Juraj Holas

# Domáca úloha č. 3

## 1.

Pózy a postupnosti medzi nimi môžeme ľahko reprezentovať ako vrcholy v grafe a cesty medzi nimi. Hľadanie postupnosti animácií, ktorá sa čo najrýchlejšie dostane z jednej animácií do druhej, je tým pádom hľadanie najkratšej cesty v grafe. Tento graf pritom vytvoríme pomocou nasledujúcich pravidiel:

* Za každú pózu pridáme jeden vrchol:
* Medzi každou dvojicou vrcholov pridáme orientovanú hranu s váhou
* Medzi každou dvojicou vrcholov a pridáme orientovanú hranu s váhou 0 vtedy, ak , resp.

Počet vrcholov v takto skonštruovanom grafe je za každú dvojicu , plus vrchol , spolu teda . Počet hrán nevieme presne vyjadriť, vieme ho však ohraničiť zhora. Máme hrán medzi vrcholmi , maximálne hrán medzi , a maximálne hrán medzi . Spolu teda v grafe bude najviac hrán.

Na výslednom grafe môžeme nakoniec spustiť jeden zo štandardných algoritmov pre nájdenie najkratšej cesty v grafe, napr. Dijkstrov algoritmus, začínajúc vo vrchole a končiac v požadovanom vrchole . Nakoľko graf je orientovaný a vzhľadom na postup jeho konštrukcie bude výsledok Dijkstrovho algoritmu cesta v tvare . Našu hľadanú postupnosť animácií potom získame pozbieraním indexov jednotlivých vrcholov , čiže postupnosť .

V našom vytvorenom grafe sa dá hýbať iba dvomi spôsobmi:

1. z hrany do , čo reprezentuje vykonanie animácie, pričom cena tohto kroku je , čiže čas potrebný na vykonanie animácie
2. z hrany do , čo reprezentuje preskočenie z animácie na animáciu . Tento krok je možný iba ak sa od seba koniec prvej a začiatok druhej líšia o povolenú hodnotu, čo sme zaručili pri konštrukcii grafu. Zároveň netrvá žiaden čas, preto je jeho cena nulová.

Z toho vyplýva, že Dijkstrov algoritmus bude naozaj vyhľadávať postupnosť animácií zaberajúcu čo najmenej času.

Zložitosť samotného Dijkstrovho algoritmu je . V našom prípade teda:

Zložitosť úpravy vstupu na potrebný graf reprezentovaný napr. tabuľkou susedov je:

Celková zložitosť algoritmu je teda , čiže . To všetko za predpokladu, že . V prípade že metóda je zložitejšia tak celkovú zložitosť získame analogickým postupom: .

Príklad: vezmime si príklad zo zadania o dvoch animáciách, začiatočnej póze a konečnej , pričom pre zaujímavosť zmeníme podobnosť . Výsledný graf bude vyzerať nasledovne:

Najkratšia cesta z  do získaná Dijkstrovým algoritmom bude , čo predstavuje postupnosť animácií .

## 2.

Vstup tohto príkladu si opäť jednoducho prerobíme na graf:

* za každú pózu jeden vrchol:
* za každú animáciu pridáme jednu orientovanú hranu s váhou

Nájdenie určitej postupnosti animácií je teda hľadanie sledu v takomto grafe.

Postup som pre jednoduchosť zápisu vyjadril vo forme nedeterministického algoritmu. Jeho princíp spočíva v tom, že najprv v grafe nájde sled dĺžky najviac , a následne overí, či v grafe existuje ešte dlhší sled neprekračujúci dĺžku . Ak taký už neexistuje, tak prvý spomínaný sled je výsledok, pričom postupnosť hrán v ňom reprezentuje hľadanú postupnosť animácií.

Kľúčom celého algoritmu je funkcia findMinMax(v\_start, v\_end, min, max), ktorá nedeterministicky nájde sled začínajúci vo vrchole , končiaci vo vrchole , ktorého cena je väčšia ako a väčšia alebo rovná .

1 function findMinMax(v\_start, v\_end, min, max){

2 price := 0

3 walk := [v\_start]

4 v := v\_start

5 while(true){

6 u := choose from v.neighbours

7 price += w(v, u)

8 walk.append(u)

9 if(v==v\_end && choose from {true, false}){

10 break

11 }

12 }

13 if(min < price <= max){

14 accept(walk)

15 }

16 else{

17 reject

18 }

19 }

Samotné hľadanie postupnosti potom prebieha nasledovne:

20 function find(v\_start, v\_end, k){

21 walk1 := findMinMax(v\_start, v\_end, 0, k)

22 walk2 := findMinMax(v\_start, v\_end, walk1.price, k)

23 if(walk2 == null){

24 accept(walk1)

25 }

26 else{

27 reject

28 }

29 }

Zložitosť algoritmu:

* choose na riadku 6 generuje číslo veľké nanajvýš , čiže -bitové číslo, čiže trvá
* cena je nanajvýš , preto sčítanie na riadku 7 počíta s -bitovými číslami, čiže trvá
* porovnanie vrcholov (-bitových čísel) na riadku 9 trvá