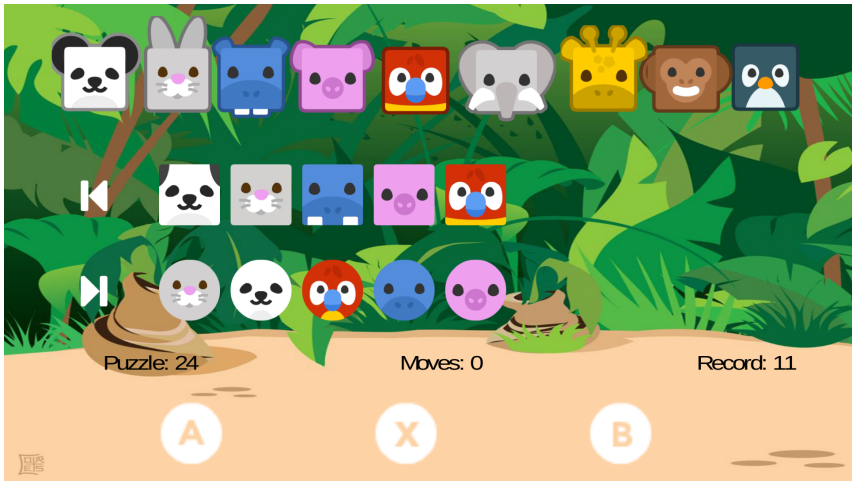
DAKI Programmeeropdracht 6: Tricky Animals

(Deze versie is van 26 oktober 2022, 12:57)

Je moet deze programmeeropdracht **zelf** en **alleen** maken. Je mag zeker met anderen overleggen over je aanpak, maar code van anderen bekijken of overnemen of zelf code delen met anderen is uitdrukkelijk niet toegestaan. Je moet je programma schrijven in C#, en inleveren via DOMjudge.

1 Opdrachtbeschrijving

Na al die intellectueel hoogstaande opdrachten is DAKIBOT toe aan een spelletje. Ze heeft een leuk spelletje ontdekt, met dieren—waar ze heel erg van houdt. $O(mg)$, ze zijn echt supercute! Kijk maar in Figuur 1! Het spel heet Tricky Animals en de bedoeling is een lijst



Figuur 1: Screenshot van het spel Tricky Animals.

met dieren te sorteren naar een gegeven volgorde. In de figuur zie je een instantie met $n = 5$ dieren. Als je bovenaan op de olifant tikt, krijg je een instantie met zes dieren, etc. Je kunt ook met de pijlen aan de linkerkant door de lijst met instanties browsen, en als je in het lege deel rechts tikt nadat je een instantie hebt opgelost, krijg je de volgende in de lijst. Een instantie bestaat uit twee rijtjes met dieren: het is de bedoeling de ronde plaatjes in de onderste rij in de volgorde van de vierkante plaatjes in de bovenste rij te krijgen. Je kunt hiervoor onder in beeld op de knoppen '**A**', '**X**' en '**B**' tikken. (We gebruiken hieronder en ook **op DOMjudge kleine letters!**) Deze drie operaties hebben de volgende gevolgen.

a: verwisselt de eerste twee elementen, dus op plekken 1 en 2.

b: verwisselt de laatste twee elementen, dus op plekken $n-1$ en n.

x: roteert de elementen op plek 2 tot en met $n-1$ rechtsom. Het element op plek 2 verplaatst naar plek 3, het element op plek 3 verplaatst naar plaatst 4, ..., het element op plek $n-2$ verplaatst naar plek $n-1$, en het element op plek $n-1$ verplaatst naar plek 2.

Om gevoel te krijgen voor dit spel, kun je het online spelen op http://www.tricky-animals.de. (Ik moest voorheen "shields down" doen in mijn browser (Brave) voordat het werkte, maar zojuist kreeg ik een unity-gerelateerde foutmelding...) Ik heb de instantie in Figuur 1 ooit opgelost in 11 zetten, en zonet in 14 zetten. Het is instantie nummer 24 (die je krijgt als je op de rode ... papegaai (?) klikt), kun jij het in minder zetten? Voor alle duidelijkheid: dat is niet het doel van deze programmeeropdracht. De opdracht is wél om voor DAKIBOT een algoritme te modellontwerpen en (*daarna*) te implementeren waarmee ze een instantie kan oplossen in *zo min mogelijk* zetten.

