Rule based graph modifier
Fogarasi Gergő
próbafeladat

Technológiák kiválasztása

- programozási nyelv: C++
- IDE: Code::Blocks
- fordító: GNU GCC Compiler (pontosabban g++)
- operációs rendszer: Windows 7
- gráf-kezelő: graphviz 2.50.0 (Windows-ra, Mac-re, Linux-ra, Solaris-ra, Unix-ra is elérhető),
  ennek segítségével a DOT Language is használható gráfok leírására,
  illetve gráfokról képeket / ábrákat lehet generálni
  lásd: https://graphviz.org/doc/info/lang.html
  https://graphviz.org/
  https://www.graphviz.org/pdf/libguide.pdf

Költözés és egyebek miatt a normális számítógépem (még) nincs nálam, csak egy kallódó Windows-os laptop, ezért ezen készítem el a feladatot. Igyekszem úgy dolgozni, hogy Linux-on is fordítható és futtatható legyen a program, a C++ nyelv szabványán kívül eső dolgokból (pl. lib-ek, SDK-k, API-k, operációsrendszer-függő dolgok) igyekszem minél kevesebbet vagy semennyit használni (vagy olyanokat használok, amik – elvileg – Linux-on is pontosan ugyanúgy elérhetőek és működnek).

Ha nálam lenne a számítógépem, akkor Linux környezetben dolgoznék, és GNUmake szintaxis szerint készítenék egy makefile-t, és GCC (illetve g++) fordítót használnék a C++ kód fordításához. Mivel nincs nálam, ezért Windows-on dolgozom, nem írok makefile-t (az ilyen dolgokat majd a Code::Blocks elvégzi helyettem), viszont ugyanúgy GCC (illetve g++) fordítót használok (illetve annak a Windows-on létező, minGW-s változatát, ami a Code::Blocks-ba be van építve).

A feladatmegoldásomról

A megoldásomat C++ nyelven készítettem el.

A bemenő gráf-fájlt egy állapotgéppel parse-olom be.

A program tárol egy DAG-ot, ami csúcspontokból (vertex-ekből) áll. Egy ilyen vertex-nek van típusa (pl. A) és ID-ja (pl. 1_A) (ez utóbbi 2 dolgot a Type típusú t nevű változó tárolja), ezen kívül minden vertex nyilvántartja a belőle kiinduló éleket (outputIDs).

A log.h és log.cpp fájlokban van egy egyszerű kis log-rendszer, ami a képernyőre és log-fájlba is logol.

A végrehajtandó módosításokat tartalmazó fájl általam kitalált szintaxisáról

A végrehajtandó módosításokat tartalmazó fájl szintaxisára példa:

```
ruleType1 replaceTriplet A B C with X (input: A, output: C)
# ruleType2 insertBetweenTheseTwo: D E this: Y
ruleType2 insertBetweenTheseTwo: D E this: Y
ruleType3 F -> G and H, G -> I and J, H -> J, I -> K and L, new Z (input: F) -> V (output: K) and W (output: L and J)
# Note: you can add your own customized rules (which should be the same type above but with modified letters), but only if they are the same (...)
```

A # karakterrel kezdődő sorok kommentek.
3-féle szabályt lehet megadni, tetszőleges sorrendben, és tetszőleges darabszámban. Ezekre van minta a fenti, szürkével kiemelt példában. A 3 szabály megadása kötött, csak az egyes betűket lehet kicserélni (például B), de azokat viszont bármire át lehet írni, akár egynél több karakter hosszúságúra is (viszont ezek a tokenek nem tartalmazhatnak például szóközt). A fenti mintában azonban minden ilyen jellegű token csupán 1 karakter hosszú (pl. B).

A bemenő és kimenő gráf-fájlok szintaxisáról

Valójában itt DAG-ok vannak. (Minden DAG egyben gráf is.)

A programba bemenő gráf (amit a program átalakít), és a programból kijövő gráf (ami az átalakítás eredménye) egyaránt egy-egy „.dot” kiterjesztésű, szöveges fájl. A dot egy nyelv, amivel le lehet írni gráfokat. Erről a nyelvről további információk: https://graphviz.org/doc/info/lang.html
Ennek a dot nyelvnek én egy általam kitalált szűkített, szigorított változatát használom (azért, hogy könnyebb legyen a parse-olás, mert amúgy ez a nyelv nagy rugalmasságot engedne meg, és minden létező esetet lekezelni egy helyes parse-olóval túl bonyolult lenne).

Példa a bemenő gráfot tartalmazó fájl szintaxisára:

```
digraph D {
  "1_A" [shape=diamond]
  "1_B" [shape=box]
  "1_C" [shape=circle]
  "1_F" [shape=circle]

  "1_A" -> "1_B" [style=dashed, color=grey]
  "1_A" -> "1_C" [color="black:invis:black"]
  "1_B" -> "1_C" [color="black:invis:black"]
  "1_D" -> "1_E" [color="black:invis:black"]
  "1_A" -> "1_D" [penwidth=5, arrowhead=curve]
}
```

- Az első szekcióban a csúcsok vannak felsorolva, a második szekcióban az élek, ezt a két szekciót pontosan egy darab üres sor választja el egymástól (az üres sorban lehetnek space-ek).
- Az első szekcióban nem kötelező az összes csúcsot felsorolni, néhány ki is hagyható, feltéve, hogy majd az élek leírásánál ezek szerepelni fognak.
- Mindkét szekcióban a csúcsok nevének megadásakor kötelező 1-1 darab idézőjelet használni (amúgy a dot nyelvben ez elhagyható, ha a névben nincs pl. szóköz vagy hasonló furcsaság, ezesetben az én szűkített dotnyelv-verziómban viszont nem hagyható el).
- Az első szekcióban a csúcsok neve után szögletes zárójelben formázási utasítások vannak. Ezeket a programom nem veszi figyelembe, de az ábra-generálás figyelembe veszi őket. A kimeneti fájlból ezek a formázások el lesznek hagyva az egyszerűség kedvéért (illetve mivel a gráf változik / változhat, bizonyos helyeken a program amúgy sem tudná kitalálni, mi legyen a formázás az adott csúcsnál / élnél, úgyhogy a kimeneti gráfnál egyáltalán nincsenek default-tól eltérő formázások).
- A második szekcióban az élek utáni szögleteszárójeles formázásokra ugyanez vonatkozik.
- Az első és második szekcióban a csúcsok nevében kötelező lennie egy underscore (_) karakternek. Az ettől jobbra lévő rész (például: A) az adott csúcs típusát határozza meg, az ettől balra lévő rész pedig egy sorozatszám. A kettő együtt (tehát például 1_A) egy egyedi azonosítót képez az adott csúcsra vonatkozóan, ami egyben tartalmazza annak típusát is (például A). Ennek az az értelme, hogy így egyszerre létezhet mondjuk három darab A típusú csúcs, mindezt úgy, hogy ezeket teljesen jól meg tudjuk egymástól különböztetni (például 1_A, 2_A, 3_A).
- A kimeneti gráf (ami szintén egy „.dot” kiterjesztésű fájl) szintaxisa pont ugyanilyen, azt a program generálja le. A különbség annyi, hogy abban nincsenek szögleteszárójeles formázási utasítások, azok teljesen el vannak hagyva.