Bevezetés a szabályos kifejezések használatába (2. rész)

Az előző lapszámban áttekintettük a szabályos kifejezések legalapvetőbb elemeit és láttunk néhány példát az alkalmazásukra is. Ebben a cikkben a kifinomultabb keresési és szövegfeldolgozási módszerekről, lehetőségekről esik szó.

Azt már tudjuk, hogy a szabályos kifejezések közönséges karakterekből (például „alma”), karakterhalmazokból (`[a-z]` vagy `[0-9]`), illetve néhány különleges jelentéssel bíró jelből (`.`, `,`, `$`, `^`) állnak össze.
Ezeken túl van az alkalmazható jelkészletnek egy olyan része is, amelynek az elemei a már ismert jelek jelentését módosítják, illetve finomítják.

Csillagok, csillagok

Talán a leggyakrabban használt jelentésmódosító jel a * (csillag) , vagyis a többszöröző karakter. Ez a közvetlenül előtte álló karakter tetszőleges számú előfordulását írja elő, a nullaszorosát is beleértve. Figyeljünk a megfogalmazásra, ugyanis ez az utóbbi kitétel nagyon fontos.
Az „a” mint szabályos kifejezés pontosan egyetlen „a” betűre illeszkedik. Tegyük fel, hogy olyan szövegrészeket akarunk azonosítani, amelyek tetszőlegesen sok „a” betűből állnak, de kizárólag ilyen betűkből. A kezdőknek ilyenkor szinte kivétel nélkül az `a*` tűnik a helyes megoldásnak, pedig ez óriási tévedés. Mivel ez a kifejezés nulla és tetszőlegesen sok „a”-ra is illeszkedik, bármi megfelel neki, hiszen bármely szövegrészre igaz, hogy vagy van benne „a” betű, vagy nincs. Ezzel a feltétellel szűrve tehát egy szöveget annak az összes sora megfelel a keresési feltételnek. Soha ne feledkezzünk meg tehát a * (csillag) nullaszoros illeszkedést is megengedő viselkedéséről, mert ha figyelmetlenek vagyunk, esetleg teljesen értelmetlenné tehetünk egy amúgy jól megtervezett szabályos kifejezést.
Esetünkben a helyes megoldás az `aa*` szabályos kifejezés használata, amelyben a * hatása csak a második „a”-ra terjed ki. Így a keresendő mintában legalább egy „a”-nak mindenképpen szerepelnie kell. Ugyanezt jelenti az `a\+` kifejezés is, amely egy újabb jelentésmódosító karaktert, a + (pluszjelet) tartalmazza. A + jelentése azonos a csillagéval, azzal az apró eltéréssel, hogy legalább egyszeres illeszkedést ír elő. Az előtte látható \ (perjel) karakterre azért van szükség, mert sem ez, sem az összes többi jelentésmódosító karakter nem része a szabályos kifejezések alapkészletének. Ha tehát nem betű szerinti értelemben akarjuk őket használni (esetünkben nem egy pluszjelet akarunk keresetni), akkor ezt a megfelelő programnak jeleznünk kell. Röviden tehát a \ kapcsolja be az illető karakter különleges jelentését.
A csillagot – hasonlóan minden egyéb jelentésmódosító jelhez – nemcsak egyetlen karakterre, hanem karakterhalmazra is alkalmazhatjuk. Ha például egy szövegből ki akarjuk válogatni az olyan sorokat, amelyekben egyessel kezdődő szám is van, akkor a

```
cat szöveg.txt | grep " 1[0-9]*"
```

parancsot használhatjuk. Figyeljük meg, hogy az egyes előtt egy szóköz is van. Ha ezt nem vettük volna bele a szabályos kifejezésbe, akkor az minden olyan számra is illeszkedne, amelyben bárhol legalább egy egyes előfordul. Ez pedig nem felel meg az előzetesen kitűzött feltételnek. Ha jobban belegondolunk, a megoldás még így sem tökéletes. Ha ugyanis egy egyessel kezdődő szám a sor elejére kerül, akkor az előtt nem lesz szóköz, vagyis hiába felel meg elvileg a feltételnek, a fenti szabályos kifejezés nem illeszkedik rá. Ismét csábító a gondolat, hogy a

```
cat szöveg.txt | grep " *1[0-9]*"
```

