FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
SLOVENSKÁ TECHNICKÁ UNIVERZITA
Ilkovičova 2, 842 16 Bratislava 4

2022/2023

Peter Bartoš
Umelá inteligencia
Zadanie A

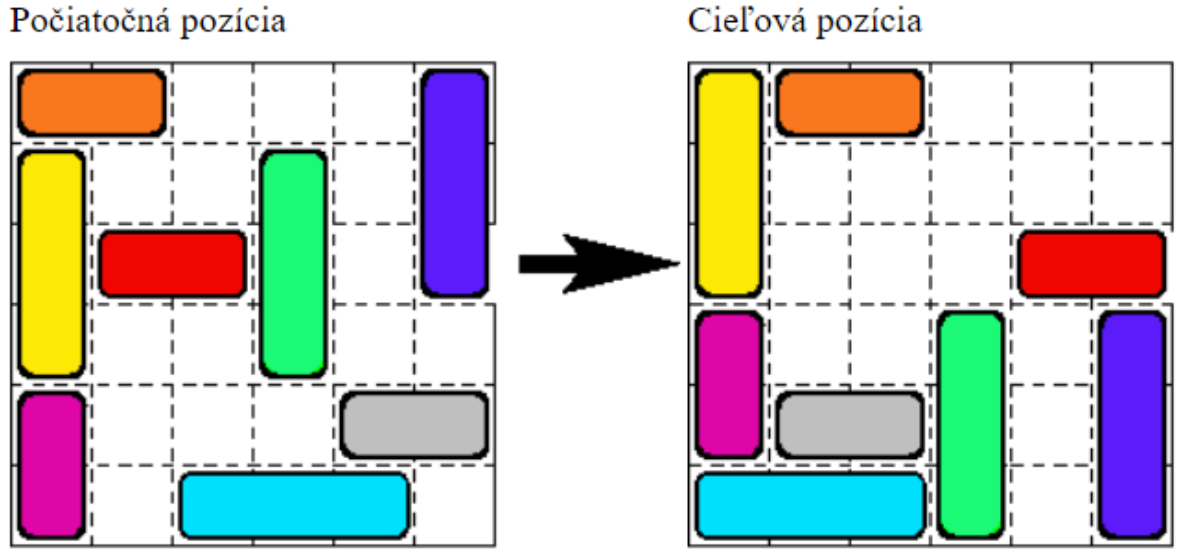
Programovacie prostredie: PyCharm (Python)
Čas cvičení: Streda 13:00 – 14:50

Vypracoval: Peter Bartoš
AIS ID: 116143

Obsah

Riešený problém

Úlohou je nájsť riešenie hlavolamu **Bláznivá križovatka**. Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6 krát 6 políčok a obsahuje niekoľko vozidiel (áut a nákladiakov) rozložených na mriežke tak, aby sa neprekrývali. Všetky vozidlá majú šírku 1 políčko, autá sú dlhé 2 a nákladiaky sú dlhé 3 políčka. Autíčka sa môžu posúvať iba v smere v ktorom sú otočené a otáčať sa nevedia. Riešenie hlavolamu predstavuje aby sa ľubovoľné auto dostalo na predom určené miesto.



Riešenie problému v zadaní A je dosiahnuté pomocou prehľadávania do hĺbky alebo prehľadávania do šírky.

Stavy

Stavy sa v riešení reprezentujú ako pole, kde v každom tomto poli je uložené pole s autom, ktoré obsahuje jeho farbu, veľkosť, súradnicu x, súradnicu y a smer:

```
uzol = [
  ["cervene", 2, 3, 2, 'h'], ["oranzove", 2, 1, 1, 'h'],
  ["zlte", 3, 2, 1, 'v'], ["fialove", 2, 5, 1, 'v'],
  ["zelene", 3, 2, 4, 'v'], ["svetlomodre", 3, 6, 3, 'h'],
  ["sive", 2, 5, 5, 'h'], ["tmavomodre", 3, 1, 6, 'v']
      ]
```

Cieľ stačí reprezentovať iba autom v cieľovej destinácií:

```
vystup = ["cervene", 2, 3, 5, 'h']
```

Operácie

Pohyb vpravo, vľavo, hore a dole je riešený pomocou matíc, kde najprv sa vyhotoví matica s prázdnymi políčkami:

```
# Vytvorenie pola
pole = [[-9, -9, -9, -9, -9, -9],
        [-9, -9, -9, -9, -9, -9],
        [-9, -9, -9, -9, -9, -9],
        [-9, -9, -9, -9, -9, -9],
        [-9, -9, -9, -9, -9, -9],
        [-9, -9, -9, -9, -9, -9]]
```

