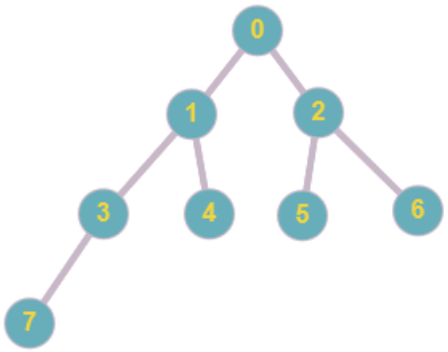
Dezvoltarea ideilor proprii

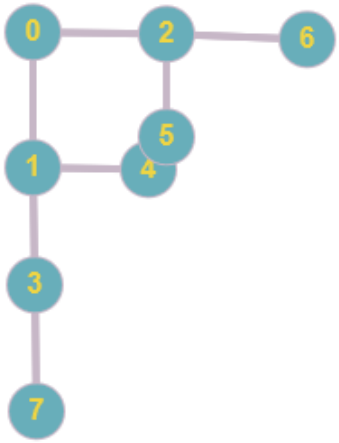
Este o metoda modificata a algoritmului de generare Binary Tree din primul capitol. Simplitatea algoritmului original este foarte atragatoare, mai ales pentru ca nu necesita o structura de date suplimentara pentru stocare. Generarea aleatoare este buna in marea majoritate a cazurilor, mai putin cand cautam consistenta. Daca dorim sa stocam labirintul generat putem sa salvam matricea cu totul, sa salvam o lista de coordonate ale zidurilor sau chiar ca arbore binar, dar toate cazurile rezulta in fisiere maricele in raport cu numarul de noduri.

Eu am venit cu o alta solutie la aceasta dilema a stocarii si reprezentarii, si totodata cu asta o noua metoda de parsare, generare si calcul al valorilor arborelui.

De ce nu orice arbore binar poate fii un labirint?

Datorita metodei in care un arbore binar este construit in general, sunt sanse foarte mari ca doua noduri (doi copii) sa se certe pentru acelasi loc din matrice. Luam exemplul urmator:





In prima figura vedem arborele dupa cum suntem obisnuiti sa il vedem. In a doua figura arborele a fost pozitionat asemanator unei matrici astfel ca radacina lui se afla in coltul din stanga sus al acesteia, copilul din stanga se afla sub el iar cel din dreapta in dreapta lui. Aceasta repozitionare se face pentru fiecare nod din arbore.

Putem observa ca nodurile 4 si 5 se afla in aceeasi locatie, lucru ce nu este de dorit deoarece astfel apar incaperi inaccesibile in labirint. Prin urmare unul dintre cele doua noduri trebuie sa dispara.

Numerotarea nodurilor

Conventia de numerotare a nodurilor dintr-un arbore binar in literatura se face astfel: nodul din stanga este 2 * n iar cel din dreapta 2 * n + 1, unde n este valoarea parintelui. Dar astfel ajungem la valori mari si pentru arbori cu 10 noduri, cea ce nu este de dorit. Luam un arbore pentru un labirint complet:

