Mesterséges intelligencia alapjai (BMEGEGTBX01)

Egyéni feladat

LABIRINTUSOK GENERÁLÁSA ÉS MEGOLDÁSA ÚTKERESŐ ALGORITMUSOKKAL

Gajdos Marcell András - QVUBJU

2023. ősz

Tartalom

Feladat

Készítsen programot, ami képes 2 dimenziós labirintusgeneráló- és megoldó algoritmusokat összehasonlítani!

Specifikáció

Alapvetések

A program python programnyelven íródott, az alábbi könyvtárakat felhasználva

- Tkinter: GUI és grafikus megjelenítés
    - Ttk: checkboxok létrehozása miatt
    - Filedialog: tallózás miatt
- Random: véletlengeneráláshoz
- Time: késleltetéshez
- Datetime: csv fájl elnevezéséhez
- Os: csv exportáláskor a könyvtár helyének megállapításához

A generált labirintusok négyzet alakúak és négyzet alakú cellákból épülnek fel. A start cella mindig a négyzet bal felső cellája, a cél pedig mindig a labirintus közepén, vagy a középpontot érintve helyezkedik el.

A cellák falai 2 dimenziós mátrixként kerülnek eltárolásra, egy cellában egy 4 elemű tömbben tárolódik, hogy melyik irányba lehet továbbmenni a cellából. A tárolás sorrendje az alábbi:

- X koordináta pozitív irány
- X koordináta negatív irány
- Y koordináta pozitív irány (fel)
- Y koordináta negatív irány (le)

Ennek értelmében a határok mint falak, vannak reprezentálva, nem mint teli cellák. Az y koordináta irányai a megjelenítésben a hagyományos Descartes-féle koordináta rendszerhez képest fel vannak cserélve, így az érték a képernyőn lefelé haladva növekszik.

A program egy két dimenziós mátrixban tárolja a cellákat, mint megjelenített objektumokat, ezzel elkerülve azt, hogy mindig új négyzeteket kelljen “egymásra” rajzolni. Ez azon túl, hogy nagyban gyorsítja a futtatást, a futtatások számának növekedésével bekövetkező lassulást is kiküszöböli.

Algoritmusok

A választott labirintus generáló algoritmusok:

- Véletlenszerű, mélységben először (random DF)
- Prim-féle (Prim’s)
- Bináris fa (Binary tree)

Az Bináris fa generáló algoritmuson kívül beiktattam egy, a labirintus méretétől függő random cellaösszenyitást (loop modifier) is, hogy több lehetséges megoldás is

létrejöhessen. Ez gyakorlatban egy 1/(oldalhossz * loopModifier) valószínűséggel nyitja össze a cellákat. Értéke 1 és 100 között állítható.

A választott útkereső algoritmusok:

- Mélységben először (DF)
- Szélességben először (BF)
- Mélységben először iteratív mélyítéssel (DFID)
- A*
- IDA*

Tulajdonságok

A program generáló módban képes a felhasználó által bevitt paraméterek (méret, választott algoritmus, loop modifier) alapján labirintusokat generálni.

Megoldó módban a program a kiválasztott algoritmus segítségével képes megoldani az akkor generált labirintust és naplózni a megoldás közben bejárt cellák számát, a talált út hosszát, ezek arányát.

Mind generáló, mind megoldó módban képes a program valós időben mutatni a művelet pillanatnyi helyzetét.

Lehetőség van továbbá a labirintusok mentésére, mentett labirintusok tallózására és megoldására is.

Kiemelten fontos szabály, hogy megoldáskor az algoritmus nem az eltárolt mátrixot beolvasva tájékozódik, hanem a labirintust celláról cellára fel kell fedeznie. Ez a megkötés a feladathoz ötletet adó Micromouse verseny jellegéből adódik, itt az egér nem látja egyben az egész labirintust, csupán már az általa bejárt cellákat, ezek alapján végzi a döntéshozást.

