Aufgabe 5: Wichteln

Team-ID: 00537

Team-Name: LuC++

Bearbeiter dieser Aufgabe:
Lucas Schwebler

14. November 2020

Inhaltsverzeichnis

1 Lösungsidee

Dieses Problem besitzt große Ähnlichkeit zu dem *bipartiten Matching*, für das eine klassische Problemstellung wäre: „Jeder Schüler aus Pirmins Klasse wünscht sich beliebig viele Geschenke. Ordne jedem Schüler genau ein Geschenk zu, sodass möglichst viele Schüler ein Geschenk ihrer Wunschliste erhalten haben."

Der einzige Unterschied ist, dass die Wünsche gewichtet sind und die Anzahl an Wünschen begrenzt ist. Es lässt sich jedoch in ein bipartites Matching umwandeln:

Zunächst erfüllen wir möglichst viele 1. Wünsche danach möglichst viele 2. Wünsche, ... Wobei alle 1. Wünsche den gleichen Wert haben, was das Problem der Gewichte löst.

2 Umsetzung

2.1 Das klassische bipartite Matching

Um dieses bipartite Matching zu finden, wird ein klassischer Ansatz leicht abgeändert. Dieser wurde bereits in meiner Einsendung zu Aufgabe 1 beschrieben, ist hier aber noch einmal (mit Geschenk Kontext) aufgeführt, um die Modifikation leichter verständlich zu machen:

```
def findMatch(schüler):
    for geschenk in wünsche[schüler]:
        if not besucht[geschenk]:
            besucht[geschenk] = True
            if (not zuordnung[geschenk]) or findMatch(zuordnung[geschenk]):
                zuordnung[geschenk] = schüler
                return True
```

```
8       return False
```

Wir versuchen nacheinander allen Schülern ein Geschenk zuzuordnen. Dazu testen wir alle möglichen Geschenke (Nachbarknoten im bipartiten Graphen). Ist das Geschenk frei, also noch nicht zugeordnet, haben wir eine passende Zuordnung gefunden. Sonst müssen wir den Schüler, der aktuell für dieses Geschenk vorgesehen ist, neu zuordnen. Wir können die Funktion also Rekursiv mit dem aktuell zuzuordnenden Schüler aufrufen. Damit wir dies für keine Schüler mehrfach tun (und vielleicht eine Endlosschleife verursachen), wird gespeichert, ob wir diesen Knoten (diesen Schüler) bereits besucht haben.

2.2 Finden eines augmentierenden Pfades

Wenn wir nun nacheinander die 1., dann die 2. und schließlich die 3. Wünsche erfüllen, ist dabei aber das Vorgehen zum finden eines augmentierenden Pfades, wie es gerade beschrieben wurde, anzupassen. Denn wenn Schüler s Wunsch Nummer n erfüllt werden soll und dieses Geschenk bereits zugeordnet ist, besteht die Möglichkeit, dass wir s den m-ten Wunsch ($m < n$) erfüllen können und stattdessen dem Schüler t, dem das Geschenk von Wunsch m zugeordnet wurde, den n-ten Wunsch von t erfüllen können.

Dies ist ähnlich zu dem normalen Vorgehen, da wir letztendlich versuchen einen „Konkurrenten“ um ein Geschenk neu zuzuordnen. Dabei muss man hier aber aufpassen, dass die Gesamtwertung dadurch nicht schlechter wird! Wir könnten aktuell nämlich Schüler t, dessen 1. Wunsch gerade erfüllt wird, seinen 3. Wunsch erfüllen, nur um s den 2. Wunsch erfüllen zu können. Dies liegt daran, dass der bisherige Ansatz nur betrachtet, ob ein anderer Schüler bereits dieses Geschenk erhält. Es muss also auch noch gewährleistet sein, dass die Wünsche von s und t für das Geschenk, um welches sie „konkurrieren“ gleichwertig sind, damit das Gesamtergebnis nicht schlechter wird.

Der Fall, dass der Wunsch von s für dieses Geschenk einen höheren Wert hat (das Erfüllen dieses Wunsches ist wichtiger), kann nicht eintreten, da erst so viele 1. und dann so viele 2. und zuletzt so viele 3. Wünsche wie möglich erfüllt werden. Der Wunsch wäre sonst nämlich bereits zuvor erfüllt worden. Da der Algorithmus in jeder der 3 Iterationen alle Möglichkeiten für eine neue Zuordnung testet[1], die zu einem besseren Ergebnis führen könnten, ist das Ergebnis optimal.

2.3 Ungenutzte Geschenke

Es kann vorkommen, dass nicht alle Geschenke, die zugeordnet werden auch tatsächlich auf der Wunschliste stehen und somit Geschenke übrig bleiben. Diese müssen dann den Schülern ohne Geschenk zugeordnet werden. Dies lässt sich z.B. mit einer queue oder einem stack realisieren.

3 Laufzeit

Das bipartite Matching benötigt $O(|V| \cdot |E|)$ Zeit, da der Graph aus $|V|$ Knoten besteht und $|V|$ damit das maximale Matching ist und jeder augmentierende Pfad das Matching um 1 vergrößert, wobei das Finden eines solchen Pfades $O(|E|)$ Zeit in Anspruch nimmt. Da $|E| = 3 \cdot |V|$ und 3 mal ein Matching gefunden werden muss, ist die Laufzeit insgesamt $O(3 \cdot |V| \cdot 3|V|) = O(|V|^2)$.

Die Laufzeit wächst also quadratisch zur Anzahl der Schüler, wodurch selbst Beispiele mit 1000 Schülern in Bruchteilen einer Sekunde berechnet werden können.

4 Beispiele

Die Ausgabe enthält jeweils maximal erfüllbare Anzahl an Wünschen, sowie die gewünschte Zuordnung in beide Richtungen:

```
wichteln1.txt
Maximale erfüllbare Wünsche:
1.  6
2.  0
3.  2

Zuordnung:  (G = Geschenk, P = Person, W = Wunsch)
G   P   W   |   P   G   W
```

[1] Indem versucht wird einen augmentierenden Pfad zu finden.

```
1    3    3    |    1    6    3
2    2    1    |    2    2    1
3    4    1    |    3    1    3
4    6    1    |    4    3    1
5    5    -    |    5    5    -
6    1    3    |    6    4    1
7    7    1    |    7    7    1
8    10   -    |    8    10   1
9    9    1    |    9    9    1
10   8    1    |    10   8    -
```

```
wichteln2.txt
Maximale erfüllbare Wünsche:
1.  3
2.  0
3.  0

Zuordnung:  (G = Geschenk, P = Person, W = Wunsch)
G    P    W    |    P    G    W
1    4    -    |    1    4    1
2    5    -    |    2    5    1
3    6    -    |    3    6    1
4    1    1    |    4    1    -
5    2    1    |    5    2    -
6    3    1    |    6    3    -
7    7    -    |    7    7    -
8    8    -    |    8    8    -
9    9    -    |    9    9    -
10   10   -    |    10   10   -
```

```
wichteln3.txt
Maximale erfüllbare Wünsche:
1.  15
2.  6
3.  1

Zuordnung:  (G = Geschenk, P = Person, W = Wunsch)
G    P    W    |    P    G    W
1    5    -    |    1    2    1
2    1    1    |    2    20   1
3    6    1    |    3    29   1
4    9    1    |    4    8    1
5    7    -    |    5    1    -
6    11   1    |    6    3    1
7    18   2    |    7    5    -
8    4    1    |    8    12   1
9    12   1    |    9    4    1
10   20   1    |    10   28   1
11   17   -    |    11   6    1
12   8    1    |    12   9    1
13   22   -    |    13   14   2
14   13   2    |    14   23   2
15   24   2    |    15   26   1
16   19   1    |    16   30   1
17   25   3    |    17   11   -
18   26   -    |    18   7    2
19   21   2    |    19   16   1
20   2    1    |    20   10   1
21   28   2    |    21   19   2
22   27   -    |    22   13   -
23   14   2    |    23   27   1
24   29   -    |    24   15   2
25   30   -    |    25   17   3
26   15   1    |    26   18   -
27   23   1    |    27   22   -
28   10   1    |    28   21   2
29   3    1    |    29   24   -
30   16   1    |    30   25   -
```

```
wichteln4.txt
```

```
Maximale erfüllbare Wünsche:
1.  15
2.  4
3.  3

Zuordnung:  (G = Geschenk, P = Person, W = Wunsch)
G   P   W   |   P   G   W
1   17  1   |   1   28  1
2   15  1   |   2   21  1
3   7   2   |   3   22  2
4   9   -   |   4   14  1
5   6   3   |   5   6   2
6   5   2   |   6   5   3
7   13  3   |   7   3   2
8   28  1   |   8   16  3
9   11  -   |   9   4   -
10  14  -   |   10  19  1
11  16  -   |   11  9   -
12  12  1   |   12  12  1
13  20  1   |   13  7   3
14  4   1   |   14  10  -
15  23  1   |   15  2   1
16  8   3   |   16  11  -
17  19  2   |   17  1   1
18  27  1   |   18  27  1
19  10  1   |   19  17  2
20  21  -   |   20  13  1
21  2   1   |   21  20  -
22  3   2   |   22  23  -
23  22  -   |   23  15  1
24  26  1   |   24  25  -
25  24  -   |   25  30  1
26  29  -   |   26  24  1
27  18  1   |   27  18  1
28  1   1   |   28  8   1
29  30  1   |   29  26  -
30  25  1   |   30  29  1


wichteln5.txt
Maximale erfüllbare Wünsche:
1.  13
2.  1
3.  7

Zuordnung:  (G = Geschenk, P = Person, W = Wunsch)
G   P   W   |   P   G   W
1   21  1   |   1   25  3
2   6   1   |   2   6   1
3   20  3   |   3   7   1
4   7   1   |   4   19  3
5   9   -   |   5   27  3
6   2   1   |   6   2   1
7   3   1   |   7   4   1
8   16  2   |   8   18  1
9   10  -   |   9   5   -
10  15  1   |   10  9   -
11  24  1   |   11  14  1
12  28  1   |   12  13  1
13  12  1   |   13  16  -
14  11  1   |   14  15  1
15  14  1   |   15  10  1
16  13  -   |   16  8   2
17  29  1   |   17  26  3
18  8   1   |   18  23  3
19  4   3   |   19  20  -
20  19  -   |   20  3   3
21  22  -   |   21  1   1
22  23  -   |   22  21  -
23  18  3   |   23  22  -
24  25  -   |   24  11  1
25  1   3   |   25  24  -
26  17  3   |   26  28  -
```

```
27   5    3    |    27   30   3
28   26   -    |    28   12   1
29   30   -    |    29   17   1
30   27   3    |    30   29   -


wichteln6.txt (Ab hier keine Zuordnung mehr, da die Ausgabe sonst zu lang wird)
Maximale erfüllbare Wünsche:
1.  37
2.  3
3.  21

...


wichteln7.txt
Maximale erfüllbare Wünsche:
1.  541
2.  127
3.  58

...
```

