Operációs rendszerek

2016/2017 tavaszi félév

II. ZH segédlet

Cirok Dávid

FONTOS: ezen segédlet elolvasása és megértése önmagában nem elég a ZH megírásához, nem helyettesíti az önálló gyakorlást!

Reguláris kifejezés

A reguláris kifejezéses feladatban a központi elem egy összetettebb regex megírása lesz, magát a feladatot Bash (`egrep`) vagy AWK scripttel is megoldhatjuk, nem ezen lesz a hangsúly. Tudni kell a regex építőelemeit használni. Ezekről nagyon sok leírást találhatunk az interneten, de a jegyzetek is részletesen tárgyalják.
Sokféleképpen tudunk egy reguláris kifejezést tesztelni, például terminálból általunk begépelt inputokkal:

```
cat | egrep '^[A-Z]$'
```

Ekkor beírunk egy sort, enter, és ha illeszkedik, visszakapjuk a sort outputként, ha nem, akkor csak a kurzor villog tovább. Ha adott inputhalmazra szeretnénk tesztelni, azokat beírhatjuk egy szöveges fájlba, és:

```
cat input.txt | egrep '^[A-Z]$'
```

Itt azokat a sorokat fogjuk visszakapni, amik illeszkednek. Ajánlott minél több**féle** “rossz” inputtal tesztelni, hogy elkerüljük a későbbi esetleges álpozitív illeszkedéseket.
A gyakorlás a hallgatóra van bízva, most egy egyszerűbb feladatot fogunk átnézni.
Példa: Írjunk egy reguláris kifejezést, ami a MAC-címekre illeszkedik (pl. `F8:23:E5:6B:89:56`). Lényegében hat, kettősponttal elválasztott mező van, 2-2 hexadecimális számmal (0-F). Egy hexadecimális számot halmazzal adunk meg: `[0-9A-F]` (ha meg akarjuk engedni a kisbetűket is, akkor `[0-9A-Fa-f]`), egy mezőt pedig ennek az ismétlésével: `[0-9A-F]{2}:` . A kettőspontot ahogy van odaírtam, mivel nem speciális karakter, ezért saját magára fog illeszteni (ne felejtsük el a speciális karaktereket escape-elni `\` karakterrel, ha pontosan rájuk szeretnénk illeszteni!). A kettőspontos mezőből kell 5 darab, majd simán egy mező. Ezt csoportosítással és ismétléssel adjuk

meg: `([0-9A-F]{2}:){5}[0-9A-F]{2}`. Végül (de lehetett volna a legelején is) beletesszük az input eleje és vége karaktereket, így a kész reguláris kifejezés:

```
^([0-9A-F]{2}:){5}[0-9A-F]{2}$
```

AWK scriptek

Itt az a legfontosabb, hogy értsük miként fut le egy AWK script (szabályok, minták, akciók), és hogyan működnek az asszociatív tömbök.
Példa 1: Írjunk egy AWK scriptet, ami megszámolja, hogy az input fájlban hány kisbetűvel, nagybetűvel és számmal kezdődő szó van!
Teljesen egyszerű AWK-os feladat, három változót használunk számlálónak, végigmegyünk a szavakon, kiírjuk mire jutottunk. A trükk (ahogy az órai wordcount feladatokban is) az, hogy nem egységesen formázott adathalmazról van szó, így potenciálisan minden rekordban eltérő számú mező lesz. Ezt rendszerint úgy kezeljük, hogy ciklussal járjuk be a rekordok tartalmát.
Azt, hogy egy szó kisbetűvel, nagybetűvel, vagy számmal kezdődik, többféleképpen tudjuk ellenőrizni, például el tudjuk kérni minden szóra az első karaktert a beépített `substr` függvénnyel (gyakorlásként oldjuk meg így!), de most reguláris kifejezést fogunk használni.

```
#! /usr/bin/awk -f

{
  for (i=1; i<=NF; i++) {
    if ($i ~ /^[a-z]/) {
      kisbetus++
    } else if ($i ~ /^[A-Z]/) {
      nagybetus++
    } else if ($i ~ /^[0-9]/) {
      szamos++
    }
  }
}

END {
  print kisbetus " kisbeture,"
  print nagybetus " nagybeture,"
  print szamos " szamra kezdodo szo van az inputban"
}
```