szabályos kifejezéssel próbálkozzunk, de ismét nincs szerencsénk. A `" *"` kifejezésrészlet az előbb elmondottaknak megfelelően „nulla vagy több szóközt” jelöl, ami első közelítésben rendben is volna, csakhogy a szabályos kifejezések logikája szerint ennek a feltételnek egy tetsző-

leges, egyetlen szóközt sem tartalmazó karakterlánc is megfelel. Ha tehát egy sorban szerepel például a "312" karakterlánc (amely nem egyessel kezdődő szám), akkor a grep ezt is megfelelőnek fogja találni. Szerinte ugyanis van benne egyes (1), amit egy van több számjegy követ ("[0-9]*"), előtte pedig – szerinte – nulla vagy több szóköz van (" *"), hiszen a 3 pontosan nulla szóközt tartalmaz. A helyes megoldást egy kicsit később tudjuk csak megvalósítani.

Csoportok

Szabályos kifejezésekből kerek zárójelek segítségével csoportokat is alkothatunk. A programok ezeket a csoportokat egy egységként kezelik, és a jelentésmódosító jelek is egyben vonatkoznak rájuk. Ha például olyan sorokat keresünk egy szövegben, amelyekben az „ab” betűpáros legalább egyszer előfordul, akkor a

```
cat szöveg.txt | grep "\(ab\)\+"
```

parancs segítségével szűrhetjük ki őket. Figyeljük meg, hogy az összes jelentésmódosító jel előtt használnunk kellett a \ (perjel) karaktert, jelezve, hogy nem kerek zárójeleket, illetve pluszjeleket keresünk a szövegben, hanem a logikai szerkezet részeként alkalmazzuk őket. A csoportosításnak köszönhetően a + jelentésmódosító itt az „ab” karakterpárra vonatkozik, és nem csupán a „b” betűre.

Logikai VAGY kapcsolat

Ha két szabályos kifejezést a | (cső) karakterrel választunk el egymástól, azt jelezzük, hogy közülük bármelyik illeszkedése megfelelő. Kicsit rövidebben fogalmazva ilyenkor logikai VAGY kapcsolatot létesítünk közöttük. Számos esetben jól jöhet ez a lehetőség. Ha például egy keresésnél egy szó bizonyos ragozott alakjai is megfelelőek, akkor nem kell külön-külön minden egyes alakra elvégezni a keresést, hanem a ragokat egyetlen önálló csoportként megjelölve logikai VAGY kapcsolattal köthetjük őket a szótőhöz. Ez így meglehetősen száraz volt, tehát nézzünk egy példát. Ha az „asztal” szó egyes ragozott alakjait akarjuk kikeresni, akkor a következőképpen járhatunk el:

```
cat szöveg.txt | grep "asztal\(on\|hoz\|nak\|ra\)"
```

Mivel a ragok egyetlen csoportot alkotnak (kerek zárójelek), a csoport elemei között pedig logikai VAGY kapcsolat áll fenn (\|), közülük bármelyik megfelel az illesztésnél, de egyszerre mindig csak egy.
Hasonló módszert alkalmazhatunk a korábban félbehagyott feladat megoldására is. Ott azt kell megoldanunk, hogy az egyes számjegy előtt vagy szóköz legyen, vagy az egész szerkezet a sor elején kezdődjön. Emlékezzünk rá, hogy a „sorkezdet” a szabályos kifejezésekben önálló elem (^), így akár egy csoport eleme is lehet:

```
cat szoveg.txt | grep "\(^\| \)1[0-9]*"
```

Igaz ugyan, hogy a sok \ miatt ez a kifejezés már meglehetősen cifrának tűnik, de pontosan úgy működik, ahogy azt szeretnénk. Az egyes előtt egyetlen csoport van, amelynek az egyik eleme egyetlen szóköz, a másik a sorkezdet, a kettőt pedig logikai VAGY köti össze. Így az illesztésnél bármelyikük megfelelő, de mindig csak az egyik.
Érdemes még megjegyezni, hogy a szabályos kifejezéseknél a szó hagyományos értelmében vett logikai ÉS kapcsolat nem létezhet, hiszen két logikailag különböző tartalmú szabályos kifejezés egyszerre nem illeszkedhet egy szövegrészre.