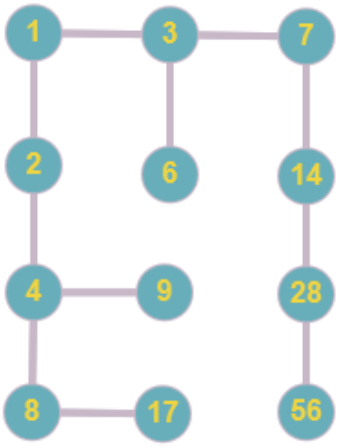


Figura 1

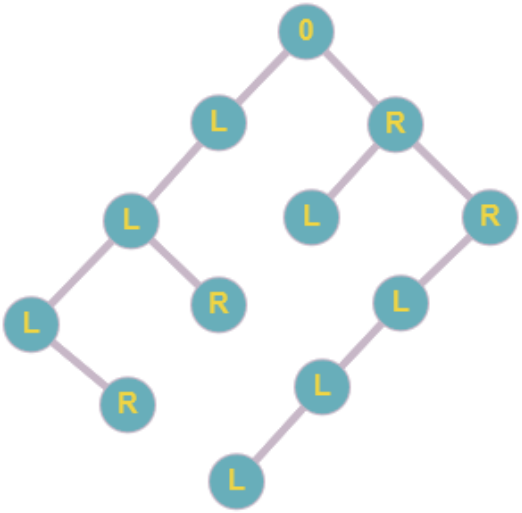


Figura 2

In prima figura arborele este notat dupa conventia precizata mai sus, iar in cea de-a doua figura am folosit o notatie mai simplista: fiecare nod (cu exceptia radacinii) este notat cu L (left) sau cu R (right) in functie de pozitia in care se afla fata de parinte.

Se observa de asemenea ca folosirea numerelor duce la valori mari dupa cum am precizat. Nu este necesara atata informatie in fiecare nod asa ca am optat pentru aceasta metoda simpla de notare, care o sa ne ajute mult si la stocarea arborelui.

Stocarea / reprezentarea arborelui

Exista multe metode / feluri de stocare a arborilor binari precum:

1. In fisiere .xml datorita naturii parent-child al formatarii si parsarii fisierului
2. Scrierea propriu-zisa in fisier a valorilor nodurilor si parsarea pe nivel a acestora (nivelul curent este de lungime x, urmatorul nivel este de lungime 2 * x) dar acest lucru duce la completarea locurilor goale a arborelui cu noduri “dummy” (pentru ca nu intotdeauna este un arbore complet, si in cazul nostru sigur nu este complet)

ex: `10 5 30 4 8 -1 40 1 -1 -1 -1 -1 -1 -1 -1`

3. Scrierea in fisier doar a muchiilor asemanator unui graph

ex: (1, 2), (2, 4), (2, 5), (1, 3), (3, 7)

Dar noi putem sa ne bazam pe cateva particularitati ale arborelui nostru.

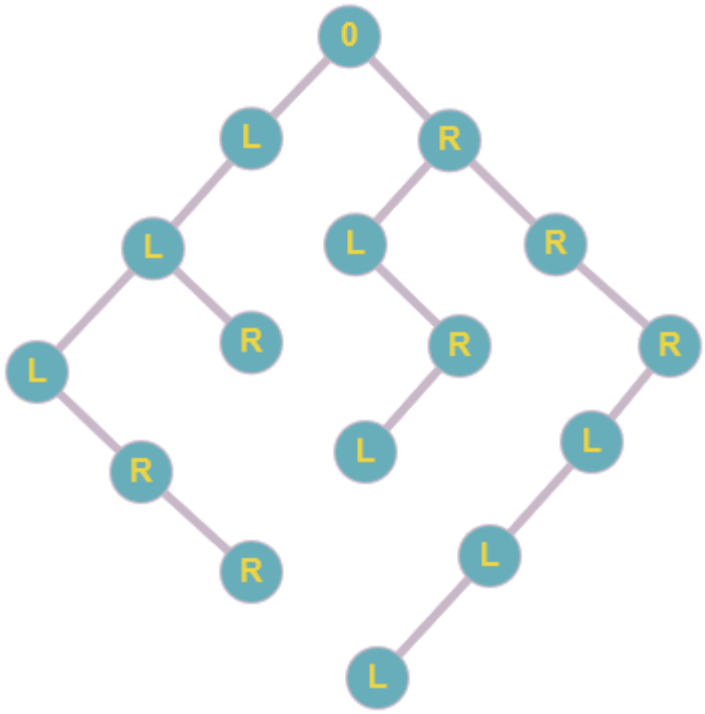


Figura 3

Asa arata un labirint “complet” reprezentat ca un arbore binar. Putem observa ca mereu nivelul urmator este cu un nod mai mare fata de cel precedent, pana la jumatatea arborelui. Dupa jumatate, nivelul urmator este cu un element mai mic fata de cel anterior.