Matica sa ďalej vyplní autíčkami pomocou cyklu:

```
i = 0
while i < len(uzol):  # Vytvaranie pola s autami
    auticko = uzol[i]   # Zobratie i-teho auta z uzlu
    velkost_auticka = auticko[1]  # velkost auticka
    suradnice_x = auticko[2] - 1  # x-ove suradnice auta
    suradnice_y = auticko[3] - 1  # y-nove suradnice auta
    if pole[suradnice_x][suradnice_y] == -9 and auticko[4] == 'v':  #
vypisanie vertikalneho auticka do pola
        pole[suradnice_x][suradnice_y] = i
        pole[suradnice_x + 1][suradnice_y] = i
        if velkost_auticka == 3:
            pole[suradnice_x + 2][suradnice_y] = i 3, tak treba
vypisat este jedno policko
    elif pole[suradnice_x][suradnice_y] == -9 and auticko[4] == 'h':
# vypisanie horizontalneho auticka do pola
        pole[suradnice_x][suradnice_y] = i
        pole[suradnice_x][suradnice_y + 1] = i
        if velkost_auticka == 3:
            pole[suradnice_x][suradnice_y + 2] = i  # ak je velkost
auticka 3, tak treba vypisat este jedno policko
    i = i + 1
```

Ak sa autíčko vie posunúť bez toho, aby nastala kolízia alebo vyšlo mimo križovatky, tak sa posunie:

```
# Posun
if (auto[4] == 'h') and ((auto[3] - 1) + (auto[1] - 1) + dlzka_posunu
<= 5):  # ak auto je horizontalne, posunutelne
    if pole[auto[2] - 1][auto[3] - 1 + (auto[1] - 1) + dlzka_posunu]
== -9:  # ak je v smere posunu volne miesto
        riadok = auto[3] + dlzka_posunu  # posunie sa
        auto[3] = riadok
        return auto  # vrati auto
    else:
        return None
else:
    return None
```

Prehľadávanie do hĺbky

Na vyriešenie zadania je určený tento rekurzívny algoritmus na prehľadávanie všetkých vrcholov grafu alebo stromovej dátovej štruktúry, čo znamená návštevu všetkých uzlov grafu s určitými pravidlami.

Tento algoritmus najprv uprednostňuje prehľadávanie do hĺbky. V tomto zadaní to znamená to, že vždy pozrie prvý možný stav, kde sa vie dostať a z toho prvého možného stavu znova pozrie do ďalšieho možného stavu až pokiaľ nemá mu stav žiadny nový stav neponúka a iba sa vrátiť na posledný stav, ktorý mu ponúkal viacej stavov a vyberie si ďalší stav.

V tejto dokumentácii bol algoritmus docielený rekurzívnou funkciou, zásobníkom a listom s navštívenými uzlami. Zásobník výborne hodí k tomuto algoritmu, pretože reprezentuje LIFO kontext, čo znamená pre uzol, že posledný príde a prvý odíde. Rekurzívna funkcia zistí, že ktoré auto sa vie pohnúť a ktorým smerom. Tieto potenciálne stavy vloží do zásobníka a uzol, ktorý práve prebádala táto funkcia sa vloží do listu s navštívenými uzlami, aby sa predišlo zacykleniu.

Najprv si na začiatku funkcie skontroluje uzol, že či nie je výstupom:

```
if vystup_auto in uzol: # ak je vystup v tomto uzli, tak skonci a
vypise statistiky a cestu
    print("Cesta sa nasla!")
    finalny_cas = time.time()
    print("Hlbka:" + str(level) + "\nCas trvania: " + str(finalny_cas
- cas) + " sekund\nPocet vytvorenych uzlov: "
          + str(len(zasobnik) + len(zasobnik)) + "\nPocet
navstivenych uzlov: " + str(len(navstivene)))
```

Ďalej program skontroluje, že či uzol sa nachádza v zásobníku a v už navštívených uzloch:

```
if uzol in zasobnik:    # Uzol vyhodi zo zasobnika, lebo prave je
navstiveny
    zasobnik.pop(zasobnik.index(uzol))
if uzol not in navstivene:  # Uzol prida do navstivenych ak v nich
nie je
    navstivene.append(uzol)
```

