

AiR rok II WEAlIB Badania Operacyjne I

Temat: Zagadnienie przydziału algorytm (metoda) węgierska.

Grupa: 1, środa godz. 14:45

Data ćwiczenia: 26.04.2023r.

Zespół i zakres obowiązków:

- Adam Barnaś (Redukcja macierzy, schemat ogólny algorytmu)
- Karolina Chochrek (Algorytm wyznaczania zer niezależnych)
- Gabriela Bergiel (Algorytm wykreślania zer macierzy minimalną liczbą linii, przygotowanie sprawozdania)

Zadanie 1:

- Implementacja procedur:
 - Redukcja macierzy, schemat ogólny algorytmu

```
14 def algorytm(macierz):
15     size = len(macierz)
16     zredukowana, koszt = reduction(macierz)
17     niezalezne, wszystkie, wiersze, kolumny = zera(zredukowana)
18     if len(niezalezne) == size:
19         return niezalezne, koszt
20     else:
21         while len(niezalezne) < size:
22             w_wiersze, w_kolumny = wykreslanie(zredukowana, wszystkie, wiersze, kolumny)
23             zredukowana, sigma = proba(zredukowana, w_wiersze, w_kolumny)
24             koszt += sigma
25             niezalezne, wszystkie, wiersze, kolumny = zera(zredukowana)
26         return niezalezne, koszt
27
28
29 rozwiazanie, koszt = algorytm(macierz)
30 print(rozwiazanie)
31 print(koszt)
```

```
1 from typing import List
2 # from main import macierz
3
4
5 # funkcja jako parametr przyjmuje macierz, którą będzie redukować
6 def reduction(matrix: List[List[int]]):
7     row_subtraction = [[0 for _ in range(len(matrix[0]))] for _ in range(len(matrix))]
8     # przygotowanie miejsca na macierz po redukcji wierszy
9     total_reduction = 0 # dolne ograniczenie wartości funkcji celu
10    for row in range(len(matrix)): # przejście po wierszach
11        min_val = min(matrix[row]) # najmniejszy element w wierszu
12        for col in range(len(matrix)): # przejście po kolejnych elementach wiersza
13            row_subtraction[row][col] = matrix[row][col] - min_val
14            # nowa macierz po redukcji wierszy
15            total_reduction += min_val # dodanie minimalnego elementu do sumarycznej redukcji
16
17    col_subtraction = [[0 for _ in range(len(matrix[0]))] for _ in range(len(matrix))]
18    # przygotowanie miejsca na macierz po redukcji kolumn
19
20    for col in range(len(matrix)): # przejście po kolumnach
21        col_vals = [row_subtraction[row][col] for row in range(len(matrix))]
22        # uzyskanie kolumny
23        min_val = min(col_vals) # najmniejszy element w kolumnie
24        for row in range(len(matrix)): # przejście po kolejnych elementach kolumny
25            col_subtraction[row][col] = row_subtraction[row][col] - min_val
26            # nowa macierz po redukcji kolumn
27            total_reduction += min_val # dodanie minimalnego elementu do sumarycznej redukcji
28
29    return col_subtraction, total_reduction
30 # zwraca macierz po obu redukcjach i dolne ograniczenie wartości funkcji celu
```

