MathProg program generátor

Zostavenie programu

Pre spustenie programu je nutné mať na počítači nainštalovaný Python 3. Skript sa spúšťa z príkazovej riadky rovnako, ako každý pythonovský skript, napríklad nasledovne:

```
python generator.py [args]
```

Ovládanie

Generátor vytvára lineárne programy a prípadne aj spúšťa vygenerované problémy v závislosti na poskytnutých parametroch.

```
Generator.py [problem] [path] [filename] [is_filelist] [run] [run_output]
```

Problem - problém, ktorý sa má transformovať na lin. Program

- ‘1’ pre Orientované grafy bez krátkých orientovaných cyklu
- ‘2’ pre Kocourkovské volby

Path - relatívna cesta priečinkami k súboru s dátami pre transformáciu na program

- Musí končiť s /

Filename - názov súboru, z ktorého sa majú načítať dáta problému

Is_filelist - slúži pre hromadné generovanie programov za využitia ‘reseni.txt’ zo zdrojových súborov. Dokáže porovnať výsledok so vzorom ak je flag **Run** povolený

- ‘-y’ - ak súbor je file list a majú sa generovať všetky problémy v súbore
- ‘-n’ - ak sa má vygenerovať problém priamo z jedného zdrojového súboru

Run - značí, či sa má spustiť riešenie problému po vygenerovaní programu

- ‘-r’ - ak sa má ihneď spustiť riešenie vygenerovaného programu
- ‘-d’ - ak sa má lineárny program vygenerovať a nič viac

Run_output - (optional) zložka, kam sa majú ukladať súbory s výstupom generovaným počas riešenia, nevyužíva sa ak je **Run** zakázaný

- Musí končiť na /
- Táto zložka už musí existovať

Pre spustenie generovania 1 programu teda stačí použiť napr.

```
generator.py 1 ukolPrakticky/uloha2_1/ vstup-s1.txt -n -d
```

Pre vygenerovanie všetkých programov, ich spustenie a kontrolu výsledku:

```
generator.py 1 ukolPrakticky/uloha2_1/ reseni.txt -y -c ukolPrakticky/uloha2_1/solutions/
```

Vygenerované programy sú umiestnené v rovnakej zložke, ako skript. Ak sa generuje len 1 program, tak sa nazýva *vygenerovane_lp.mod*. Ak sa generuje viacero súborov naraz, tak je výsledný program v súbore *vygenerovane_lp_<názov zdrojového súboru>.mod*.

Vygenerovaný program

Orientované grafy bez krátkych orientovaných cyklu:

```
# vytvori vrcholy a hrany, priradi kazdej hrane jej vahu a vytvori flag pre kazdu hranu, ci je
# dana hrana odoberana (true), alebo je ponechana (false)
set Nodes := 0..5;
set Edges := {(4,5),(5,0),(0,4),(5,3),(2,3),(4,2),(1,3),(0,2),(1,2),(4,1),(0,3),(2,5),(1,5)};
param Weights{(i,j) in Edges};
var Removed{(i,j) in Edges}, binary;

# ucelova funkcia - minimalizacia hmotnosti
minimize removedWeight: sum{(i,j) in Edges} Weights[i,j] * Removed[i,j];

# zabezpecit, ze nie je 3-cyklus (2-cykly v grafe nie su zo zadania), ak je cyklus, tak aspon
# jedna z hran musi byt odobrana
s.t. noThreeCycle{i in Nodes, j in Nodes, k in Nodes: i!=j and j!=k and k!=i}:
  (if ((i,j) in Edges and (j,k) in Edges and (k,i) in Edges) then (Removed[i,j] + Removed[j,k] +
Removed[k,i]) else 1) >= 1;

# to iste pre 4-cykly
s.t. noFourCycle{i in Nodes, j in Nodes, k in Nodes, l in Nodes: i!=j and j!=k and k!=l and l!=i}:
  (if ((i,j) in Edges and (j,k) in Edges and (k,l) in Edges and (l,i) in Edges) then (Removed[i,j] +
Removed[j,k] + Removed[k,l] + Removed[l,i]) else 1) >= 1;

solve;
# vypise pozadanovany vystup
printf "#OUTPUT: %d\n", sum{(i,j) in Edges} Weights[i,j] * Removed[i,j];
```

```
printf{(i,j) in Edges} (if Removed[i,j] then "Edge (%d,%d), Removed: %d\n" else ""), i, j,
Removed[i,j];
printf "#OUTPUT END\n";

# vkladanie dat pre vahy hran
data;
param Weights  [4,5] 37 [5,0] 41 [0,4] 32 [5,3] 27 [2,3] 27 [4,2] 20 [1,3] 17 [0,2] 26 [1,2] 33
[4,1] 28 [0,3] 37 [2,5] 12 [1,5] 27;
end;
```

Kocourkovské volby:

```
# vytvorenie vrcholov, hran a mnoziny premennych pre priradzovanie strany obyvatelovi
# kazdy vrchol grafu ma |v| premennych (teda n^2 premennych), kazda premenna sluzi na
# symbolizaciu, ci ma dany vrchol priradanu stranu x.
param N := 5;
set NodeIndexes := (0..N-1);
set Nodes := (0..N*N-1);
# obrateny graf - pre graf s vela klikami je jednoduchsie prechadzat inverznym grafom, ktory
# bude mat menej vrcholov
set Edges := {(0,4),(4,0)};
var Parties, >= 0, <= N-1, integer;
var nodeColor{i in 0..N*N-1}, binary;
# ucelova funkcia
minimize obj:Parties;
# jeden vrchol ma priradenu prave jednu stranu
s.t. oneColor{i in NodeIndexes}:
sum{j in NodeIndexes} nodeColor[i*N + j] = 1;
# (inverzny graf) - ak je hrana medzi 2 vrcholmi, obyvatelia musia byt v inych stranach
s.t. twoColorEdge{(i, j) in Edges, k in NodeIndexes}:
nodeColor[i*N + k] + nodeColor[j*N + k] <= 1;
# vzdy volime Z (pocet stran) minimalne - ak stacia 2 strany, tak ich cisla budu 0,1 a nie 2,3
s.t. minZ{i in NodeIndexes, j in NodeIndexes}:
nodeColor[i*N + j] * j <= Parties;
Solve;
```

```
# print output podla zadania
printf "#OUTPUT: %d\n", (Parties + 1);
printf{i in 0..N*N-1} (if nodeColor[i] = 1 then "v_%d: %d\n" else ""), i div N, (i mod N);
printf "#OUTPUT END\n";
End;
```

Pozn.: Generátor v rámci optimalizácie rozhoduje o tom, či je lepšie pracovať s inverzným grafom, alebo je lepšie pracovať so zadanými dátami (pre takmer úplné grafy je jednoduchšie pracovať s inverzným grafom - menej hrán). Ak sa ukáže, že hrán v programe je menej, ak je použitý graf zo zadania, tak sa pravidlo twoColorEdge zmení na:

```
s.t. twoColorEdge{i in NodeIndexes, j in NodeIndexes, k in NodeIndexes: i!=j}:

  (if (i,j) in Edges then 0 else (NodeColor[i*N + k] + NodeColor[j*N + k])) <= 1;
```

Problémy bez riešenia

1) Ak nemá problém riešenie, tak sa z grafu nedajú odobrať hrany, ktoré by prerušovali krátke cykly
2) Ak nemá problém riešenie, tak sa nedá rozdeliť obyvateľov do politických strán. Toto sa môže stať napríklad vtedy, keď je graf úplný a všetci môžu byť v jednej strane.