Halting Problem

"The halting problem" este problema care constă în a decide, pentru un program si un input date, dacă programul apelat pentru inputul respectiv se va termina ("to halt" = a (se) opri) sau va rula la infinit. Considerăm că programul despre care vrem să știm dacă se termină este rulat pe o mașină Turing sau pe un model computațional care este Turing-complete (un limbaj de programare care este suficient de general cât să fie echivalent cu o mașină Turing).

Halting Problem este nedecidabilă!, ceea ce înseamnă că nu există un algoritm care să răspundă la Halting Problem pentru toate perechile posibile de program-input. Explicația simplă este următoarea:

```
/// Presupunem că avem o funcție "bool halts(program p, input i)"
/// care returnează True când p este apelat cu inputul i și se
/// termină, și False când rulează la infinit.

/// Apoi avem următorul program:
void contradiction(string input) {
    if (halts(contradiction, input)) {
        while (true)
            continue;
    }
    return;
}
```

Practic, funcţia *halts* joacă rolul de algoritm "magic" care rezolvă halting problem, iar funcţia *contradiction* se foloseşte de ea însăşi ca să apeleze *halts*, *halts* prezice dacă *contradiction* se termină sau nu, iar apoi *contradiction* face opusul a ce a prezis *halts*. Acest contraexemplu nu poate fi învins de niciun algoritm, prin urmare halting problem este nedecidabilă.

Dificultatea problemei constă în faptul că procedura de decizie TREBUIE să funcţioneze pentru absolut toate programele şi inputurile.

Busy Beaver

"Halting state" (starea de oprire) este starea în care o mașină Turing intră după ce și-a terminat computarea. O mașină Turing se numește totală dacă ajunge într-o stare de oprire pentru orice input.

Ne imaginăm o mașină Turing care nu are stări de acceptare sau de respingere, ci în locul acesta are pur și simplu stări de oprire.

Busy Beaver game, Busy Beaver problem sau simplu Busy Beaver este o aplicație a Halting Problem.

Informal, Busy Beaver constă în găsirea unui program de o dimensiune dată, care se termină și produce cel mai mare output posibil. Ne gândim că mașina noastră are alfabetul *{0,1}.* Regulile sunt:

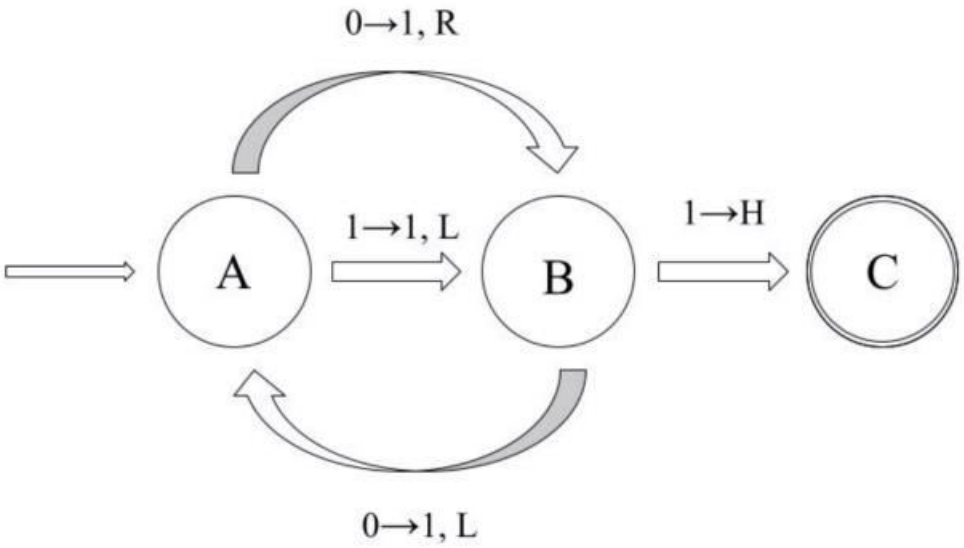
- mașina trebuie să aibă cel mult 2 stări în afară de starea de oprire
- banda conține inițial doar zerouri

Scopul/Provocarea este să se creeze tranziții astfel încât să se obțină ca output pe bandă cel mai mare număr de cifre de 1 posibil (și de asemenea mașina să ajungă în starea de oprire).

