KSI 2014/2015

Úloha 5-3: Karlík a poklad

Jan Horáček
Gymnázium, Brno, Vídeňská 47; jan.horacek@seznam.cz

30. dubna 2015

1 Nápad

Cílem tohoto textu je nalézt a popsat algoritmus řešící zadanou úlohu v lineárním čase vůči velikost grafu – přeneseně řečeno "každý vrchol a hranu projít maximálně jednou". I když se níže popisované řešení může zdát komplikovanější, věřím, že nároky na lineární časovou složitost splňuje.

Reprezentujme mapu k pokladu grafem, kde jednotlivá políčka mapy jsou vrcholy grafu a povolené cesty mezi políčky (ať už pro Karlíka, nebo pro vodu) jsou hranami takového grafu. Body se stalagmity v grafu nejsou.

Rozdělme si úlohu na několik částí:

1. Přiřaďme každému bodu mapy jeho vzdálenost od bodu $[x_v, y_v]$
2. Na základě údajů vypočtených v bodě 1) hledejme čas, ve kterém je možné cestu projít nejvýše s jedním ponořením na jednu sekundu.

2 Výpočet vzdálenosti

Vzdálenost v našem případě není klasická Euklidovská, ale spíše Newyorská – resp. měřená v Newyorské metrice. Každému bodu přiřadíme jeho vzdálenost *vzdalnost* od bodu, kde vytryskává voda – bodu $[x_v, y_v]$. Nechť jsou vzdálenosti od startu uložené v matici $vzdalenosti[x, y]$ indexované polohou vrcholu od nuly.

Takové vzdálenost vypočteme pomocí mírně upraveného prohledávání do šířky, ve kterém využijeme dvě fronty, které budeme postupně prohazovat. Nechť je zpočátku celá matice *vzdalenosti* inicializovaná na hodnotu -1 a přístup k matici mimo její indexy vrací hodnotu 0.

Předpokládejme také, že cesty mezi vstupem do jeskyně, pokladem a výtryskem s vodou nejsou úplně oddělené stalagmity.

```
1 def vypocti_vzdalenosti(xp, yp):
2         vzdalenost = 0
3         Q1 = Queue()
4         Q1.Enque(Point(xp, yp))
5         Q2 = Queue()
6
7         while not Q1.empty:
```

```
8                   bod = Q1.Deque()
9                   vzdalenosti[bod.X, bod.Y] = vzdalenost
10
11                  # vpravo
12                  novy_bod = Point(bod.X+1, bod.Y)
13                  if (vzdalenosti[novy_bod.X, novy_bod.Y] == -1) and (not
                        Stalaktit(novy_bod)): Q2.Enque(novy_bod)
14
15                  # vlevo
16                  novy_bod = Point(bod.X-1, bod.Y)
17                  if (vzdalenosti[novy_bod.X, novy_bod.Y] == -1) and (not
                        Stalaktit(novy_bod)): Q2.Enque(novy_bod)
18
19                  # nahore
20                  novy_bod = Point(bod.X, bod.Y-1)
21                  if (vzdalenosti[novy_bod.X, novy_bod.Y] == -1) and (not
                        Stalaktit(novy_bod)): Q2.Enque(novy_bod)
22
23                  # dole
24                  novy_bod = Point(bod.X, bod.Y+1)
25                  if (vzdalenosti[novy_bod.X, novy_bod.Y] == -1) and (not
                        Stalaktit(novy_bod)): Q2.Enque(novy_bod)
26
27                  Prohod(Q1, Q2)
28                  vzdalenost++
```

3 První odhad maximálního času

Tato vzdálenost má pro nás důležitý význam: každému políčku říká, za kolik sekund od počátku času bude zaplaveno. Na základě nyní vypočtených údajů můžeme provést první přiblížení maximálního času, za který se Karlík může dostat ven z jeskyně. Jelikož zadání říká, že vstupní políčko nemůže být zaplaveno, resp. vstoupením na něj se dostáváme ven z jeskyně nezávisle na tom, jestli je zaplaveno, nebo ne, je první odhad nejdelšího času čas, ve který bude zaplaveno políčko s pokladem. Nechť tedy $max_cas = vzdalenosti[x_p, y_p] - 1$. Čas max_cas je pro nás teď ideálním odhadem času, o kterém víme, že v něm zůstane nezaplavené políčko s pokladem, ale ještě nevíme nic o tom, jestli je volná cesta mezi políčkem s pokladem k východem z jeskyně. Tento čas tedy budeme postupně zkracovat algoritmem popsaným níže.

4 Přesný výpočet času

Přesný výpočet času, po který může Karlík zůstat v jeskyni a plnit batoh drahokamy, provedeme pomocí dvojitého iterativního prohledávání do šířky.

Konkrétně budeme postupně zmenšovat proměnnou max_cas a dívat se, jestli v tomto čase už je cesta průchozí. Jakmile narazíme na průchozí cestu, máme jisto, že