Acest lucru ne este de folos cand dorim sa reprezentam in fisier arborele (reprezentarea pe nivel de la stanga la dreapta a figurii):

0 LR LLR LRRR RLL RL L

Daca arborele este patratic, precum in figura 3, delimitarile nu sunt necesare. Un simplu string pentru stocarea intregului arbore este suficient (explicatie in subcapitolul de parsare).

ex: 0LRLLRLRRRRLLRLL

Daca arborele nu este patratic (figura 2) atunci pentru stocare o sa mai fie nevoie de doi parametrii care ne indica lungimea respectiv latimea asemanator unei matrici sau putem sa despartim nivelele cu ajutorul unui delimitator.

ex: 1) “0 LR LLR LRL RL L”

2) “0, LR, LLR, LRL, RL, L”

3) 4, 3 // lungimea, latimea

0LRLLRLRLRLL

Astfel ca reprezentarea simplista nu este extrem de avantajoaza in cazul acesta, comparativ cu arborele patratic.

Parsare

In cazul arborelui patratic (figura 3) in fisierul de intrare o sa fie doar un string, ce reprezinta intreg arborele. Acum intervine problema parsarii: cum delimitam nivelele pentru a putea construi arborele? Evident prima linie va avea lungimea unu, a doua linie lungimea doi si tot asa pana la o anumita valoare, la care numarul de noduri de pe nivel trebuie sa scada treptat pana la valoarea unu.

Datorita formei patratice a arborelui nostru valoarea pe care o cautam pentru delimitare este chiar radacina patrata a lungimii string-ului.

$$value = \sqrt{len(string)}$$

Astfel ca lungimea nivelelor o sa inceapa cu valoarea unu, se incrementeaza treptat pana la aceasta valoare, dupa care lungimea o sa scada cu fiecare nivel cu cate unu.

Exemplu de parsare in python pentru arborele patratic din figura 3:

```python
from math import sqrt


def parse(string):
    tree_levels = []  # unde stocam nivelele
    start_pos = 0  # pozitia de start din string
    length = 1  # lungimea nivelului
    step = 1
    while length > 0:
        current_level = ""
        for pos in range(start_pos, start_pos + length):
            current_level += string[pos]

        tree_levels.append(current_level)
        if length == sqrt(len(string)):  # verificam daca trebuie sa incepem decrementarea
            step = -1
        start_pos += length  # calculam noul punct de start
        length += step  # modificam lungimea niveluli
    return tree_levels


print(parse("0LRLLRLRRRRLLRLL"))
```

Output: ['0', 'LR', 'LLR', 'LRRR', 'RLL', 'RL', 'L']

Avand aceasta lista putem sa construim matricea/arborele pentru o reprezentare mai adecvata a labirintului.

Generare

Generarea este foarte simpla datorita reprezentarii simpliste a labirintului. Regulile ce trebuie urmate sunt:

N = marimea labirintului (NxN)

1. Radacina este 0 (conventie)
2. Daca lungimea nivelului curent este mica sau egala cu N: nivelul trebuie sa inceapa cu “R” si sa se termine cu “L”
3. Contrar: nivelul trebuie sa inceapa cu “L” si sa se termine cu “R” (exceptia este nodul final)
4. Nodul final poate sa fie “L” sau “R”

Putem conveni ca nodul final sa fie si “1” daca dorim ca labirintul sa nu aibe iesire (cazul trebuie tratat separat), conventia devenind astfel:

0 – radacina

L/R - nod intermediar

1 – nod final

Cum noi dorim sa avem o iesire din labirint nu o sa folosim aceasta conventie.

Spre deosebire de generarea originala din algoritmul Binary Tree de la care a pornit aceasta metoda, nu e nevoie sa ne deplasam pe coordonate sau pe elementele unei matrici ci generam direct un string ce ulterior poate fi reprezentat ca un labirint.

