DAKI Programmeeropdracht 5: Toetsenspion

(Deze versie is van 28 mei 2020, 00:44)

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

Je hebt een weekje geleden een toets gemaakt voor het vak Acroniemen Kunnen Intelligente Dingen Ontzettend Kunstmatig Invalideren (AKIDOKI), maar omdat je weet dat de mogelijkheid bestaat dat die toets niet zo goed is gegaan, besluit je alvast voorzorgsmaatregelen te nemen om jezelf op een (nogal kunstmatig) intelligente manier te helpen. Je plan is om in te breken in het Osiris account van je docent dr. Tamos Klas, zodat je je cijfer zult kunnen veranderen, mocht dat nodig blijken te zijn. Hiervoor moet je z'n wachtwoord hebben, dus je programmeert DAKIBOT om in te breken in de woning van je docent, die tegenwoordig altijd thuis werkt, om op z'n laptop een 'keylogger' te installeren—hardware of software die toetsaanslagen registreert, om zo bijvoorbeeld een ingevoerd wachtwoord te kunnen achterhalen. Het nieuws van je inspanningen verspreidt zich als een lopend vuurtje, en je robot gaat ook voor al je vrienden en vriendinnen (en recursief voor die van hen, natuurlijk) aan de slag om ook bij hún docenten in te breken en keyloggers te installeren.

Het resultaat is dat je nu een hele hoop strings met toetsaanslagen hebt: één per wachtwoord van een docent. Nou hebben de docenten tijdens het invoeren van hun wachtwoord niet alleen letters en cijfers getikt, maar ook de pijltjestoetsen gebruik om de cursor naar links of rechts te bewegen, en de backspace-toets om karakters te verwijderen. Gelukkig zijn ook al die toetsaanslagen geregistreerd, maar jíj moet nu de wachtwoorden reconstrueren uit deze invoer-strings, en omdat het er nogal veel zijn, besluit je hier een programmaatje voor te schrijven. Ondanks je slinkse motieven is dr. Klas zo goedhartig om je tegemoet te komen met een paar testcases op DOMjudge, zodat je je programma kunt testen.

2 Invoer en Uitvoer

De eerste regel van de invoer is een getal n: het aantal wachtwoorden (maximaal $1\,000\,000$). Daarna volgen n regels met op elk van die regels een string met de toetsaanslagen, van maximaal $1\,000\,000$ karakters. Docenten vinden namelijk lange wachtwoorden erg belangrijk, in het kader van veiligheid en adequate bescherming van persoonsgegevens van studenten, maar ze zijn soms ook best verstrooid, en maken dan wel eens wat foutjes, die ze met pijltjestoetsen en backspace herstellen. Elke string bestaat uit een rij karakters van vier mogelijke typen:

hoofd- of kleine letter, of cijfer: zo'n karakter is onderdeel van het wachtwoord, tenzij dit karakter later verwijderd wordt. We gaan uit van een standaard tekstinvoer: het nieuwe karakter wordt ingevoegd op de plek van de cursor, en alle karakters rechts van de cursor schuiven een plek op. De cursor staat hierna direct rechts van het zojuist ingevoerde karakter.

"-": het karakter 'min' representeert de backspace: als er een karakter direct links van de cursor staat wordt dit verwijderd.

"<": het karakter 'kleiner dan' representeert pijltje naar links: de cursor wordt 1 karakter naar links verplaatst, als dat mogelijk is, anders blijft de cursor staan.

">": het karakter 'groter dan' representeert pijltje naar rechts: de cursor wordt 1 karakter naar rechts verplaatst, als dat mogelijk is, anders blijft de cursor staan.

De uitvoer moet bestaan uit n regels: voor elke rij toetsaanslagen het gereconstrueerde wachtwoord. Het gereconstrueerde wachtwoord bestaat altijd uit minstens 1 karakter, en uit maximaal 1 000 000 karakters.