○ Algorytm wyznaczania zer niezależnych

```

16 def zera(macierz):
17     rozmiar = len(macierz[0])
18     row = [0 for _ in range(rozmiar)] # lista zliczająca zera w wierszach
19     col = [0 for _ in range(rozmiar)] # lista zliczająca zera w kolumnach
20     wszystkie = set() # zbiór współrzędnych zer
21     for i in range(rozmiar): # pętla znajdująca zera i uzupełniająca listy
22         for j in range(rozmiar):
23             if macierz[i][j] == 0:
24                 row[i] += 1
25                 col[j] += 1
26                 wszystkie.add((i, j))
27     wiersze = row.copy() # kopiowanie danych z list aby ich nie naruszyć dalszym działaniem algorytmu
28     kolumny = col.copy()
29     idx = (-1, -1)
30     wybrane = [set(), set()] # zbiory zawierające numery wierszy i kolumn zer wybranych jako niezależne
31     niezalezne = set() # zbiór współrzędnych zer niezależnych
32     for _ in range(rozmiar): # pętla wywoływana n razy dla znalezienia n zer niezależnych
33         minimum = rozmiar * 2
34         changed = False
35         for i in range(rozmiar): # pętla znajdująca zero z minimalną sumą liczby innych zer w tej samej kolumnie lub wierszu
36             for j in range(rozmiar):
37                 if macierz[i][j] == 0 and row[i] + col[j] < minimum and i not in wybrane[0] and j not in wybrane[1]:
38                     minimum = row[i] + col[j]
39                     idx = (i, j)
40                     changed = True
41             if not changed: # ewentualne przerwanie pętli w przypadku niezalezienia kolejnego zera
42                 break
43             wybrane[0].add(idx[0])
44             wybrane[1].add(idx[1])
45             for j in range(rozmiar): # aktualizacja informacji o liczbie zer w kolumnach
46                 if macierz[idx[0]][j] == 0:
47                     col[j] -= 1
48             for i in range(rozmiar): # aktualizacja informacji o liczbie zer w wierszach
49                 if macierz[i][idx[1]] == 0:
50                     row[i] -= 1
51             row[idx[0]] = 0
52             col[idx[1]] = 0
53             niezalezne.add(idx) # dodanie zera do zbioru zer niezależnych
54             # zalezne = wszystkie - niezalezne
55     return niezalezne, wszystkie, wiersze, kolumny

```

○ Algorytm wykreślania zer macierzy minimalną liczbą linii

```

1 def wykreslanie(G, zera, zeros_in_rows, zeros_in_cols):
2     G_size = len(G)
3     idx_of_max = lambda list: list.index(max(list))
4
5     crossed_rows = [] #lista indeksów skreślonych kolumn
6     crossed_cols = [] #lista indeksów skreślonych wierszy
7     zeros_list = [zero for zero in zera] #kopia listy zer
8
9     #póki w wierszach i kolumnach są nieskreślone zera
10    while(not (all(v == 0 for v in zeros_in_cols) & all(v == 0 for v in zeros_in_rows))):
11        row_with_max_zeros = idx_of_max(zeros_in_rows) #szukam wiersza z maks. liczbą kolumn
12        crossed_rows.append(row_with_max_zeros) #dodanie wiersza do listy skreślonych
13        zeros_in_rows[row_with_max_zeros] = 0 #"wykreślenie" tego wiersza, by nie brać go pod uwagę potem
14
15        #odjęcie od kolumn zer ze skreślonego wiersza
16        for zero in zeros_list:
17            if(zero[0] == row_with_max_zeros and zeros_in_cols[zero[1]] > 0):
18                zeros_in_cols[zero[1]] -= 1
19
20        #jeśli w kolumnach są jeszcze nieskreślone zera:
21        if(not (all(v == 0 for v in zeros_in_cols))):
22            col_with_max_zeros = idx_of_max(zeros_in_cols) #wyszukaj kolumnę z maks. ilością wierszy
23            crossed_cols.append(col_with_max_zeros) #dodaj ją do skreślonych
24            zeros_in_cols[col_with_max_zeros] = 0 #nie bierz jej pod uwagę w dalszych wykreśleniach
25
26        #odjęcie od wierszy zer ze skreślonej kolumny:
27        for zero in zeros_list:
28            if(zero[1] == col_with_max_zeros and zeros_in_rows[zero[0]] > 0):
29                zeros_in_rows[zero[0]] -= 1
30    #zwróć listę skreślonych zer i kolumn
31    return (crossed_rows, (crossed_cols))

