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 animácie 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.

Na začiatku si vrcholy animácie uložíme do štruktúry na spôsob tabuľky susedov: pre každú pózu si budeme pamätať zoznam póz, do ktorých sa vieme z  dostať nejakou jednou animáciou, pričom uloženú budeme mať aj informáciu o danej animácii. Napr. pre príklad zo zadania:

1: (pose: 2, animation: 1), (pose: 3, animation: 3)

2: (pose: 1, animation: 2)

3:

Ďalej si vytvoríme T – pole množín póz, ktorého veľkosť bude , a na začiatku budú všetky množiny v ňom prázdne. Množina T[i] reprezentuje zoznam takých póz, do ktorých sa vieme dostať v čase rovnému i. Ku každej z týchto póz si navyše budeme pamätať akou animáciou sme sa do nej dostali. Vyplnené pole T pre príklad zo zadania by teda vyzeral nasledovne:

T[0] = {(p: 1, a: null)}

T[1] = {(p: 2, a: 1), (p: 3, a: 3)}

T[2] = {}

T[3] = {(p: 1, a: 2)}

T[4] = {(p: 2, a: 1), (p: 3, a: 3)}

T[5] = {}

T[6] = {(p: 1, a: 2)}

T[7] = {(p: 2, a: 1), (p: 3, a: 3)}

T[8] = {}

T[9] = {(p: 1, a: 2)}

Takéto vyplnené pole teda zachytáva jednak všetky pózy, do ktorých sa vieme dostať v čase i, ale zároveň aj cestu – postupnosť animácií, ktorou sme sa do danej pózy dostali (znázornené šípkami). Z tohto poľa vieme následne aj rýchlo získať výsledok – pozrieme sa, v ktorej najvyššej množine (T[i] s čo najvyšším i) sa nachádza cieľový vrchol. V našom prípade sa vrchol 3 nachádza naposledy v množine T[7] – 7 je teda dĺžka najdlhšej možnej postupnosti animácií. Samotnú postupnosť potom získame spätným prechodom: pre aktuálnu pózu T[i] si pozrieme akou animáciou sme sa do nej dostali, a presunieme sa na pózu T[i-dj]. Toto opakujeme kým sa nedostaneme do štartovacej pózy v T[0]. V našom prípade takto pozbierame pózy , čo je po obrátení presne náš hľadaný výsledok.

Ostáva naplniť pole T, čo vykonáme nasledujúcim postupom:

1. inicializujeme T[0] tým, že doňho pridáme počiatočnú pózu, a počítadlo i nastavíme na 0
2. pre každú pózu z T[i] sa pozrieme na všetky jej susedné pózy podľa tabuľky susedov
3. každého z týchto susedov pridáme do množiny T[i+d] , kde d je dĺžka animácie k tomuto susedovi
4. keď sme vykonali kroky 2 a 3 pre všetky pózy v T[i] a všetkých ich susedov, zvýšime i
5. ak je i väčšie od , tak je pole vyplnené, inak zvýš i o jedna a pokračuj od bodu 2

**Pseudokód pre daný algoritmus:**

1 function sequence(p\_start, p\_end, k){

2 Neighbours := create\_neighbour\_table();

3 T := array[0..k] of set of (pose, animation);

4 T[0] := {(pose: p\_start, animation: null)};

5 for i := 0 to k do {

6 for every pose P in T[i] do {

7 for every (pose, animation) Q in Neighbours[P] do {

8 if( (Q.pose, whatever) not in T[i + Q.animation.duration] ){

9 T[i + Q.animation.duration].add(Q);

10 }

11 }

12 }

13 }

14 return extract\_result\_from(T);

15 }

**Časová zložitosť:**

* tabuľku susedov (r. 2) vieme jednoducho vytvoriť v čase
* inicializácia poľa T (r. 3) nám zaberie čas
* cyklus na r. 5 sa vykoná -krát
* cyklus na r. 6 sa vykoná nanajvýš -krát, nakoľko v množine môžu byť nanajvýš všetky pózy, nemôžu sa ale opakovať
* cyklus na r. 7 sa rovnako vykoná nanajvýš -krát, z obdobného dôvodu
* test či prvok patrí množine (r. 8) ako aj pridanie do množiny (r. 9) vieme pri správnej implementácií, napr. HashSet, vykonať v čase
* získanie výslednej postupnosti (r. 14) spôsobom ako sme si ukázali sa vykoná v čase

Celková časová zložitosť je teda: