1 Markov is coming ... *(40p)*

1.1 Enunț

După ce a obținut note foarte bune la materiile de pe semestrul întâi (în mod special la programare și la algebră liniară), Mihai s-a decis să își ocupe timpul cu o problemă interesantă, pentru care își propune să găsească o soluție cât mai performantă.

Fiind pasionat de tehnologie, el și-a cumpărat un robotel pe care îl poate programa după cum dorește. În timpul liber, a mai construit și un mic labirint pe care intenționa să testeze roboțelul.

Acum, Mihai dorește să plaseze roboțelul undeva în labirint și să-l programeze astfel încât să aleagă, la fiecare pas, cea mai bună direcție pentru a reuși să evadeze. Roboțelul se consideră evadat dacă găsește una din ieșirile consancrate ale labirintului.

Pentru că Mihai nu este încă familiarizat cu algoritmii avansați de căutare (căci aceștia se învață abia în anul 2), își propune să plece de la o problemă mai simplă, iar apoi să implementeze un algoritm simplu, astfel: Având la dispoziție un labirint și o poziție de plecare, care este probabilitatea ca robotul meu **să ajungă într-o zonă de câștig**, dacă la fiecare pas el **alege o direcție aleatoare de deplasare** dintre cele disponibile? De asemenea, cum aș putea folosi probabilitățile determinate anterior pentru determinarea unei căi pentru robot prin labirintul meu, într-o manieră **mai eficientă** decât o căutare exhaustivă ?

1.2 O explicație vizuală

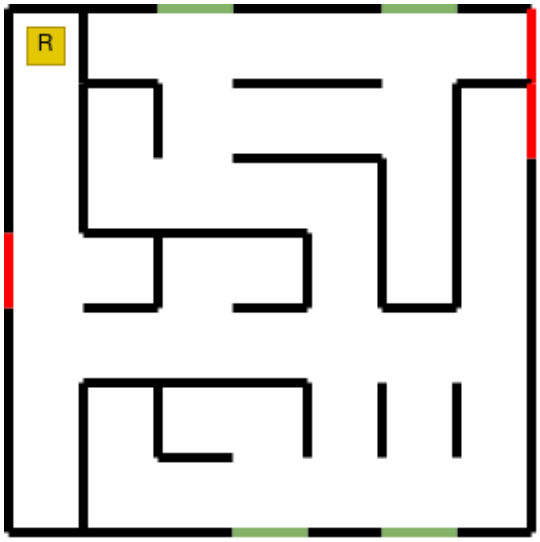

Figura 1: Exemplu de labirint

Să presupunem labirintul de mai sus, foarte simplu reprezentat sub forma unei matrice pătratice, în care poziția de plecare a roboțelului nostru este colțul din stânga–sus. Considerăm că punctul de pornire are coordonatele $(1, 1)$. Robotul va alege la fiecare pas să se mute într-o nouă celulă a labirintului pe care nu a vizitat-o anterior, putând să se deplaseze câte o pătrățică în sus, în jos, la stânga sau la dreapta, însă nu are voie să meargă pe diagonală.

Spre exemplu, din poziția de start, acesta va vizita celula de coordonate $(2, 1)$, apoi celulele de coordonate

$(3, 1)$ și $(4, 1)$, iar apoi va alege între cele 2 celule adiacente pe cea care are probabilitatea mai mare de a ajunge la o ieșire câștigătoare din labirintul nostru. Ieșirile în cadrul problemei studiate sunt de 2 tipuri:

- **Ieșiri care duc la câștig.** Ele sunt ieșirile marcate cu verde pe figura de mai sus, și se suprapun tot timpul cu limita superioară, respectiv cu cea inferioară ale labirintului. În momentul în care robotul alege să iasă din labirint pe una dintre aceste ieșiri, se poate spune că acesta a câștigat (probabilitatea de câștig este 1).
- **Ieșiri care duc la pierderea jocului.** Ele sunt ieșirile marcate cu roșu pe figura de mai sus, și se suprapun tot timpul cu limitele laterale (stânga / dreapta) ale labirintului. În momentul în care robotul alege să iasă din labirint prin una dintre aceste ieșiri, se poate spune că acesta a pierdut jocul (probabilitatea de câștig în această stare este 0).