Felhasználói felület és vizualizáció

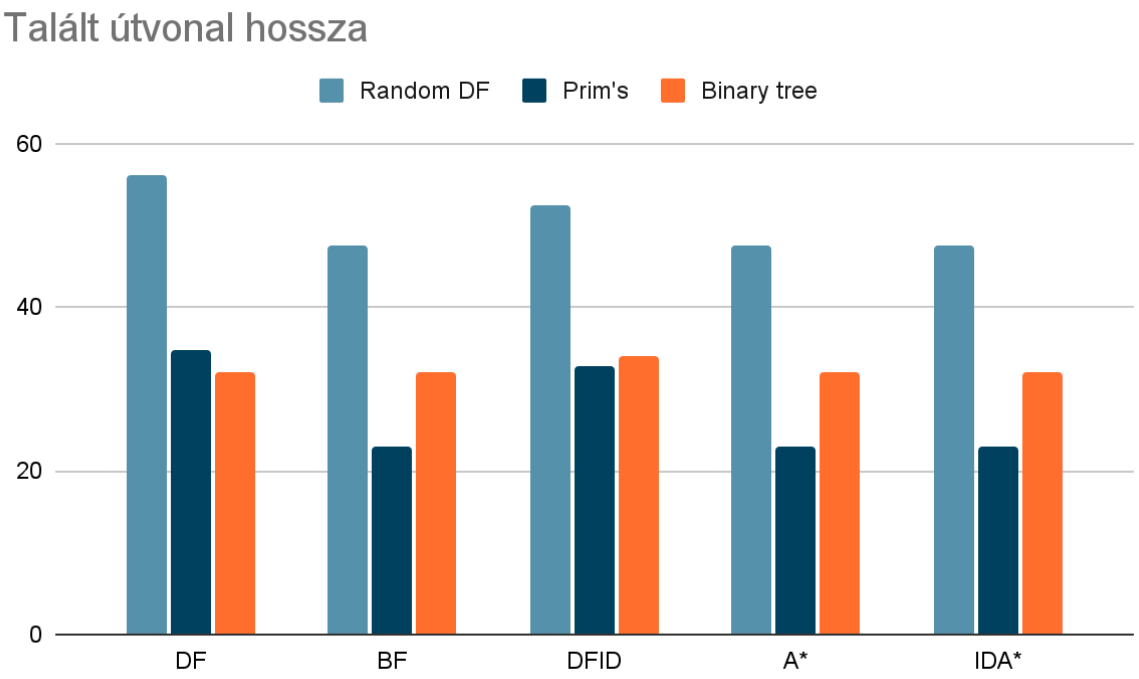
A program teljes mértékben grafikus felhasználói felületről kezelhető. A labirintus egy oldalának cellaszáma megadható, generáló és megoldó algoritmusok választhatók és a valósidejű futást is ki-be lehet kapcsolni, akár generálás vagy megoldás közben. Ezen kívül algoritmusonként a bejárt cellák és a visszaadott útvonal hosszának átlagát is naplózza a rendszer, ezt az erre rendszeresített gombbal lehet törölni.

A cellaszínek jelentése az alábbi:

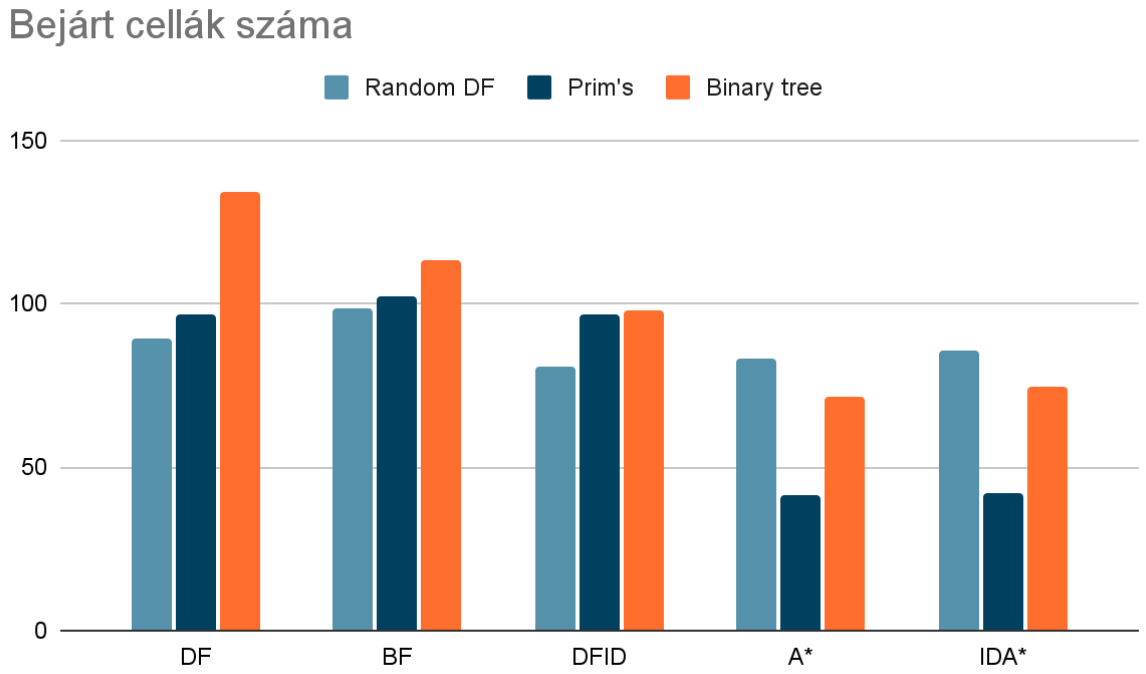
- Világosszürke cella: Generáláskor a még le nem generált cellák
- Sötétszürke cella fekete falakkal:
    - generáláskor: legenerált cellák
    - megoldáskor: be nem járt cellák
- Zöld cella: start
- Piros cella: cél
- Kék cella: bejárt cellák
- Narancssárga cella: algoritmus által visszaadott útvonal

Algoritmusok összehasonlítása

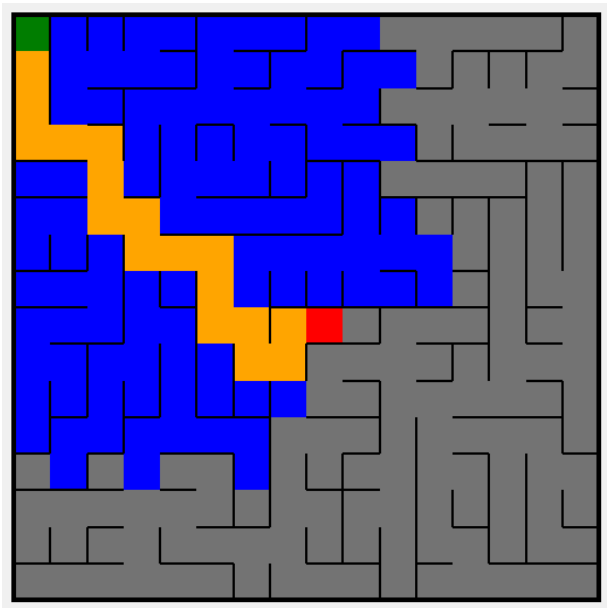
Az útkereső algoritmusokat mindhárom generáló algoritmus által generált, 16 cella oldalhosszú labirintuson teszteltem, a loop modifier értékét 5-nek választva. Az értékek 4 generált labirintus átlagai.