Az illeszkedések számának finomhangolása

Az ismétlődő részek számának pontos megadására a * (csillag) és a + (pluszjel) alkalmatlan, hiszen az első csak nulla vagy több, a második pedig egy vagy több illeszkedést ír elő, ami a pontosságtól meglehetősen távol esik. Ugyanakkor létezik a szabályos kifejezések jelkészletének olyan eleme is, amellyel egészen pontosan határozhatjuk meg, hogy egy karakter vagy csoport hányszor forduljon elő a keresett szövegrészben. Erre valók a kapcsos zárójelek, amelyekkel három különböző illeszkedési korlátot fogalmazhatunk meg.
Ha a nyitó és záró kapcsos zárójelek között csak egyetlen számot adunk meg, akkor pontosan ennyiszeres illeszkedést írunk elő. Az

```
"a\{5\}"
```

kifejezés tehát pontosan öt „a” betűre illeszkedik.
Ha a zárójelek között megadott számot egy vessző is követi, az azt jelenti, hogy legalább ennyiszeres illeszkedést várunk, vagyis az illeszkedések számát csak alulról kívánjuk korlátozni. Végül, ha két számot adunk meg vesszővel elválasztva, azzal alsó és felső korlátot egyaránt meghatározunk. Mindez egyébként azt jelenti, hogy „\{0,\}” jelentése megegyezik a * műveletjel (operator) jelentésével, míg „\{1,\}” ugyanúgy működik, mint a + – csak sokkal hosszabb.
Nézzünk megint néhány példát! A befejezetlen mondatok végét jelző három pontot a következő szabályos kifejezéssel azonosíthatjuk:

```
cat szöveg.txt | grep "\.\{3\}"
```

Ha négy vagy több pontot keresünk (ilyenek például az űrlapokon a kitöltendő mezők), azt a következőképpen tehetjük meg:

```
cat szöveg.txt | grep "\.\{4,\}"
```

A legalább három, de legfeljebb nyolcjegyű számokat a következő módszerrel válogathatjuk ki:

```
cat szöveg.txt | grep " [0-9]\{3,8\} "
```

(Figyeljük meg, hogy a kifejezés elején és végén is van egy-egy szóköz.)

Néhány példa a gyakorlatból

A sorozat előző részében is bemutattunk már néhány olyan megoldást, amelyekkel bizonyos gépelési hibák javíthatók. Ott még viszonylag keveset tudtunk a szabályos

kifejezésekről, most viszont csaknem a teljes fegyverkészlet a rendelkezésünkre áll. Lássunk először két új feladatot, majd visszatérhetünk a korábban említettekhez.
Gyakori gépelési hiba, hogy a szavak közé egyetlennél több szóköz kerül, vagy lemarad a mondat végéről a pont. Írjunk olyan egyszerű programokat, illetve feldolgozási sorokat, amelyek képesek ezeket a hibákat azonosítani, valamint ki is javítják őket.
A többszörös szóközök a

```
"  \+"
```

szabályos kifejezéssel azonosíthatók. Ezek egyszeresre cserélését a legegyszerűbben egy `sed` programmal oldhatjuk meg, valahogy így:

```
cat szöveg.txt | sed "s/  \+/ /g"
```

Hasonló, de egy kicsit körmönfontabb megoldást igényel a mondatvégi pontok pótlása. Ha a szövegben nincsenek tulajdonnevek, akkor a pont hiányát arról tudjuk felismerni, hogy egy kisbetűt egy szóköz, majd egy nagybetű követ. Ennek a helyzetnek a következő szabályos kifejezés felel meg:

```
"[a-z] [A-Z]"
```

Az igazi gondot az jelenti, hogy itt a cserét nem tudjuk olyan egyszerűen megfogalmazni, mint az előbb, hiszen a fenti kifejezésre illeszkedő szövegrészt nem egy pontra kell cserélni, hanem magukra a megtalált karakterekre és az őket elválasztó pontra. Szerencsére a sed-nek van memória-feladatköre. Ha a kereséshez használt kifejezésben valamit kerek zárójelek közé zárunk, akkor az így kialakított csoportok tartalmára azok sorszámával hivatkozhatunk.
A megoldás tehát a következőképpen fest:

```
cat szöveg.txt | sed "s/\([a-z]\) \([A-Z]\)/\1\. \2/g"
```

A cserélőutasítás második felében szereplő `\1` most azt a részt jelenti, amire `[a-z]`, a `\2` pedig azt, amelyikre `[A-Z]` illeszkedett. A többi `\` annak jelzésére szolgált, hogy az illető karaktert vezérlésre és nem mintaként kívánjuk használni.