În interiorul labirintului, pot exista și pereți care să nu permită trecerea roboțelului între 2 celule adiacente din punct de vedere spațial. Spre exemplu, în figura de mai sus, nu se poate merge din celula $(1, 1)$ direct în celula $(1, 2)$. Astfel, în orice moment de timp, robotul va avea un număr de cel mult 4 alegeri corespunzătoare direcțiilor în care se poate deplasa, din care acesta o poate alege complet aleator (cu probabilitate egală) pe oricare dintre ele.

1.3 Referințe teoretice

Pentru modelarea situației prezentate vom folosi lanțuri de probabilități, cunoscute sub denumirea de **lanțuri Markov**. Lanțurile Markov sunt deosebit de utile în teoria probabilităților, având aplicații în domenii precum economie, fiabilitatea sistemelor dinamice, respectiv în algoritmii de inteligență artificială. **Google Page Rank** este, de asemenea, o formă modificată a unui lanț Markov.

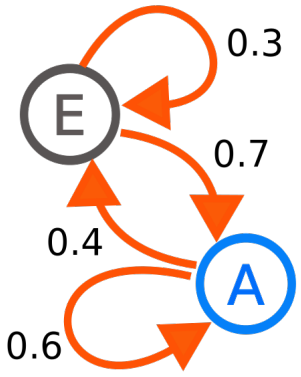

Figura 2: Ilustrație simplă a unui lanț Markov

Structura de date de bază în cadrul lanțurilor Markov este graful orientat, în care nodurile reprezintă stările, iar fiecare muchie existentă între două stări reprezintă o probabilitate nenulă de trecere de la o stare inițială la o stare finală. Evident, pentru orice stare, suma tuturor probabilităților de tranziție este egală cu unitatea. Matematic, acest lucru se poate reprezenta astfel:

$$\sum_{j=1}^{n} p_{ij} = 1, \forall i \in \overline{1, n} \tag{1}$$

Putem astfel să aplicăm o idee asemănătoare și în problema noastră. Vom asocia fiecărei celule a labirintului câte o stare și vom numerota stările, începând cu colțul din stânga-sus, ca în figura de mai jos:

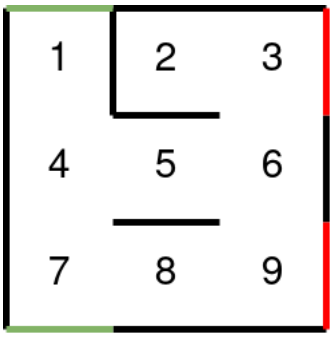

Figura 3: Exemplu de numerotare a unui labirint de dimensiune 3×3

În afară de stările corespunzătoare amplasării robotului într-una dintre celulele labirintului, vom mai avea nevoie de două stări suplimentare:

- **Starea WIN.** Aceasata va fi starea în care putem considera că am câștigat, din care nu mai putem ieși ulterior;
- **Starea LOSE.** Aceasta va reprezenta starea în care putem considera că am pierdut.

Odată adăugate și acestea în graful nostru orienat, discutăm de următorul lanț Markov:

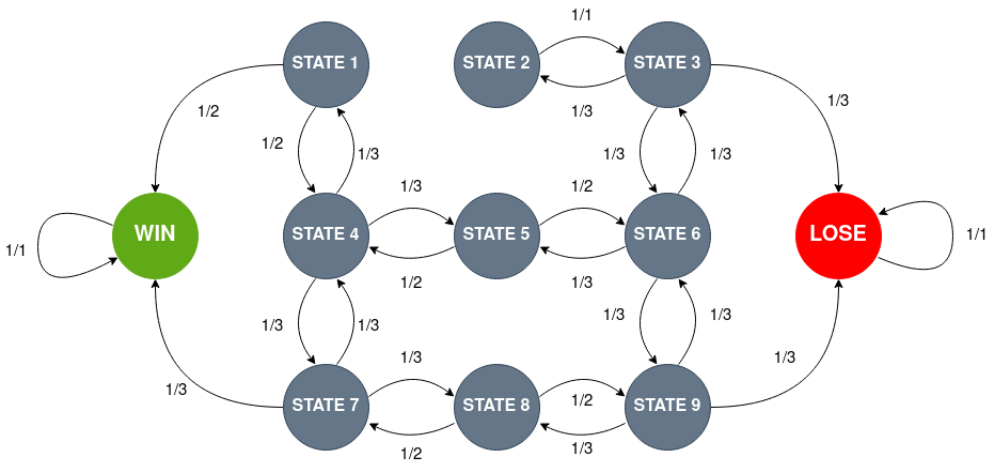

Figura 4: Lanțul Markov asociat exemplului de labirint din fig. 5

Pentru acest lanț Markov (care, din punct de vedere abstract, reprezintă un graf orientat în care fiecare muchie are o anumită greutate, egală cu probabilitatea descrisă mai sus), avem nevoie de o caracterizare concretă (cu alte cuvinte, de un mod de stocare) care să ne ajute în reținerea efectivă a lanțului. În continuare, vom prezenta câteva caracterizări posibile.

1.3.1 Matricea de adiacență

Matricea de adiacență a unui graf orientat, asemănătoare cu conceptul de matrice de adiacență a unui graf neorientat, se poate defini prin următoarea relație:

$$A = (A_{ij})_{i,j\in\overline{1,n}} \in \{0,1\}^{n\times n}, \text{ unde } A_{ij} = \begin{cases} 1, & \text{dacă există o tranziție din starea } i \text{ în starea } j \\ 0, & \text{altfel} \end{cases}$$

În situația grafului din fig. 4, reprezentat prin 11 stări (9 stări corespunzătoare celulelor labirintului, respectiv 2 stări suplimentare, WIN și LOSE, **în această ordine**), matricea de adiacență A a grafului este din $\{0,1\}^{11\times 11}$ și are următoarea formă:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Se observă că sectorul $A(1:9, 1:9)$[2] este simetric, întrucât pereții **nu** sunt unidirecționali (dacă tranziția de la starea i la starea j este posibilă, atunci și tranziția inversă este posibilă).

1.3.2 Matricea legăturilor

Matricea legăturilor reprezintă o formă mai potentă a matricei de adiacență, fiind foarte asemănătoare cu aceasta – singura diferență este dată de semnificația elementelor ce o populează. În cazul matricei de legături, elementele sunt chiar probabilitățile de tranziție de la o stare la alta în lanțul Markov. Folosind notația de probabilitate p_{ij} introdusă anterior (ec. 1), ea se definește astfel:

$$L = (p_{ij})_{i,j\in\overline{1,n}} \in [0,1]^{n\times n} \Leftrightarrow L_{ij} = \begin{cases} p_{ij}, & 0 < p_{ij} \leq 1 \\ 0, & \text{altfel} \end{cases}$$

Pentru exemplul de lanț din fig. 4, matricea legăturilor este următoarea:

$$L = \begin{bmatrix} 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 1/3 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

[2] Am utilizat notația standard din Octave, adică $A(1:9, 1:9)$ se referă la submatricea formată din intersecția primelor 9 linii cu primele 9 coloane.

Observăm că matricea este o matrice stochastică pe linii[3]., ceea ce îi aduce o mulțime de proprietăți interesante, discutate deja în cadrul cursului de Metode Numerice.

1.3.3 Sistem de ecuații liniare

Pe lângă abordările anterioare, putem ține minte lanțul Markov și ca pe o formulare ce se pretează pe un alt stil de problemă: rezolvarea unui sistem de ecuații liniare.

Mai exact, să considera un vector $\boldsymbol{p} \in \mathbb{R}^{mn}$ reprezentând probabilitățile de câștig pentru fiecare celulă din labirint, unde $m \in \mathbb{N}^*$ și $n \in \mathbb{N}^*$ sunt dimensiunile labirintului; spre exemplu, în cazul fig. 5, $\boldsymbol{p} \in \mathbb{R}^9$. Popularea acestui vector se face respectând numerotarea stărilor descrisă anterior.

Să experimentăm puțin cu cazul în care robotelul nostru se află în starea 1. Atunci, el poate efectua următoarele tranziții / deplasări valide prin labirint:

- Poate trece în **starea 4**, cu probabilitatea ½;
- Poate trece în **starea WIN** și să câștige, tot cu probabilitatea ½.