Teraz potrebuje zistiť, že či s autami vie pohnúť nejakým smerom. To spraví v cykle tým, že v uzle prechádza každé auto, skontroluje jeho otočenie a podľa toho sa ho snaží pohnúť daným smerom. Ak sa mu to podarí, tak skontroluje, či tento budúci stav sa už nenachádza v zásobníku alebo v už navštívených uzloch:

```
for auto in uzol:  # V cykle skusa pre kazde auto pohnut sa vlavo,
vpravo, ore, dole
    if auto[4] == 'h':  # Ak je auto horizontalne, tak skusa len
vpravo a vlavo
        dolava = copy.deepcopy(uzol)     # Skopiruje uzol
        kopia_auta = dolava[pocitadlo]  # Skopiruje i-te auto
        if vlavo(kopia_auta, 1, dolava) is not None:     # Ak sa dad
posunut auto dolava
            if uzol_predikcia(kopia_auta, pocitadlo, dolava) not in
navstivene:  # ak uzol nie je v navstivenych
                if dolava not in zasobnik:  # ani v zasobniku
                    zasobnik.append(dolava) # prida do zasobniku na
dalsie spracovanie
```

Nakoniec už len skontroluje, že či nie je prázdny zásobník a podľa toho ďalej prehľadáva stavy zo zásobníka:

```
if len(zasobnik) != 0:  # ak zasobnik nie je prazdny, tak dalej
prehladava
    vyhladavanie_do_hlbky(vstup_uzol, vystup_auto, zasobnik[-1],
level + 1, max_level, cas, id(uzol))
```

Prehľadávanie do šírky

K vyriešeniu zadania je určený aj tento rekurzívny algoritmus na prehľadávanie všetkých vrcholov grafu alebo stromovej dátovej štruktúry, čo znamená návštevu všetkých uzlov grafu s určitými pravidlami.

Tento algoritmus najprv uprednostňuje prehľadávanie do šírky. V tomto zadaní to znamená to, že vždy prezrie všetky možné stavy, ktoré môžu nastať v terajšom stave a následne pozerá prvý možný stav a jeho všetky možné stavy.

V tejto dokumentácii bol algoritmus docielený rekurzívnou funkciou, frontou a listom s navštívenými uzlami. Fronta sa výborne hodí k tomuto algoritmu, pretože reprezentuje FIFO kontext, čo znamená pre uzol, že prvý príde a prvý odíde. Rekurzívna funkcia zistí, že ktoré auto sa vie pohnúť a ktorým smerom. Tieto potenciálne stavy vloží do zásobníka a uzol, ktorý práve prebádala táto funkcia sa vloží do listu s navštívenými uzlami, aby sa predišlo zacykleniu.

Najprv si na začiatku funkcie skontroluje uzol, že či nie je výstupom:

```
if vystup_auto in uzol: # ak je vystup v tomto uzli, tak skonci a
vypise statistiky a cestu
    print("Cesta sa nasla!")
    finalny_cas = time.time()
    print("Hlbka:" + str(level) + "\nCas trvania: " + str(finalny_cas
- cas) + " sekund\nPocet vytvorenych uzlov: "
          + str(len(zasobnik) + len(zasobnik)) + "\nPocet
navstivenych uzlov: " + str(len(navstivene)))
```

Ďalej program skontroluje, že či uzol sa nachádza vo fronte a v už navštívených uzloch:

```
if uzol in fronta:    # Uzol vyhodi z fronty, lebo prave je
navstiveny
    fronta.pop(fronta.index(uzol))
if uzol not in navstivene:  # Uzol prida do navstivenych ak v nich
nie je
    navstivene.append(uzol)
```

Teraz potrebuje zistiť, že či s autami vie pohnúť nejakým smerom. To spraví v cykle tým, že v uzle prechádza každé auto, skontroluje jeho otočenie a podľa toho sa ho snaží pohnúť daným smerom. Ak sa mu to podarí, tak skontroluje, či tento budúci stav sa už nenachádza v zásobníku alebo v už navštívených uzloch:

```
for auto in uzol:  # V cykle skusa pre kazde auto pohnut sa vlavo,
vpravo, ore, dole
    if auto[4] == 'h':  # Ak je auto horizontalne, tak skusa len
vpravo a vlavo
        dolava = copy.deepcopy(uzol)    # Skopiruje uzol
        kopia_auta = dolava[pocitadlo]  # Skopiruje i-te auto
        if vlavo(kopia_auta, 1, dolava) is not None:    # Ak sa dad
posunut auto dolava
            if uzol_predikcia(kopia_auta, pocitadlo, dolava) not in
navstivene:  # ak uzol nie je v navstivenych
                if dolava not in fronta:  # ani vo fronte
                    fronta.append(dolava) # prida do fronty na dalsie
spracovanie
```

