Zadanie 2

Michal Franczel, Fakulta Informatiky a Informačných Technológií, STU 25/03/2020

Zadanie

V mojom zadaní bolo úlohou nájsť riešenie hlavolamu **Bláznivá križovatka**. Tento hlavolam je reprezentovaný na mriežke fixnej veľkosti 6 krát 6, na ktorej sa nachádzajú autá dĺžky 2 alebo 3 políčka orientované horizontálne alebo vertikálne, ktoré sa môžu pohybovať smerom dopredu alebo dozadu. Hlavolam sa považuje za vyriešený, ak sa červené auto nachádza na pravom okraji hracej plochy. Hlavolam bol riešený použitím algoritmu cyklicky sa prehlbujúceho hľadania, ktorý by mal nájsť optimálnu postupnosť krokov na vyriešenie problému.

Implementácia

Program bol implementovaný v jazyku *Go*.

Reprezentácia údajov

Stav riešenia hlavolamu je reprezentovaný pomocou dvoch štruktúr. V hlavnej štruktúre Stav sa nachádza hĺbka stavu v prehľadávacom strome, zoznam áut, ktoré sa v ňom nachádzajú, zoznam krokov, ktoré boli vykonané od počiatočného stavu a mapa reprezentovaná dvojdimenzionálnym poľom. V jednotlivých autách sa nachádzajú ich súradnice, dĺžky, smer a farba. Každý krok je reprezentovaný tromi číslami - smerom, indexom auta, ktoré sa pohlo a veľkosť kroku, o ktorú sa pohlo.

```
1  type Auto struct{
2      X uint8
3      Y uint8
4      Farba string
5      Smer bool
6      Dlzka uint8
7  }
8
9  type Stav struct {
10     Hlbka int
11     Auta []Auto
12     Kroky []int
13     Mapa[6][6] uint8
14 }
```

Použitý algoritmus

Na nájdenie riešenia bolo použité cyklicky sa prehlbujúce prehľadávanie no hĺbky. Tento algoritmus spočíva v tom, že sa cyklicky vyvoláva ohraničené prehľadávanie do hĺbky pre postupne sa zvyšujúce hĺbky, pričom sa tento cyklus preruší v prípade nájdenia riešenia alebo pri vyčerpaní stavov.

```
1  func najdiRiesenie(pociatocnyStav Stav, finalneAuto string) *Stav {
2      pocitadlo := 0
3      for i := 0; i < 50; i+=1 {
4
5          najdene := dls(pociatocnyStav, i, finalneAuto, &pocitadlo)
6          if najdene != nil {
7              fmt.Println("Prehladalo sa ", pocitadlo, " stavov..")
8              return najdene
9          }
10     }
11     fmt.Println("Prehladalo sa ", pocitadlo, " stavov..")
12     return nil
13 }
```

V ohraničenom prehľadávaní so hĺbky využívam zásobník, do ktorého dám najprv začiatočný stav. Ten v cykle vyberiem, nájdem jeho potomkov a uložím ich do zásobníka. Následne vyberiem posledný stav zo zásobníka a znovu hľadám jeho potomkov. Takto postupne vyberám pokiaľ nie je zásobník prázdny, alebo kým nenájdem požadovaný cieľový stav. Ak je stav vybraný zo zásobníka už navštívený alebo ak je jeho hĺbka väčšia alebo rovná hraničnej hĺbke, jeho potomkov neprehľadávam a nepridávam do zásobníka.

```
1  func dls(zacStav Stav, hlbka int, finAuto string, poc *int)(*Stav) {
2      zacStav.Hlbka = 0
3      zacStav.Mapa = vytvorMapuZAut(zacStav.Auta)
4      //hash mapa, kde sa uchovavaju navstivene stavy
5      hshMapa := make(map[uint64]int)
6      //stavy, ktore su na okraji (este neprehladane)
7      okraj := make([]Stav, 0)
8      okraj = append(okraj, zacStav)
9
10     //prehladavany stav
11     var prehlStav Stav
```

```
12
13      for len(okraj) != 0 {
14          //pop stavu, ktory sa ide navstivit
15          prehlStav = okraj[len(okraj)-1]
16          okraj = okraj[:len(okraj)-1]
17          h := hash(prehlStav)
18
19          //ak sa nasiel finalny stav, vrat ho, inak pokracuj v prehladavani
20          if prehlStav.Hlbka == hlbka && is_state_final(prehlStav, finAuto) ↩
                {
21              return &prehlStav
22          } else if hshMapa[h] == 0 || hshMapa[h] > prehlStav.Hlbka{
23              //ak stav este nebol navstiveny, alebo bol navstiveny, ale ↩
                    hlbsie -> navstiv ho
24              hshMapa[hash(prehlStav)] = prehlStav.Hlbka
25              //ak je hlbka vramci ohranicenia, najdi deti a pushni do ↩
                    stacku
26              if prehlStav.Hlbka < hlbka{
27                  *poc += 1
28                  children := najdiDeti(prehlStav)
29                  okraj = append(okraj, children...)
30              }
31          }
32      }
33
34      return nil
35  }
```

Potomkovia sa vyhľadávajú tak, že postupne prechádzam všetky autá v stave a postupne ich posúvam o krok veľkosti 1 až 4 dopredu aj dozadu, pričom sa vždy overuje, či je daný posun možný.

Testovanie

Na testovanie bolo vytvorených päť rôznych konfigurácií áut. Tieto konfigurácie mali variabilné počty áut a boli zoradené podľa krokov, ktoré treba na ich vyriešenie. Program bol teda testovaný na 4 scenároch, kde bolo treba 2, 4, 6, 8, 15 krokov na vyriešenie problému a jeden scenár bol neriešiteľný.

	Čas	Navštívené stavy
2 kroky	121 µs	3
4 kroky	6.4 ms	83
6 krokov	70.3 ms	1377
8 krokov	289.2 ms	6699
15 krokov	5.574 s	102147

Zhodnotenie

Cyklicky sa prehlbujúce prehľadávanie nachádza optimálne riešenie, čím vylepšuje prehľadávanie do hĺbky a vďaka čomu je v konečnom dôsledku vhodným algoritmom pre zadaný problém. Rieši základný problém prehľadávania s ohraničením, ktorým je voľba vhodnej hraničnej hĺbky. Aj napriek tomu, že môže byť časovo náročnejšie, vďaka vlastnosti prehľadávacích stromov (na každej ďalšej úrovni je oveľa viac stavov ako v predošlej) nie je tento rozdiel v náročnosti až tak priepastný. V porovnaní s prehľadávaním do šírky je výhodný najmä pre svoju pamäťovú zložitosť.

Moja implementácia zároveň bola vylepšená o ukladanie navštívených stavov pomocou hašovacej tabuľky, vďaka čomu je dramaticky znížený počet stavov, ktoré treba navštíviť za účelom nájdenia riešenia. Vďaka tomu, že bola moja implementácia implementovaná v programovacom jazyku go, je zároveň moja implementácia značne rýchlejšia ako implementácie v iných programovacích jazykoch, akým je napríklad Python.