Astfel, starea 1 va fi caracterizată prin următoarea ecuație:

$$\begin{aligned} p_1 &= \frac{1}{2} \cdot p_4 + \frac{1}{2} \cdot p_{\text{WIN}}, \text{ dar } p_{\text{WIN}} = 1 \\ \Rightarrow p_1 &= \frac{1}{2} \cdot p_4 + \frac{1}{2} \end{aligned}$$

În mod similar, se pot scrie ecuații pentru toate stările:

$$\begin{cases} p_1 &= \frac{1}{2} \cdot p_4 + \frac{1}{2} \\ p_2 &= p_3 \\ p_3 &= \frac{1}{3} \cdot p_2 + \frac{1}{3} \cdot p_6 \\ p_4 &= \frac{1}{3} \cdot p_1 + \frac{1}{3} \cdot p_5 + \frac{1}{3} \cdot p_7 \\ p_5 &= \frac{1}{2} \cdot p_4 + \frac{1}{2} \cdot p_6 \\ p_6 &= \frac{1}{3} \cdot p_3 + \frac{1}{3} \cdot p_5 + \frac{1}{3} \cdot p_9 \\ p_7 &= \frac{1}{3} \cdot p_4 + \frac{1}{3} \cdot p_8 + \frac{1}{3} \\ p_8 &= \frac{1}{2} \cdot p_7 + \frac{1}{2} \cdot p_9 \\ p_9 &= \frac{1}{3} \cdot p_6 + \frac{1}{3} \cdot p_8 \end{cases}$$

După cum ne-am obișnuit, putem trece acest sistem în forma sa matriceală:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \end{bmatrix} + \begin{bmatrix} 1/2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1/3 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

[3]Suma elementelor de pe fiecare linie este egală cu 1

În sistemul anterior, vectorul termenilor liberi (evidențiat cu albastru) provine din acele elemente care inițial erau combinații liniare de p_{WIN}.

Am făcut toată această prelucrare în ec. 2 pentru a putea scrie următorul produs:

$$\boldsymbol{p} = G\boldsymbol{p} + \boldsymbol{c}$$

Această formă se pretează perfect metodei Jacobi de rezolvare în care identificăm G și $\boldsymbol{c}$ drept matricea, respectiv vectorul de iterație. De aceea, vom opta pentru această metodă iterativă pentru a soluționa sistemul. În cazul nostru particular (fig. 5), raza spectrală a matricei G are valoarea $\rho(G) \approx 0.85192$, ceea ce înseamnă că Jacobi va converge.

Pentru cei curioși, soluția este:

$$\begin{cases} p_1 \approx 0.84615 \\ p_2 \approx 0.15385 \\ p_3 \approx 0.15385 \\ p_4 \approx 0.69231 \\ p_5 \approx 0.50000 \\ p_6 \approx 0.30769 \\ p_7 \approx 0.73077 \\ p_8 \approx 0.50000 \\ p_9 \approx 0.26923 \end{cases}$$

1.3.4 Algoritm euristic de căutare

Rezultatele de mai sus reflectă o intuiție evidentă: stările care sunt mai *„apropiate”* de **starea WIN** au o probabilitate mai mare de câștig, iar cele care sunt apropiate de **starea LOSE** au o probabilitate mai mică. Acesta este motivul pentru care am putea gândi un algoritm de căutare **euristic** cu ajutorul căruia robotul ar putea ajunge din poziția inițială la una din stările câștigătoare.

Un algoritm de căutare euristic este un algoritm care nu ne furnizează o soluție optimă (în cazul nostru, un drum minim) pentru toate cazurile posibile, însă are avantajul de a fi foarte rapid în comparație cu algoritmii de căutare exhaustivi (clasici).

Apelăm la un algoritm greedy simplu, bazat de DFS, care are pseudocodul de pe următoarea pagină.

În pseudocodul de mai jos, parametrii funcției de căutare sunt următorii:

- ***start_position***, reprezentând poziția de start a robotului în codificarea utilizată până la acest moment (un indice de la 1 la $n \cdot m$ inclusiv, unde n și m sunt dimensiunile labirintului);
- ***probabilities***, prin care se înțelege vectorul probabilităților fiecărei stări în parte, de lungime $n \cdot m + 2$ (acesta este de fapt vectorul extins al probabilităților, spre deosebire de cel calculat anterior folosind metoda iterativă – conține și probabilitățile asociate stărilor WIN, respectiv LOSE);
- ***adjacency_matrix***, matricea de adiacență a labirintului propus, în maniera în care aceasta a fost descrisă mai devreme.