5 Quellcode

```
1
2  bool find_path(int i, int order, vector<bool>& visited)
3  {
4      // falls der order-te Wunsch von Person i erfüllt werden kann
5      if(best_match[wuensche[i][order]] == -1)
6      {
7          erfuellt[order]++;
8          best_match[wuensche[i][order]] = i;
9          return true;
10     }
11     for(int j = 0; j <= order; ++j)
12     {
13         // Person, der wir ein neues Geschenk zuordnen müssen, falls wir Person i den j-
       ten Wunsch erfüllen
14         int next = best_match[wuensche[i][j]];
15         // Prüfe, ob dieses Geschenk wirklich j-te Wunsch von Person next ist
16         // Es könnte auch sein, dass dies der erste Wunsch von next ist, wir aber i
17         // nur den zweiten erfüllen.
18         // Die andere Richtung (der Wunsch von i hat eine höhere Priorität) kann nicht
       eintreten,
19         // da wir das Geschenk sonst schon in der vorherigen iteration Person i
       zugeordnet hätten
20         if(best_match[wuensche[next][j]] == next)
21         {
22             if(!visited[next])
23             {
24                 visited[next] = true;
25                 if(find_path(next, order, visited))
26                 {
27                     best_match[wuensche[i][j]] = i;
28                     return true;
29                 }
30             }
31         }
32     }
33     return false;
34 }
35
36 void solve()
37 {
38     // Betrachte erst Wünsche mit Priorität 1, danach 2 und 3
39     for(int i = 0; i < 3; ++i)
40     {
41         for(int j = 0; j < n; ++j)
```

```
42          {
43              // Falls Person j bereits ein Geschenk zugeordnet wurde, kann sie ü
    bersprungen werden
44              bool matched = false;
45              for(int k = 0; k < i; ++k)
46              {
47                  if(best_match[wuensche[j][k]] == j) matched = true;
48              }
49              if(!matched)
50              {
51                  vector<bool> visited(n, false);
52                  visited[j] = true;
53                  find_path(j, i, visited);
54              }
55          }
56      }
57  }
58
59  void assign_unmatched_gifts()
60  {
61      queue<int> unmatched_gifts;
62      for(int i = 1; i <= n; ++i)
63      {
64          if(best_match[i] == -1) unmatched_gifts.push(i);
65      }
66      for(int i = 0; i < n; ++i)
67      {
68          bool matched = false;
69          for(int k = 0; k < 3; ++k)
70          {
71              if(best_match[wuensche[i][k]] == i) matched = true;
72          }
73          if(!matched)
74          {
75              best_match[unmatched_gifts.front()] = i;
76              unmatched_gifts.pop();
77          }
78      }
79  }
```