Egz és ay

Cikksorozatunk első darabjában csak részben, illetve csak rendkívül körülményesen voltunk képesek elhárítani azt a nehézséget, amikor gépelés közben felcseréljük a „z” és „y” betűket. Azt már akkor is tudtuk, hogy a megfelelő eszköz a `sed`, amelynek keresési mintaként a megfelelő szabályos kifejezést megadva az önműködően végigolvassa a szöveget, és elvégzi a szükséges helyettesítéseket.
A gondot tulajdonképpen az okozta, hogy az elgépelés – egyszerűsége ellenére – egészen változatos helyeken bukkanhat fel. A közönséges előfordulási helyzeteken túl előfordulhat például, hogy egy mondat „Ay” karakterpárossal kezdődik, vagy „egz.”-re végződik. Ilyenkor sajnos nem igaz, hogy a hibásan gépelt rész előtt és után is szóköz van, vagyis nehézzé válik a cserélendő karakterlánc behatárolása.

Ugyanakkor most is megfogalmazhatunk olyan általános szabályokat, amelyeket aztán szabályos kifejezéssé alakíthatunk. Lássuk először az „egz” esetét. Ez kezdődhet kis vagy nagy „e” betűvel, amit szóköz vagy sorkezdet előz meg. Az „e” betűt biztosan egy „g” majd egy „z” követi. Végül vagy szóköz, vagy pont, vagy vessző következik.
Hasonló a helyzet az „ay” karakterpárossal is. Annyi csupán az eltérés, hogy itt a cserét éppen fordítva kell elvégezni, vagyis nem „z”-t kell „y”-ra cserélni, hanem fordítva. Ennyi elmélkedés után vegyünk egy nagy levegőt (a gyengébb idegzetűek inkább üljenek le), és lássuk a megoldást.

```
1: #!/bin/sh
2: # z-y csere: általános megoldás
3:
4: cat $1 | sed 's/\( \|^\)\(A\|a\)y\(
   ➥\|\.\|,\)/\1\2z\3/g
5: s/\( \|^\)\(Eg\|eg\)z\( \|\.\|,\)/\1\2y\3/g'
```

Ez első látásra meglepő. Másodikra mulatságos vagy rémisztő – a felhasználó lelkivilágától függően. Akármit is érez azonban most a nyájas olvasó, vizsgálja meg először a fenti förmedvényt abból a szempontból, hogy van-e benne olyan rész, amit nem ismer. Elvileg nem lehet ilyen. Jelenlegi ismereteink birtokában ezt már képesek vagyunk egyedül is összehozni.
Azért mielőtt a következő cikkhez lapoznánk, pusztán a biztonság kedvéért elemezgessük egy kissé művünket. Az valószínűleg azonnal nyilvánvaló, hogy itt két cseréről van szó: az első az „ay” a második az „egz” típusú elgépelésekkel foglalkozik. A keresési minta a következővel indul: `\( \|^\)`. Némi bogarászás után azonnal látszik, hogy ez két egyszerű szabályos kifejezés (egy szóköz és egy sorkezdet) logikai VAGY kapcsolattal összeláncolva, és kerek zárójelekkel csoporttá tömörítve. A helyettesítésnél az erre illeszkedő szövegrész lesz `\1`.
Aztán egy hasonló csoport következik: `\(A\|a\)`. Ez tehát egy kis vagy nagy „a” betű, amire a helyettesítési programban a `\2` szimbólummal hivatkozunk. Most jön az elgépelt karakter („y”), majd egy újabb logikai VAGY-okkal összetartott csoport: `\( \|\.\|,\)`.
A cserét előíró program a csoportokra illeszkedő tartalmakat változatlanul hagyja, csupán az elgépel „y”-t cseréli „z”-re (`"\1\2z\3"`). Az utasítás után szereplő „g” azt jelenti, hogy ezt a műveletet a minta valamennyi illeszkedési helyén el kell végezni. Az 5. sorban egy újabb, az előbbihez teljesen hasonló minta, illetve program következik, amely mindenütt az „egz” jellegű hibákat javítja ki.
Ennyi logikai szépség és matematikai báj után mit is kívánhatnék annak, aki átrágta magát ezen és az előző cikken? Haladjon bátran előre a Linux rögös útjain, és soha többé ne féljen a szabályos kifejezésektől!

Büki András (buki.andras@insilico.hu)
Körülbelül kilenc éve dolgozik Linux operációs rendszerrel. Állandó szerzőtársa Prof. Dr. H. V. Kuksinak, akivel a Duna vagy a Tisza partján szoktak az élet és a tudomány viselt dolgairól töprengeni.