Această versiune se numește 2-state Busy Beaver. Acest lucru poate fi generalizat astfel: dacă în prima regulă permitem ca mașina să aibă *n* stări, atunci jocul se va numi *n-state Busy Beaver.* Numărul de 1 sau timpul de rulare al *n-state Busy Beaver* nu este calculabil!

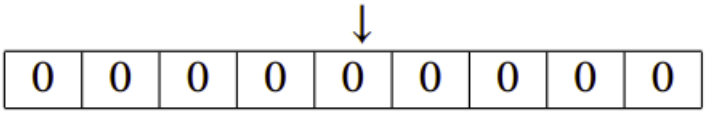
Pentru joc în sine, există 2 modalități de "scor" cunoscute. Prima este cea menționată, numărul de 1 din output, notată Σ(n). Cea de-a doua este numărul maxim de pași pe care mașina îl face înainte de a se opri, notat S(n).

Exemplu:

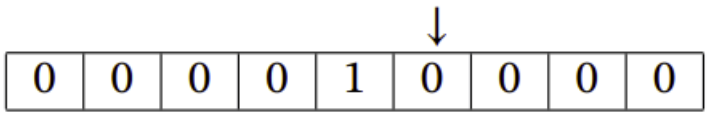


Vom parcurge pașii mașinii de mai sus (2-state).

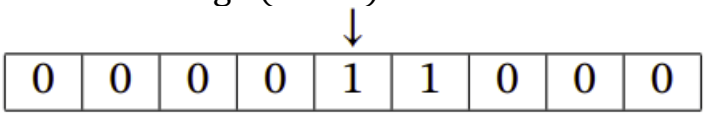
Banda este inițial vidă.



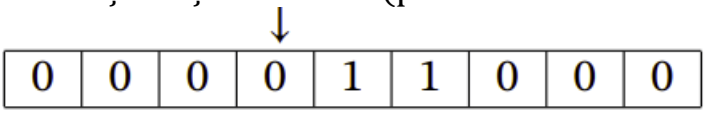
1. Se citește 0 și se scrie 1 (mergem din A în B). Tranziția este "0->1,R" cu semnificația "citește 0, înlocuiește-l cu 1, și mută pointerul de pe bandă la dreapta ("Right").



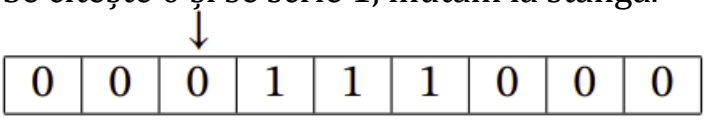
2. Se citește 0 și se scrie 1 (mergem din B în A), mutând pointerul de pe bandă la stânga ("Left").



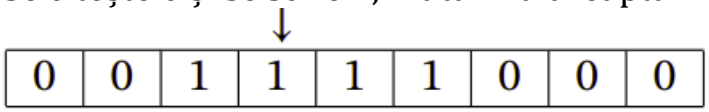
3. Se citește 1 și se scrie 1 (practic rămâne neschimbat), mutăm la stânga.



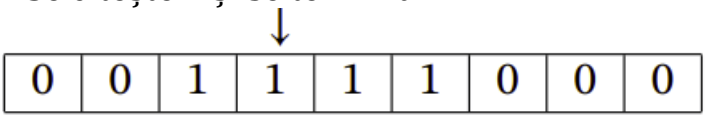
4. Se citește 0 și se scrie 1, mutăm la stânga.



5. Se citește 0 și se scrie 1, mutăm la dreapta.



6. Se citește 1 și se termină.



Numărul de 1 de pe bandă este 4 => **Σ(2) = 4.**

Numărul de pași este 6 => **S(2) = 6.**

Atât Σ(n), cât și S(n), sunt necalculabile, pe motiv că cresc mai rapid decât orice funcție calculabilă. Din acest motiv, este **nedecidabil** dacă o mașină Turing dată este un Busy Beaver!