Projekt 3: Spy Union

autorzy: Olga Waszczuk, Małgorzata Łukomska

1. Opis rozwiązania:

Daną mamy firmę, w której znajdują się dwie organizacje zatrudniające niezależnie tych samych pracowników na różne stanowiska. Każdy pracownik ma przypisany numer od 0 do n. Pracownik może być zarówno szefem dla departamentu, w którym ma podwładnych, jaki równocześnie podlegać pod swojego przełożonego. Każdy departament ma podane zapotrzebowanie na liczbę pracowników – tzn. minimalna liczba podwładnych tego pracownika. Chcemy zakodować problem liniowy, którego rozwiązanie da nam odpowiedź na to, których pracowników zwolnić tak, żeby możliwie najbardziej zmniejszyć liczbę zatrudnionych osób w firmie.

W celu rozwiązania problemu optymalizacji struktury przedsiębiorstwa, przeniosłyśmy panującą w nim hierarchię na strukturę drzewa. Na początku na obie organizacje patrzymy jak na osobne drzewa. Poszczególny z węzłów reprezentuje jednego pracownika, odpowiadające mu poddrzewo składa się z wierzchołków reprezentujących każdego podwładnego. Najwyższy w strukturze szef jest korzeniem drzewa.

Następnie chcemy połączyć drzewa tak, żeby otrzymać graf sieci przepływów i z niego skonstruować problem liniowy. Rozpatrujemy dwa typy zmiennych. **Zmienne typu pierwszego:** Pierwszy typ zmiennych utożsamiamy z każdym wierzchołkiem grafu. Zmienne te przyjmują 1, jeśli pracownik reprezentowany przez wierzchołek zostanie zwolniony i 0, jeśli pracownik reprezentowany przez wierzchołek nie zostanie zwolniony. Zmienne te są oczywiście wspólne dla obu drzew. **Zmienne typu drugiego:** Drugi typ zmiennych $x_{i,j}$ odpowiada krawędzi w drzewie łączącej szefa o numerze i z jego podwładny o numerze j. Każda krawędź jest w tym ujęciu traktowana jako krawędź grafu, przez którą przekazywana jest informacja o liczbie osób zwalnianych w poddrzewie podwładnego (łącznie z tym podwładnym). Do każdego pracownika przypisałyśmy maksymalną liczbę osób, które mogą zostać zwolnione w hierarchii pod tym pracownikiem (tzn. różnica liczby podwładnych tego pracownika a liczby osób, które potrzebne są w departamencie). Dzięki temu na każdej krawędzi otrzymujemy ograniczenie, mówiące maksymalnie ilu podwładnych danego szefa może zostać zwolnionych.

Z powodów technicznych (solver problemów liniowych wymaga unikatowości nazw zmiennych) do nazwy każdej zmiennej opisującej krawędź, do-

dana jest liczba 0 lub 1, opisująca numer drzewa (numeracja w pythonie rozpoczyna się od indeksu 0, stąd pierwsze drzewo ma numer 0).

Problem liniowy będący rozwiązaniem problemu składa się z:

1) **Funkcji celu**: Szukamy maksimum po sumie zmiennych utożsamianych z każdym wierzchołkiem grafu, czyli maksimum po sumie zwolnionych osób.

2) Dwóch typów funkcji ograniczających:

a) **Ograniczenia wydajności** (Capacity constraints) – są to ograniczenia mówiące, że przepływ idący po krawędzi od podwładnego do jego szefa (czyli suma zwolnień podwładnych podwładnego oraz jego samego) nie może przekroczyć maksymalnej liczby osób do zwolnienia przypisanej do podwładnego.

b) **Ograniczenia równowagi**(Flow conservation equations) – ograniczenia te mówią, że suma przepływów idący po krawędziach od każdego podwładnego do jego szefa (czyli suma zwolnień podwładnych) dodana do informacji, czy szef jest zwolniony, czy nie musi być równa informacji idącej od tego szefa do jego przełożonego. W przypadku gdy szef jest korzeniem (nie ma przełożonego) suma ta nie może przekroczyć maksimum zwolnień, dla tego szefa.

3) **Ograniczenia zmiennych** (Bounds): Jak wspomniane powyżej, zmienne pierwszego typu ($x_{i,j}$), kodujące czy dany pracownik zostanie zwolniony czy nie, to zmienne przyjmujące wartości {0, 1}. Drugi typ zmiennych jest nieujemny i przyjmuje dowolną liczbę naturalną n, nieprzekraczającą maksimum osób możliwych do zwolnienia w przypadku podwładnego. Oznacza to, że np. zmienna $x_{i,j}\_0$ przyjmuje liczbę naturalną między 0 a n={maksimum podwładnych pracownika j, których można zwolnić w zerowej[1] organizacji}.