```

• Zadanie obliczeniowe

```

6 macierz = [[1, 5, 1, 8, 1, 3],
7            [4, 3, 7, 3, 6, 3],
8            [2, 4, 7, 8, 4, 5],
9            [4, 3, 6, 7, 6, 5],
10           [4, 7, 5, 6, 6, 9],
11           [7, 2, 4, 3, 5, 9]]
12

```

Zadanie 2:

Wykonanie obliczeń do zadania:

- **macierz początkowa:**

```
▼ macierz: [[1, 5, 1, 8, 1, ...]
> special variables
> function variables
> 0: [1, 5, 1, 8, 1, 3]
> 1: [4, 3, 7, 3, 6, 3]
> 2: [2, 4, 7, 8, 4, 5]
> 3: [4, 3, 6, 7, 6, 5]
> 4: [4, 7, 5, 6, 6, 9]
> 5: [7, 2, 4, 3, 5, 9]
```

- **macierze pośrednie**

Macierz po redukcji i znalezione zera niezależne:

```
> function variables
▼ 0: [[0, 4, 0, 7, 0, 2], [
> special variables
> function variables
> 0: [0, 4, 0, 7, 0, 2]
> 1: [1, 0, 4, 0, 3, 0]
> 2: [0, 2, 5, 6, 2, 3]
> 3: [1, 0, 3, 4, 3, 2]
> 4: [0, 3, 1, 2, 2, 5]
> 5: [5, 0, 2, 1, 3, 7]
```

```
> niezalezne: {(3, 1), (1, 3), (0, 2), (2, 0)}
```

Wykreślanie zer:

```
✓ G: [[0, 4, 0, 7, 0, 2], [1...
> special variables
> function variables
> 0: [0, 4, 0, 7, 0, 2]
> 1: [1, 0, 4, 0, 3, 0]
> 2: [0, 2, 5, 6, 2, 3]
> 3: [1, 0, 3, 4, 3, 2]
> 4: [0, 3, 1, 2, 2, 5]
> 5: [5, 0, 2, 1, 3, 7]
  len(): 6
  G_size: 6
> crossed_cols: []
> crossed_rows: []
```

```
✓ G: [[0, 4, 0, 7, 0, 2], [1...
> special variables
> function variables
> 0: [0, 4, 0, 7, 0, 2]
> 1: [1, 0, 4, 0, 3, 0]
> 2: [0, 2, 5, 6, 2, 3]
> 3: [1, 0, 3, 4, 3, 2]
> 4: [0, 3, 1, 2, 2, 5]
> 5: [5, 0, 2, 1, 3, 7]
  len(): 6
  G_size: 6
  col_with_max_zeros: 1
> crossed_cols: [1]
> crossed_rows: [0]
```

```
✓ G: [[0, 4, 0, 7, 0, 2], [1...
> special variables
> function variables
> 0: [0, 4, 0, 7, 0, 2]
> 1: [1, 0, 4, 0, 3, 0]
> 2: [0, 2, 5, 6, 2, 3]
> 3: [1, 0, 3, 4, 3, 2]
> 4: [0, 3, 1, 2, 2, 5]
> 5: [5, 0, 2, 1, 3, 7]
  len(): 6
  G_size: 6
  col_with_max_zeros: 1
> crossed_cols: [1]
> crossed_rows: [0, 1]
```

```
✓ G: [[0, 4, 0, 7, 0, 2], [1...
> special variables
> function variables
> 0: [0, 4, 0, 7, 0, 2]
> 1: [1, 0, 4, 0, 3, 0]
> 2: [0, 2, 5, 6, 2, 3]
> 3: [1, 0, 3, 4, 3, 2]
> 4: [0, 3, 1, 2, 2, 5]
> 5: [5, 0, 2, 1, 3, 7]
  len(): 6
  G_size: 6
  col_with_max_zeros: 0
> crossed_cols: [1, 0]
> crossed_rows: [0, 1]
```