3 Voorbeeld

Bij de volgende invoer:

```
2
ThIsIsG3h31m
<<d<okl-i>akl<->-i<k>>
```

hoort de volgende uitvoer:

```
ThIsIsG3h31m
okidaki
```

Merk op dat in het tweede voorbeeld eerst twee keer op het pijltje naar links is gedrukt, maar aangezien de cursor nog aan het begin van de regel staat heeft dit geen effect. Loop vooral het tweede voorbeeld stap-voor-stap door, zodat je begrijpt hoe het wachtwoord wordt gereconstrueerd uit de inputstring. Dat is ook wat je programma zal moeten kunnen, en dan natuurlijk ook voor andere strings.

4 Implementatie eisen

Omdat de ingevoerde wachtwoorden nogal lang kunnen zijn, is het van belang dat alle operaties efficiënt geïmplementeerd worden. Hiervoor moet je gebruik maken van je **eigen implementatie van een Linked List als een verzameling van elementen, die met pointers aan elkaar verbonden zijn** (dus niet als array van elementen). Je moet dus een linked list maken die bestaat uit element objecten. Dat houdt in dat je ook een `class Element` moet maken, zodat je daar nieuwe objecten van kunt aanmaken als je ze nodig hebt. Zo'n object van de `Element` klasse, is wat je ziet als de drie lichtblauwe blokjes onder elkaar, op pagina 33 van de slides van hoorcollege 8. Je kunt hiervoor in C# een 'inner class' definiëren. Het voorbeeld in de C# programming guide over 'nested types' maakt het gebruik hiervan duidelijk, kijk ook naar de Design guidelines voor nested types. Zo'n element klasse schrijven als nested type is net en handig, maar niet noodzakelijk.

5 Hints, Tips

Het is belangrijk eerst weer goed over je ontwerp na te denken. Wat moet er gebeuren, en wat heb je (qua datastructuren) tot je beschikking? De plek van de cursor is natuurlijk erg belangrijk, die verandert door de "<" en ">", en bepaalt wat er op welke plek wordt toegevoegd (een letter of cijfer) of verwijderd (als backspace is gedrukt).

- Je moet de karakters die je tegenkomt stuk voor stuk verwerken, in de vorm van bepaalde aanpassingen aan een datastructuur waarin je uiteindelijk het wachtwoord opbouwt. Die datastructuur moet een Linked List zijn, opgebouwd als een verzameling gelinkte objecten. Zo'n object heeft een key (het karakter), plus één of meerdere pointers naar andere objecten.

- Denk goed na over de complexiteit van elke operatie. Het is bij deze opdracht gemakkelijk om een oplossing te schrijven die te traag is. De invoerstrings kunnen dermate lang zijn dat je je *niet meer dan een constante hoeveelheid werk per invoerkarakter* kunt veroorloven. Je kunt het je bijvoorbeeld niet veroorloven om een element met key k te bereiken door aan het begin te beginnen, en `next` pointers te volgen tot je bij een element met k als key bent aangekomen.

- In C# kun je de individuele karakters in een string `myString` bereiken met directe adressering.

  ```
  for (int ctr = 0; ctr < myString.Length; ctr++) {
      // doe iets met het karakter myString[ctr]
  }
  ```

  Omdat de String klasse van C# de `IEnumerable` interface implementeert, krijg je er ook een foreach construct bij. Bij *Modelleren en Programmeren* heb je kennis gemaakt met het `switch` statement, dat in deze context heel handig kan zijn. Je vindt er ook uitleg over in dit C# tutorial.

- Met de `Console.Write` methode in C# kun je één karakter naar de uitvoer kunt schrijven. Maar bij deze opgave kan het aantal karakters in de uitvoer zó groot worden dat dat *te traag* is, waardoor je in DOMjudge een TIMELIMIT krijgt. Maak in plaats daarvan met behulp van de C# `StringBuilder` klasse (tutorial) eerst een string `wachtwoord` die je, als je klaar bent, in één keer naar de uitvoer schrijft met `Console.WriteLine(wachtwoord)`.