Első karakterre illesztünk minden szónál, összeszámolunk, aztán kiírunk. Ami érdekes itt, az a rekordok ciklussal való bejárása. A megoldás szépséghibája, hogy ha egy sincs az adott kategóriában, akkor üres stringként fog szerepelni az outputban, de ezt tudjuk orvosolni azzal, hogy a `BEGIN` blokkban inicializáljuk a változókat nullára.
Példa 2: Írjunk egy interaktív angol értelmező szótár scriptet a WordNet segítségével!
A WordNetről ezen a linken találunk információt. Ami minket érint most, az ez a fájl, ami tartalmazza

az 5000 leggyakrabban használt angol szót és (többek között) azoknak rövid magyarázatát, esetleg pár szinonímáját. Valami olyan scriptet kell összeraknunk, ami végigmegy ezeken, begyűjti az egyes szavakat és a hozzájuk tartozó információkat, majd végtelen ciklusban user inputot kér, és ha az input szóra van találat, kiírja a magyarázatot. Nézzük meg, hogy néz ki a szótár:

```
> head core-wordnet.txt
a [able%5:00:00:capable:00] [able] capable
a [abnormal%3:00:00::] [abnormal]
a [absent%3:00:00::] [absent]
a [absolute%3:00:00::] [absolute] perfect or complete
a [abstract%3:00:00::] [abstract] existing only in the mind
a [abundant%3:00:00::] [abundant] plentiful
a [academic%3:01:00::] [academic]
a [acceptable%3:00:00::] [acceptable]
a [accessible%3:00:00::] [accessible]
a [accurate%3:00:00::] [accurate]
```

A fájl számos egyéb információt tartalmaz, amit nem használunk fel, ill. egy szóhoz több rekord is tartozik, ha többféle jelentése van, de ezzel most nem foglalkozunk.
Az elválasztó karaktereket meghagyjuk az alapértelmezettnek, így minden sorban a harmadik mező lesz maga a szó (szögleges zárójelekben, de ezt később lekezeljük), a negyediktől a sor végéig pedig a szóhoz tartozó információ. Logikus lenne egy tömbben eltárolni mindent, ahol a kulcsok az egyes szavak (`$3`), az értékek pedig az utána következő mezők (`$4` -től `$NF` -ig).
Kezdjünk is el valami naív prototípust:

```
#! /usr/bin/awk -f

{
  szotar[$3] = $4
}

END {
  while (1) {
    getline szo < "-"
    szo = "[" szo "]"
    if (szo in szotar) {
      print szotar[szo]
    } else {
      print "not found"
    }
  }
}
```

Kigyűjtjük a szótárba a magyarázatból az első elemet (ez a `$4`), majd a végén beolvasunk egy szót, körétesszük a szögletes zárójelet (biztos lehetett volna szebben is megoldani, de jó lesz így), megnézzük tartalmazza-e a szótárunk, ha igen, kiírjuk a kulcshoz tartozó értéket, ha nem, kiírjuk, hogy bocsi nem. Próbáljuk ki:

```
> ./szotar.awk core-wordnet.txt
bad
spoiled,
ilyennincs
not found
```

Beírtam, hogy `bad` , kiírta a hozzá tartozó magyarázat első szavát, aztán beírtam egy olyat, ami biztosan nincs benne, hibaüzenet, első körben jó. (Ha meguntam, akkor `Ctrl+C` terminálja a futást.) A tömb feltöltésénél jó lenne, ha nem csak az első szó lenne tárolva az információból, hanem egészen a sor végéig minden ami van. Ezt egy ciklussal tudjuk megoldani a feldolgozás során, amiben a negyedik mezőtől az utolsóig járva konkatenációkkal (stringek egymás után írása) összegyűjtjük a teljes magyarázatot. Konkatenáláskor nekünk kell a szóközt betenni, mivel amikor szóköz mentén darabolásra kerül a rekord, azok nem lesznek benne a mezőkben.

```
# ...script eleje

{
  for (i=4; i<=NF; i++) {
    szotar[$3] = szotar[$3] " " $i
  }
}

# ...script vege
```

Annyi történt, hogy a szótár “szó”-adik eleméhez hozzáfűzünk egy szóközt, majd a soron következő mezőt a magyarázatból, és ezt addig, amíg van a magyarázatból. Próbáljuk ki újra:

```
> ./szotar.awk core-wordnet.txt
bad
 spoiled, spoilt, capable of harmingance, defective
great
 large in size, number or extent
ilyentovabbrasincs
not found
```

A tömbben így minden érték szóközzel kezdődik, mivel az első konkatenációkor (amikor még üres string az érték) is elétesszük a szóközt, de játsszuk azt, hogy ez nem bug, hanem feature. Kész a feladat.