+1] = prefix[i]+parity[i];
    for (int i = N-1; i >= 1; i--)
        suffix[i-1] = suffix[i]+parity[i];

    vector<int> sol;
    for (int i = 0; i < N; i++)
        if (prefix[i] == 0 && suffix[i] == 0)
            sol.push_back(nums[i]);

    out << sol.size() << endl;
    for (int n : sol)
        out << n << ' ';
    out << endl;
}
```