Bludiště

1 Popis řešení

K hlednání nejkratší cesty použijeme modifikovaný BFS algoritmus. Algoritmus si bude pamatovat, ke kterým klíčům má přístup, t.j. jaké různé barvy klíčů již při prohledávání objevil. Při prohledávání pak bude dveře, k jejichž patřičným kličům má přístup, považovat za volně průchozí prvky a ostatní dveře za zdi.

Ve chvíli, kdy prohledávání narazí na dosud nenalezený, spustí se nová instance prohledávání vycházející z místa nalezení klíče a přidávající právě nalezený klíč na svůj seznam klíčů. V momentě, kdy najde tento dceřinný algoritmus cestu do cíle, uloží si algoritmus, který ho spustil, délku jeho cesty jako vrácenou hodnotu zvýšenou o délku cesty od startu ke klíči.

Algoritmus si tedy musí mimo jiné pamatovat i délku nejkratší dosud nalezené cesty, kterou změní, když skončí dceřinné vyhledávání s celkovou délkou kratší než je dosavadní uložená. V momentě, kdy prohledávání skončí, tedy najde vlastní cestu k cíli nebo projde všechny vrcholy, vrátí délku nejkratší nalezené cesty.

2 Pseudokód

```
main()
{
  int delkaCesty = NajdiCestu(start, prazdny seznam);
}

int NajdiCestu(Vrchol start, Seznam klicu){
  int nejkratsiCesta = nekonečno;
  while (fronta neni prazdna) {
     Vrchol v = vyber z fronty;
     if (v == zed || (v == dvere && nemam klic))
       continue;
     if (v == princezna){
       if (v.delkaCesty < nejkratsiCesta)
         nejkratsiCesta = v.delkaCesty;
       return nejkratsiCesta;
     }
     if (v == klic && nemam klic v){
       meziCesta = NajdiCestu(v, Seznam klicu + v) + v.delkaCesty;
       if (meziCesta < nejkratsiCesta)
```

```
            nejkratsiCesta = meziCesta;
        }

        navstiveno[v] = true;
        foreach (u nenavstiveny soused v){
           u.delkaCesty = v.delkaCesty+1
           Pridej na frontu (u)
        }
    }
    return nejkratsiCesta;
}
```

3 Důkaz správnosti

Jelikož BFS používá frontu k prohledávání, můžeme vyhledávání ukončit, když dojdeme k cíli, protože každé další výskyty cíle budou mít minimálně stejnou délku cesty. Algoritmus prochází všechny prvky které může, to jest volně průchozí vrcholy a dveře, k nimž má klíč. To znamená, že při prohledávání nevynechá žádnou možnou kratší cestu.

Každá instance procedury má zároveň vlastní seznam navštívených vrcholů, tedy ve chvíli, kdy narazíme na klíč, nově spuštěná procedura může znovu objevit již prošlé vrcholy, tudíž opět nevynecháváme žádnou možnou cestu. Víme také, že každé prohledávání samo o sobě nachází nejkratší cesty do všech objevených vrcholů, tedy délka celkové cesty od startu do cíle vedoucí přes nalezený klíč se bude rovnat délce cesty ke klíči přičtené k délce cesty od klíče do cíle. Můžeme tedy tento součet porovnávat s nejkratší nalezenou cestou.

Nejkratší cesta se mění, jen když cesta nalezená přes klíč nebo cesta bez klíčů je kratší než nekratší cesta doposud nalezená, tudíž si vždy pamatujeme délku nejkratší cesty do cíle, kterou pak vracíme.

4 Časová složitost

Na první úrovni volacího zásobníku bude spuštěna jedna instance prohlednávání. Pokud objeví před princeznou všechny klíče, bude na druhé úrovni zásobníku spuštěno k instancí. Každá z těchto k instancí může spustit až dalších $k-1$ instancí, tedy celkem $k \cdot (k-1)$.

Celkem tedy může být zavoláno až $\left(\sum_{i=0}^{k-1} \prod_{j=0}^{i} (k-i)\right) + 1$ instancí prohledávání, což můžeme shora omezit pomocí $k \cdot k!$.

Mimo rekurze je prohledávání vzhledem k BFS modifikováno jen o konstantní množství operací, tedy běží v $O(n+m)$. Výsledná časová složitost tedy náleží do $O(k \cdot k!(n+m)$. Pokud považujeme k za konstantní, je složitost v $O(n+m)$.

5 Prostorová složitost

Na volacím zásobníku bude vždy nejvýše $k+1$ instancí procedury prohledávání. Procedura si pamatuje informace o vrcholech a hranách ($O(n+m)$), nejkratší dosavadní cestu ($O(1)$) a seznam dostupných klíčů ($O(k)$). Celková prostorová složitost programu nám tedy z tohoto vychází jako $O(k \cdot (n+m+k))$. Pokud k považujeme za konstantu, je prostorová složitost v $O(n+m)$.