Algoritmul întoarce un vector de indecși reprezentativi celuleleor / stărilor labirintului, urmând ca aceștia să fie traduși ulterior în perechi linie–coloană pentru a putea fi interpretați mai ușor.

Algoritm euristic

```
1:  procedure HEURISTIC_GREEDY(start_position, probabilities, adjacency_matrix)
2:      path ← [start_position]
3:      visited[start_position] ← True
4:      while path is not empty do
5:          position ← top() / last element of the path vector
6:          if position is the WIN state then
7:              return path
8:          if position has no unvisited neighbours then
9:              erase position from the end of the path
10:         neigh ← the unvisited neighbour (with greatest probability to reach WIN) of the current position
11:         visited[neigh] ← True
12:         path ← [path, neigh]
13:     return path (since there is no path to the WIN state)
```

În figura de mai jos, am evidențiat drumul pe care îl va alege robotul, luând în considerare probabilitățile calculate anterior (vom porni în acest exemplu din starea / celula 2).

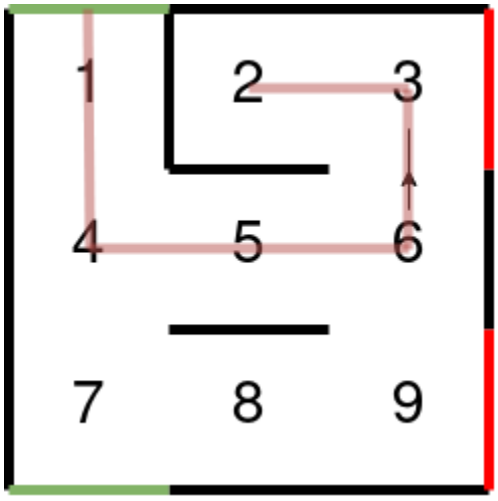

Figura 5: Exemplu de drum obținut folosind algoritmul euristic

1.3.5 Codificarea labirintului

Pentru a prelucra labirintul sub forma unor date de intrare, este necesară o reprezentare a labirintului într-o formă condensată. Astfel, Mihai se inspiră dintr-un algoritm pe care l-a găsit într-un alt context, cel al graficii pe calculator, numit algoritmul Cohen-Sutherland.

Ideea preluată din algoritmul original este de a codifica binar zidurile ce separă celule adiacente spațial: labirintul nostru poate fi stocat drept o matrice cu $m \times n$ intrări, numere întregi reprezentate pe 4 biți de forma $\overline{b_3 b_2 b_1 b_0}_{(2)}$, unde fiecare bit activ (setat pe 1) reprezintă o posibilă direcție de deplasare obturată de un perete al labirintului. În cazul nostru, ne însușim următoarea codificare:

- Bitul b_3 setat pe 1 indică un **zid la nordul celulei**;
- Bitul b_2 setat pe 1 indică un **zid la sudul celulei**;

- Bitul b_1 setat pe 1 indică un **zid la estul celulei**;
- Bitul b_0 setat pe 1 indică un **zid la vestul celulei**;

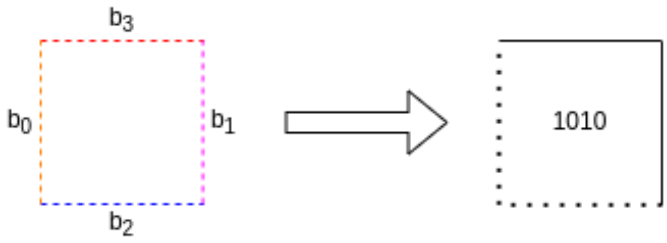

Figura 6: Reprezentarea direcțiilor în algoritmul Cohen-Sutherland, alături de un exemplu

În situația labirintul nostru, codificarea ar fi:

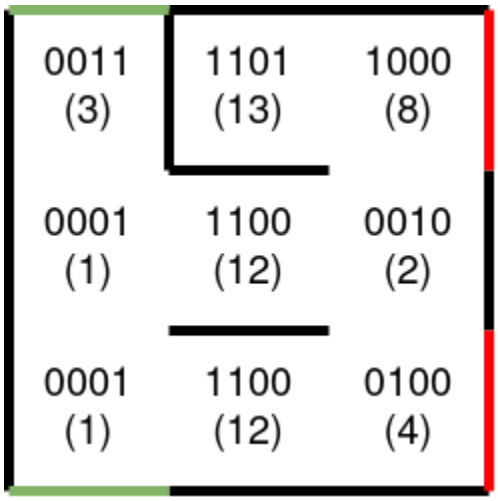

Figura 7: Codificarea pereților labirintului pentru exemplul dat

Este foarte important să observați faptul că pereții sunt bidirecționali (anume că, deși o codificare aleatoare ar permite tranziții unidirecționale între stări, noi vom trata exclusiv cazul pereților care blochează tranzițiile în labirint în ambele sensuri între oricare stări adiacente).

1.4 Cerințe

În urma parcurgerii materialului teoretic furnizat anterior, sunteți pregătiți să implementați următoarele funcții în Matlab:

- `function [Labyrinth] = parse_labyrinth(file_path)`

  Funcția `parse_labyrinth` va primi o cale relativă către un fișier text unde se află reprezentarea codificată a labirintului, așa cum a fost descrisă în secțiunea de teorie dedicată.

  Formatul fișierului de intrare va fi următorul:

```
m n
l_11 l_12 l_13 ... l_1n
l_21 l_22 l_23 ... l_2n
l_31 l_32 l_33 ... l_3n
...
l_m1 l_m2 l_m3 ... l_mn
```

- `function [Adj] = get_adjacency_matrix(Labyrinth)`

  Funcția `get_adjacency_matrix` va primi matricea codificărilor rezultată după pasul anterior și va întoarce matricea de adiacență a grafului / lanțului Markov.

- `function [Link] = get_link_matrix(Labyrinth)`

  Funcția `get_link_matrix` va primi matricea codificărilor unui labirint valid și va returna matricea legăturilor asociată labirintului dat.

- `function [G, c] = get_Jacobi_parameters(Link)`

  Funcția `get_Jacobi_parameters` va primi matricea legăturilor obținută anterior și va returna matricea de iterație și vectorul de iterație pentru metoda Jacobi.

- `function [x, err, steps] = perform_iterative(G, c, x0, tol, max_steps)`

  Funcția `perform_iterative` va primi matricea și vectorul de iterație, o aproximație inițială pentru soluția sistemului, o toleranță (eroare relativă maxim acceptabilă pentru soluția aproximativă a sistemului, între doi pași consecutivi) și un număr maxim de pași pentru execuția algoritmului.

- `function [path] = heuristic_greedy(start_position, probabilities, Adj)`

  Funcția `heuristic_greedy` va primi o poziție de start (un index al unei celule / stări din intervalul $\overline{1, mn}$), vectorul extins al probabilităților (incluzând cele două probabilități pentru stările WIN și LOSE) și matricea de adiacență a lanțului Markov.

  Va returna apoi o cale validă către starea de WIN. Se garantează că labirintul (și, implicit, graful asociat) este conex, și deci va exista întotdeauna o cale de câștig validă.

- `function [decoded_path] = decode_path(path, lines, cols)`

  Funcția `decode_path` va primi o cale validă (sub forma unui vector coloană) și dimesniunile labirintului și va returna un vector de perechi (matrice cu două coloane), fiecare pereche reprezentând linia și coloana celulei cu codificarea dată.

1.4.1 Restricții și precizări

Înainte să vă apucați de lucru, ar fi bine să luați aminte că:

- Labirintul **NU** este neapărăt pătratic (numărul de coloane nu trebuie să coincidă cu numărul de linii);
- Se garantează faptul că labirintul este conex și că există mereu câte o cale către ieșirea / starea de WIN și către cea de LOSE;

- Observați că matricele de adiacență și de legătură sunt matrice mari, dar rare. Este **OBLIGATORIE** reținerea acestor matrice și a matricelor derivate (precum matricea de iterație sau vectorul de iterație) sub forma unor **matrice rare**. Octave vă oferă posibilitatea stocării matricelor rare într-o manieră mult mai eficientă decât cea convențională, anume prin stocarea elementelor nenule și a pozițiilor acestora. De asemenea, există funcții specializate pentru lucrul cu matrice rare, pe care vă încurajăm să le descoperiți aici.

Pentru restricțiile general valabile, verificați sfârșitul acestui document.