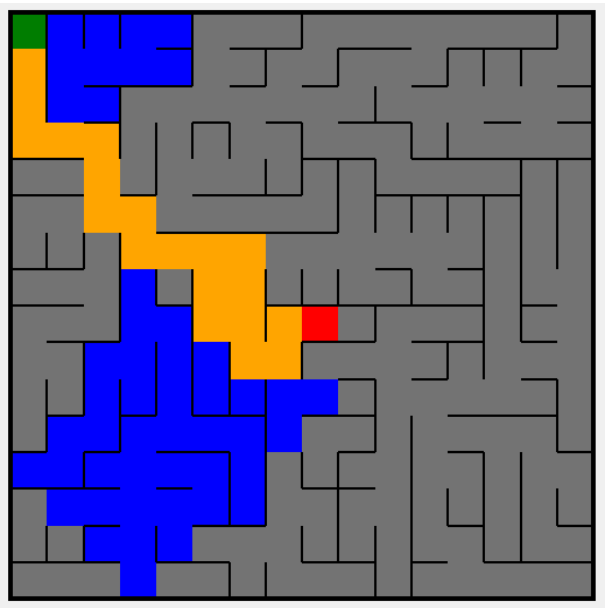


Az adatok beigazolják az úkereső algoritmusok elméleti karaktersztikáit. A BF, A* és IDA* algoritmusok mivel optimálisak, mindig a legrövidebb utat találják meg, így az utak átlaga megegyezik és a legalacsonyabb. A DF és IDDF nem optimális, ezek hosszabb utakat találtak.
Érdemes még megjegyezni, hogy a random DF által generált algoritmusokban lényegesen hosszabb az útvonal, mint a Prim’s esetében. A bináris fa algoritmus optimális útvonalainak szórása igen magas, így az úthosszak átlaga a másik kettő közé esik.



A bejárt cellák száma sem okozott különösebb meglepetést, az algoritmusok papírforma alapján teljesítettek. A DF és a BF egyaránt sok cellát járt be, a DFID némileg kedvezőbb volt, míg az A* és az IDA* a heurisztikának köszönhetően lényegesen kevesebb cellát járt be.
Ezen felül futtatás közben szembetűnő volt, hogy az iteratív mélyítést használó algoritmusok lényegesen lassabban dongoztak mint amelyek nem mélyítettek iteratívan, az eredményeik viszont nem kárpótolják a futásidejüket.
Minden tényezőt figyelembe véve a vakon keresők közül a szélességben először, míg a heurisztikusok közül az A* használata a legcélszerűbb a feladat ellátására. Mivel az útvonaltervező algoritmusok többsége az A* keresést használja, így ez az eredmény is igazolja a modell helyességét. Az IDA* többnyire sokkal hosszabb idő alatt adott vissza az A*-gal megegyező megoldást, így ez jelen problémára nem hatékony megoldás.
A DFID algoritmus esetében előfordul még, hogy az iteratív mélyítés csupán csekély mértékben javítja a hatékonyságot. Ennek oka, hogy a már bejárt cellákba nem lép ismét, így hurkokat tartalmazó labirintusok esetében előfordulhat, hogy egy jobb útvonalat később térképezne fel és egy rosszabb már blokkol egy cellát, amit ki kellene terjesztenie. A jelenséget alább szemléltetem, az első képen az optimális útvonal BF algoritmussal, a másodikon a DFID által talált nem optimális.





Létrehozott függvények:

- genD1st: Véletlenszerű mélységben előszöri generálás
- genPrim: Prim féle generálás
- genBinTree: Bináris fa generálás
- solveDF: Mélységben először keresés
- solveBF: Szélességben először keresés
- solveDFID: Mélységben először keresés iteratív mélyítéssel
- solveAstar: A* keresés
- solveIDAstar: IDA* keresés
- getPossible: Cella körüli nem generált cellákat adja vissza
- getAll: Cella körüli összes cellát adja vissza
- updateWalls: Generálás közben a falakat tartalmazó mátrixot frissíti
- getPassage: Megoldáskor lehetséges kiterjeszthető cellákat adja vissza
- initCells: a cellák vizualizációját tartalmazó mátrixokat inicializálja
- modifyCell: Cella vizualizációt változtat
- generate: generáló gomb által hívott függvény, generáló algoritmust választ
- solve: megoldó gomb által hívott függvény, generáló algoritmust választ
- clear: letörli a megoldást
- exportcsv: csv-be exportálja a legenerált labirintust
- Importcsv: csv-ből importál mentett labirintust
- updateStats: frissíti a tárolt statisztikát
- clearStats: törli a tárolt statisztikát