Nakoniec už len skontroluje, že či nie je prázdna fronta a podľa toho ďalej prehľadáva stavy z fronty:

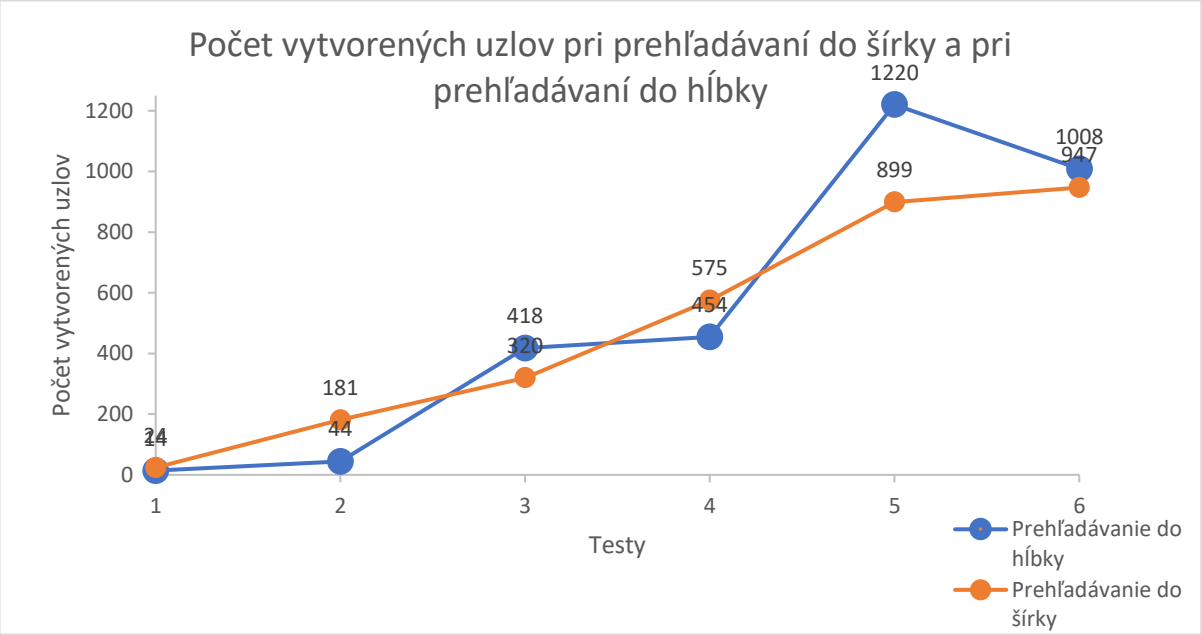
```
if len(fronta) != 0:  # ak fronta nie je prazdna, tak dalej
prehladava
    vyhladavanie_do_sirky(vstup_uzol, vystup_auto, fronta[0], level +
1, max_level, cas, id(uzol))
```