Daca dorim sa implementam generarea intr-un joc 2D e mai usor sa ne ghidam dupa o “cheie” (si anume string-ul nostru generat) cand vrem sa mai accesam anumite noduri, mai ales cand vrem sa persistam labirintul.

Mai mult, aceasta cheie este practic “seed"-ul labirintului generat.

Exemplu de generare in python a unui labirint patratic:

```python
from random import randint


def generate(n):
    string = "0"  # string-ul ce urmeaza sa il generam
    length = 2  # lungimea e 2 deoarece am generat deja pentru lungime 1
    begin = 'R'  # caracterul cu care incepe nivelul
    end = 'L'  # caracterul cu care se incheie nivelul
    step = 1
    while length > 1:
        in_between = ""
        for pos in range(length - 2):
            in_between += 'L' if randint(0, 1) else 'R'  # generam ce este in nivel

        string += begin + in_between + end  # concatenam nivelul la string
        if length == n:
            step = -1
            begin, end = end, begin  # daca am trecut de jumatate incepem nivelul cu L si terminam cu R
        length += step
    string += 'L' if randint(0, 1) else 'R'  # tratam nodul final separat
    return string


print(generate(3))
```

Output: 0RLRRLLRR (din cele patru cazuri posibile)

Tot ce ramane de facut acum este sa integram seed-ul nostru generata in program. Un exemplu bun este punerea in matrice a elementelor din seed pe cazuri. Daca elementul este “L”, atunci facem legatura cu elementul de deasupra celui curent, iar daca elementul este “R” atunci il legam de cel din drepta nodului actual, metoda fiind identica cu cea din algoritmul original.

Avand acces la toate aceste informatii putem stii cu exactitate si locatiile care indiferent de seed vor fi mereu goale. Acest lucru este foarte bun in caz ca vrem sa populam anumite zone din labirint cu diverse lucruri. Nu mai este nevoie sa interogam celula din matrice cu legatura la valabilitatea ei, si vom merge direct pe celulele goale.

Formule

Inevitabil o sa ne lovim si de necesitatea accesarii valorilor din seed in functie de coordonatele celulelor din matrice.

	1	2	3	4	5	6
1	0	2	5	9	14	20
2	1	4	8	13	19	25
3	3	7	12	18	24	29
4	6	11	17	23	28	32
5	10	16	22	27	31	34
6	15	21	26	30	33	35

In tabelul de mai sus se afla o matrice de 6x6, indexata de la 1 atat pe lungime cat si pe latime. Chenarele cu rosu sunt menite sa scoata in evidenta diagonala secundara. Valorile din matrice semnifica pozitia elementului din seed-ul labirintului.

Astfel ca daca transpunem arborele cu radacina in pozitia (1, 1), copilul din stanga lui o sa fie pe pozitia (2, 1) iar cel din dreapta lui in (1, 2) si tot asa cu fiecare element si copiii lui aferenti.

Deci daca de exemplu avem la coordonatele (4, 2) valoarea 11, acest lucru inseamna ca acolo este elementul de pe pozitia 11 din string-ul generat pentru labirint (indexare de la 0).

Linia rosie mai reprezinta astfel si nivelul de lungime maxima din arbore sau nivelul de mijloc astfel ca arborele parca este “culcat” pe o parte si numerotat de la radacina.

Stiind coordonatele din matrice putem calcula valorile ce se afla pe pozitia respectiva si prin urmare pozitia elementului care trebuie sa se afle in acea celula.

Am dedus urmatoare formula:

$$value = \sum_{x=1}^{i-1} x + \sum_{y=i+1}^{i+k-1} y + \sum_{z=1}^{j-k} (n-z+1)$$

Unde:

i, j – coordonatele la care ne aflam in matrice

k – min(j, n - i + 1)