2 Invoer en uitvoer

In het echte spel is de input een lijst van (tekeningen van) dieren. Op DOMjudge sorteren we een rij letters, en het is de bedoeling om ze op alfabetische volgorde te zetten. De operaties zijn wel hetzelfde als in het spel. Een inputbestand heeft meerdere regels. De eerste regel bevat een getal n. Daarna volgen n regels, met op iedere regel één string letters die je moet sorteren (verder niks). Als output hoef je niet de gesorteerde letters te geven, maar je programma moet printen in hoeveel stappen je van de inputstring naar de gesorteerde string kunt komen, en een volgorde van operaties geven waarmee dat kan. Om precies te zijn, moet je voor een gegeven inputbestand n regels schrijven, met op iedere regel het aantal operaties dat je hebt gevonden, dan een spatie, en dan de lijst van operaties.

3 Voorbeeld

Bij de volgende input

```
2
bcdea
adbecf
```

hoort bijvoorbeeld de volgende output

```
3 bxa
9 xbxbxxxbx
```

De eerste input kun je met 3 operaties sorteren, en niet met minder dan 3. Je doet eerst operatie **b** om de twee meest rechtse karakters '`e`' en '`a`' om te wisselen (dan is de string '`bcdae`'), dan doe je operatie **x** om de middelste drie karakters naar rechts te roteren (met als resultaat '`bacde`'), en tenslotte doe je operatie **a** om de eerste twee karakters '`b`' en '`a`' om te wisselen, zodat je '`abcde`' overhoudt, en de string gesorteerd is. Probeer zelf de tweede inputstring '`adbecf`' te sorteren, of verifieer in elk geval dat de 9 operaties **xbxbxxxbx** dit doen.

4 Algoritmische eisen

Je ziet dat het aantal benodigde operaties snel kan oplopen, van drie operaties voor een string van vijf letters, naar negen operaties voor een string van slechts één letter meer. Alles hangt natuurlijk af van de sortering van de inputstring. Het is heel belangrijk dat je algoritme snel genoeg is. Als enige algoritmische eis geldt dat je het probleem moet representeren als graaf, en moet oplossen met een graaf-zoekalgoritme. Je mag dus geen (al dan niet zelf geschreven) sorteeralgoritme aanroepen.

5 Hints, tips

In de voor de hand liggende graaf om over dit probleem na te denken, is elke mogelijke permutatie van letters een knoop. De kanten representeren operaties, en zijn dus gericht. Tussen twee knopen i en j bestaat een kant, gelabeld met een operatie (zeg k), als je vanuit permutatie i met operatie k in permutatie j terecht komt. De graaf kan cykels bevatten, want als je op permutatie i bijvoorbeeld twee keer operatie **b** uitvoert, ben je weer terug bij permutatie i, maar ook $n - 2$ keer operatie **x** brengt je terug waar je begon. Hoe dan ook, in deze graaf geeft elk kortste pad tussen de beginknoop (de input) en de eindknoop (de gesorteerde input) een correcte oplossing. Dit kortste pad zou je kunnen vinden met behulp van een kortste-pad algoritme, maar het is niet praktisch—althans, niet eentje uit hoofdstukken 22 of 23 van CLRS.

Wat ik bedoel is dat je niet daadwerkelijk een graaf moet gaan bouwen, om daarin met een korste-pad algoritme te gaan zoeken. Je hoeft namelijk in veel gevallen maar een klein deel van de ('impliciete') graaf te doorzoeken om bij je antwoord te komen. Het kost dan relatief veel tijd en geheugen om eerst de graaf helemaal op te bouwen, en er daarna in te gaan zoeken. Er zijn nou eenmaal $n!$ permutaties van n verschillende letters, en dat aantal is zelfs méér dan exponentieel in n. Het is dus beter (sneller) om een 'partiële' graaf te gebruiken, waarin je alleen de knopen en kanten toevoegt die je daadwerkelijk tegenkomt bij het exploreren. Let ook op dat je bij elke nieuwe knoop checkt of je hem al gezien hebt, vanwege het feit dat er cykels in de (impliciete) graaf zitten.