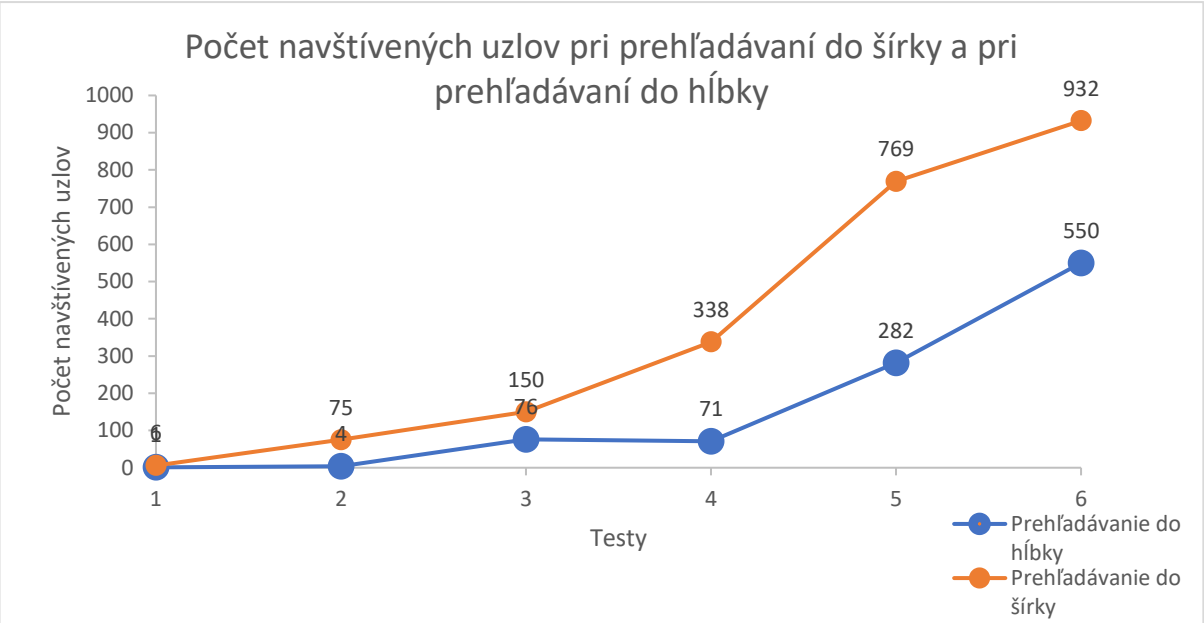
Spustenie

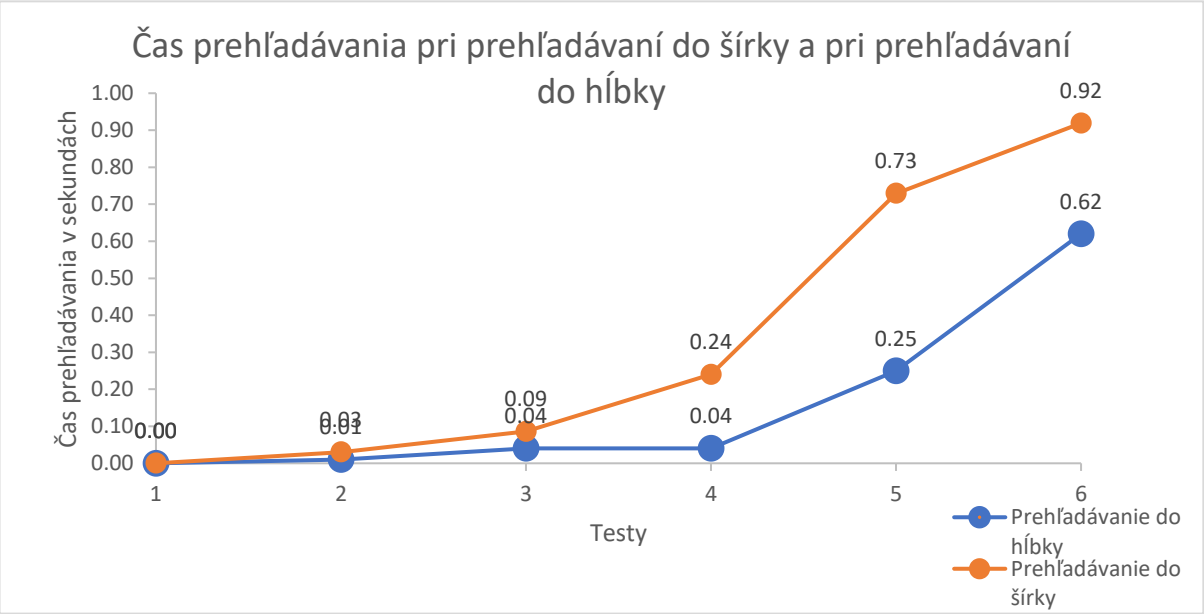
Program sa vždy spýta, že čo má spraviť. Zadaním príkazu „dfs“ sa spustí prehľadávanie do hĺbky. Zadaním príkazu „bfs“ sa spustí prehľadávanie do šírky. Program sa pri oboch príkazoch spýta na maximálnu hĺbku rekurzie a keď sa zadá -1, tak hĺbka nie je limitovaná. Vstupy a výstupy pre program sa dajú nájsť v textovom súbore „testovacie_vstupy.txt“, ktoré majú stúpajúcu zložitosť.

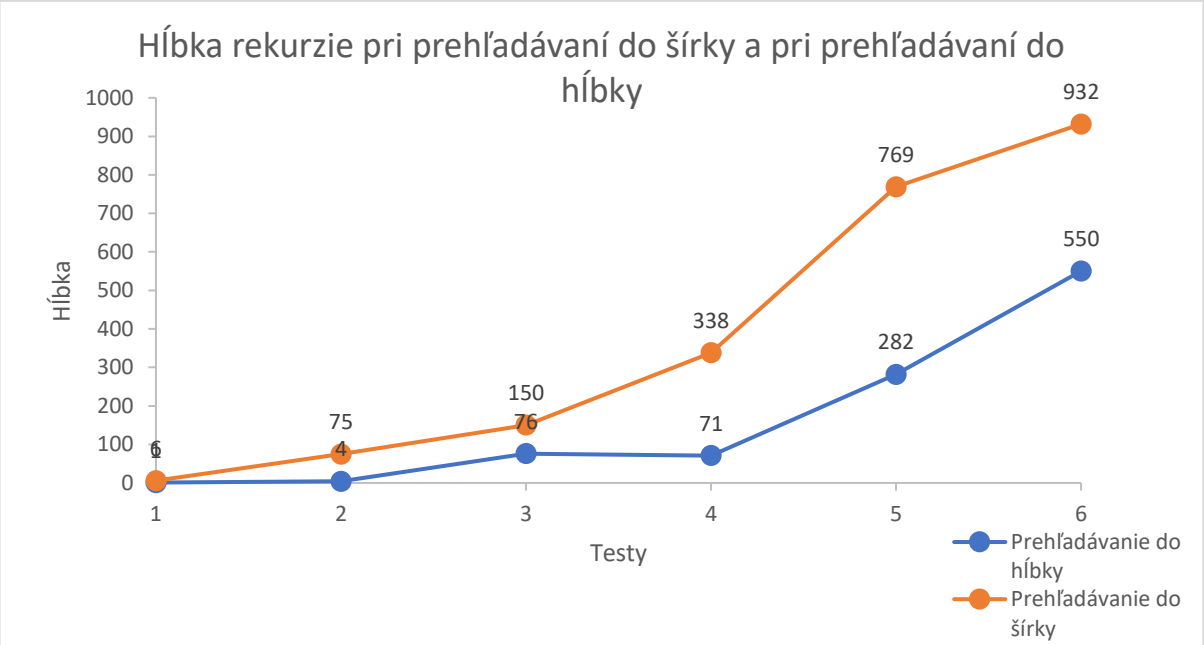
Testovanie

V testovaní bol meraný čas, hĺbka rekurzie, počet vytvorených uzlov a počet navštívených uzlov pri obidvoch algoritmoch na všetky vstupy z testovacieho súboru.




Počet navštívených uzlov pri prehľadávaní do šírky a pri prehľadávaní do hĺbky
Počet navštívených uzlov
1000
900
800
700
600
500
400
300
200
100
0
1
2
3
4
5
6
Testy
6
75
4
150
76
338
71
769
282
932
550
Prehľadávanie do hĺbky
Prehľadávanie do šírky


Čas prehľadávania pri prehľadávaní do šírky a pri prehľadávaní do hĺbky
Čas prehľadávania v sekundách
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
1
2
3
4
5
6
Testy
0.00
0.01
0.03
0.04
0.09
0.04
0.24
0.25
0.73
0.62
0.92
Prehľadávanie do hĺbky
Prehľadávanie do šírky


Hĺbka rekurzie pri prehľadávaní do šírky a pri prehľadávaní do hĺbky
Hĺbka
1000
900
800
700
600
500
400
300
200
100
0
1
2
3
4
5
6
Testy
6
75
4
150
76
338
71
769
282
932
550
Prehľadávanie do hĺbky
Prehľadávanie do šírky

Zhrnutie

Výhody prehľadávania do hĺbky nad prehľadávaním do šírky sú, že pri jednoduchých problémoch je tento algoritmus oveľa efektívnejší, rýchlejší a menej náročný na zdroje. Podľa grafov vidieť, že prehľadávanie do hĺbky má priemerne menší počet rekurzií funkcie, je priemerne rýchlejší a priemerne navštívi menšie množstvo uzlov. Pri zložitejších riešeniach začína prevládať prehľadávanie do šírky v počte vytvorených uzlov, pretože prehľadávanie do hĺbky sa môže ľahko dostať do zlej vetvy v stromovej dátovej štruktúre a zbytočne sa tam cyklí. Čo sa týka efektivity, tak program v tejto dokumentácii má celkovo veľmi dobrú časovú náročnosť, ale je pomerne náročný na pamäť, ale to vychádza z charakteristiky týchto algoritmov.