4) **Zmiennych**: Wypisanie wszystkich zmiennych.

2. Opis programu:

W tej części opiszemy działanie programu generującego problem liniowy.

Dane wejściowe do programu stanowi czterokolumnowa tabela, w której dla itego wiersza:

- w pierwszej kolumnie znajduje się indeks szefa i-tego pracownika w organizacji o numerze 0;

[1]de facto w pierwszej, ale python numeruje od 0

- w drugiej kolumnie znajduje się indeks szefa i-tego pracownika w organizacji o numerze 1;
- w trzeciej kolumnie znajduje się minimalna liczba podwładnych i-tego pracownika (włączając jego samego), w organizacji o numerze 0;
- w czwartej kolumnie znajduje się minimalna liczba podwładnych i-tego pracownika (włączając jego samego), w organizacji o numerze 1.

Na początku program przepisuje te dane do macierzy , zachowując powyższą konwencję. Jest to zabieg techniczny, ułatwiający nam dalszą pracę z danymi.

Poniżej znajduje się opis właściwego programu:

1. Funkcja **count**(tree, boss_i, matrix) jako dane wejściowe bierze numer organizacji, indeks pracownika (liczbę przypisaną do pracownika) i macierz przechowującą strukturę organizacji. Dla danej liczby będącej indeksem pracownika funkcja przeszukuje kolumnę macierzy odpowiadającą wczytanej organizacji w poszukiwaniu tych indeksów, dla których w kolumnie opisującej numer szefa, znajduje się wczytany indeks. Zlicza te indeksy i zwraca tym samym liczbę podwładnych (ponieważ zliczanie zaczyna sie od 1, tym samym mamy oddany fakt, że do liczby podwładnych wliczamy też samego pracownika).

2. Funkcja **get_employees_under_boss**(boss_id, tree, matrix) działa analogicznie do funkcji **count** lecz tym razem zamiast liczyć podwładnych wpisuje ich indeksy kolejno do listy.

3. Funkcja **get_employees_lists**(tree, matrix) dla macierzy opisującej strukturę firmy i wybranej organizacji zwraca słownik opisujący strukturę wybranej organizacji, tzn. do każdego pracownika przypisuje następujące własności: "Self" - jego własny indeks, "Count" - liczba jego podwładnych, "ToFire" - maksymalna liczba jego podwładnych (łącznie z nim), która może zostać zwolniona, "Tree" - organizację dla której opisywana jest hierarchia, "Boss" - indeks szefa tego pracownika, "Employees" - listę podwładnych tego pracownika (bez niego).

   Następujące funkcje biorą jako dane wejściowe słownik opisujący strukturę organizacji i na jego podstawie wypisują problem w postaci programu liniowego. Kodowanie wygląda w następujący sposób: zmienne

pierwszego rodzaju (tzn zmiennych binarnych kodujących czy zwolnimy danego pracownika) nazywają się "eb_(indeks pracownika)e_(indeks pracownika)", natomiast krawędzie łączące podwładnego i szefa nazywają się "eb_(indeks szefa)e_(indeks podwładnego)_(numer organizacji w której występuje taka hierarchia)".

4. **getSelfObjectiveFunction** - funkcja zwracająca tekst, który koduje funkcję celu. Mianowicie wypisujemy sumę wszystkich zmiennych pierwszego typu (tzn zmiennych binarnych kodujących czy zwolnimy danego pracownika).

5. **getCapacityConstraint** - funkcja zwraca jako tekst funkcje ograniczające typu a (Ograniczenia wydajności).

6. **getConservationConstraint** - funkcja zwraca jako tekst funkcje ograniczające typu b (Ograniczenia równowagi).

7. **getSelfBounds** - funkcja zwracająca tekst będący ograniczeniami dla zmiennych pierwszego typu.

8. **getBounds** - funkcja zwracająca tekst będący ograniczeniami dla zmiennychdrugiego typu.

9. **getSelfGenerals** - funkcja wypisująca zmienne pierwszego typu.

10. **getGenerals** - funkcja wypisująca zmienne drugiego typu.

Na końcu następuje przypisane hierarchii do odpowiedniej organizacji organizacji (tree0 to pierwsza organizacja, tree 1 - druga organizacja) i wygenerowanie programu liniowego.

Tak wygenerowany program rozwiązujemy przez solver online:
http://hgourvest.github.io/glpk.js/.
Wygenerowane problemy jak i ich rozwiązania znaleźć można w folderze "testy".