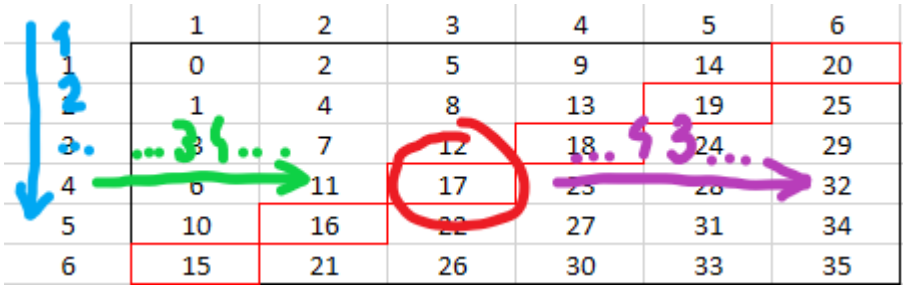
n – dimensiunea matrice

Desi arata destul de complex si neprimitor, fiecare suma e destul de simplu de inteles. Am sa explic detaliat ce face fiecare:

1. Prima suma (suma x): calculeaza elementul de la care plecam si anume elementul de pe prima coloana la indicele ‘i’.
2. A doua suma (suma y): calculeaza valoarea ce trebuie adunata pana la elementul de pe diagonala secundara (daca perechea noastra (i, j) se afla sub aceasta). Dupa cum se observa, daca plecam cu o valoare de pe prima coloana si ne mutam in dreapta cu cate o casuta, numerele cresc cu valori consecutive, pana la elementul de pe diagonala secundara. Astfel apare acel ‘k’, menit sa faca suma sa se deplaseze pana la valoarea de pe diagonala, daca cumva ‘j’-ul o depaseste.
3. A treia suma (suma z): calculeaza valoarea ce trebuie adunata pana la ‘j’-ul cautat, daca acesta depaseste diagonala secundara. Se observa ca odata cu diagonala secundara, numerele incep sa fie incrementate cu valori descrescatoare consecutive. Datorita relatiei lui ‘k’ si ‘j’, daca elementul cautat se afla deasupra diagonalei, suma va da 0.

Pe scurt: prima suma calculeaza elementul de start, al doile imi calculeaza cresterea consecutiva a valorilor iar cea de-a treia descresterea consecutiva a valorilor (daca exista).

O exemplificare mai vizuala:



	1	2	3	4	5	6
1	0	2	5	9	14	20
2	1	4	8	13	19	25
3	3	7	12	18	24	29
4	6	11	17	23	28	32
5	10	16	22	27	31	34
6	15	21	26	30	33	35

Exemplu pentru i = 4 si j = 6 (rezulatatul dorit este 32, precum e in table la (4, 6)):

Prima suma: 1 + 2 + 3 = 6 // pe pozitia (4, 1) se afla valoarea 6, pozitia noastra de inceput

A doua suma: k = min(6, 6 - 4 + 1) = 3 // pe pozitia i = 4 si k = 3 se afla elementul de pe diagonala

5 + 6 = 11, pe care il adunam la prima suma si da 17 // elementul de pe diagonala

A treia suma: 6 + 5 + 4 = 15, pe care il adunam la tot ce am calculat precedent si da 32, elementul pe care il cautam

Sumele pot fi transformate in formule si arata astfel:

$$value = \frac{i(i-1) + (k-1)(2i+k) + (j-k)(2n-j+k+1)}{2}$$

Nu are rost sa desfacem parantezele pentru ca nu se simplifica multe. Nu arata nici in forma asta prea atractiv de aceea prefer formula cu sume, care e si mai usor de vizualizat odata ce intelegi ce se intampla in spate.

Avantaje

1. Salvarea intregului arbore/labirint ca un singur string este convenabil
2. String-ul actioneaza si ca un “seed”
3. Generare foarte simpla si usor de implementat
4. Parsarea nu e complicata
5. Gasirea elementelor se face rapid cu ajutorul formulelor

Dezavantaje

1. Suntem limitati de lungimea maxima acceptata de un string. De exemplu: in C++ lungimea maxima a unui string este 4294967291, lucru ce ne permite sa avem labirinte de marime maxima 65535
2. Gasirea elementelor, desi mai rapida, nu este asa intuitiva