Próba zredukowania, czyli od elementów nieprzykrytych liniami odejmowany jest najmniejszy element a do elementów przykrytych dwoma liniami dodawany jest najmniejszy element (etap pośredni i końcowy):

```
> function variables
> 0: [1, 5, 0, 7, 0, 2]
> 1: [2, 1, 4, 0, 3, 0]
> 2: [0, 2, 5, 6, 2, 3]
> 3: [1, 0, 3, 4, 3, 2]
> 4: [0, 3, 1, 2, 2, 5]
> 5: [5, 0, 2, 1, 3, 7]
  len(): 6
  najmniejszy: 1
```

```
> function variables
> 0: [1, 5, 0, 7, 0, 2]
> 1: [2, 1, 4, 0, 3, 0]
> 2: [0, 2, 4, 5, 1, 2]
> 3: [1, 0, 2, 3, 2, 1]
> 4: [0, 3, 0, 1, 1, 4]
> 5: [5, 0, 1, 0, 2, 6]
  len(): 6
  najmniejszy: 1
```

Zwiększenie kosztu:

koszt: 17

Powiększenie zbioru zer niezależnych (liczba zer niezależnych jest równa liczbie wierszy i kolumn) macierzy:

```
> function variables
> 0: [1, 5, 0, 7, 0, 2]
> 1: [2, 1, 4, 0, 3, 0]
> 2: [0, 2, 4, 5, 1, 2]
> 3: [1, 0, 2, 3, 2, 1]
> 4: [0, 3, 0, 1, 1, 4]
> 5: [5, 0, 1, 0, 2, 6]
len(): 6
minimum: 2
v niezależne: {(0, 4), (1, 5), (3, 1), (2, 0), (4, 2), (5, 3)}
```

- macierz końcowa i optymalny koszt przydziału

```
[1, 5, 0, 7, 0, 2]
[2, 1, 4, 0, 3, 0]
[0, 2, 4, 5, 1, 2]
[1, 0, 2, 3, 2, 1]
[0, 3, 0, 1, 1, 4]
[5, 0, 1, 0, 2, 6]
rozwiązanie: {(0, 4), (1, 5), (3, 1), (2, 0), (4, 2), (5, 3)}
koszt: 17
```

Zadanie 3

- **Czy wynik redukcji jest zależny od kolejności (wiersze-kolumny/ kolumny-wiersze) – uzyskamy zawsze tę samą macierz / sumaryczną wielkość redukcji?**
Niezależnie od kolejności redukcji wierszy i kolumn, otrzymamy tę samą macierz redukcji i sumaryczną wielkość redukcji. Wynik końcowy zależy tylko od wartości tych minimalnych wartości w każdym wierszu i kolumnie, a nie od kolejności ich redukcji.
- **Jak jest możliwa minimalna / maksymalna liczba zer niezależnych w macierzy zredukowanej?**
Maksymalna liczba zer niezależnych jest równa wymiarowi macierzy, minimalna to dwa zera niezależne. Z minimalnej liczby zer niezależnych wynika też maksymalna ilość kroków w jakim zostanie znalezione rozwiązanie problemu czyli $n-2$ kroków.
- **Czy wykreślanie zer macierzy jest prawidłowa (zawsze) jeśli będziemy wykreślać kolejno linie (wiersz/kolumna) z największą liczbą nieskreślonych zer?**
W większości przypadków metoda wykreślania zer jest dowolna (o ile prowadzi do minimalnej ilości linii). Należy jednak ją stosować konsekwentnie do końca.

- **Jak się ma minimalna liczba linii (wykreślająca zera) i liczba (maksymalna) zer niezależnych?**

Minimalna liczba linii wykreślających musi być większa lub równa liczbie zer niezależnych. Gdy znajdziemy maksymalną liczbę zer niezależnych to liczba linii wykreślających będzie równa liczbie kolumn/wierszy macierzy.

- **Czy procedura zwiększająca liczbę zer niezależnych zawsze jest skuteczna / o ile może zmienić się ich liczba?**

W danym wierszu/kolumnie może być tylko 1 zero niezależne, więc ich liczba może być zwiększana póki ich ilość nie będzie równa wymiarowi macierzy.