ДАА Семинар 11
Алгоритми върху графи

Деф. Топологическо сортиране на ориентиран граф $G=(V,E)$

Топологическо сортиране на G е всяка биекция $h: V \to \{1,\dots,n\}$, такава че за всяко ребро $(u,v) \in E$ е в сила, че $h(u) < h(v)$

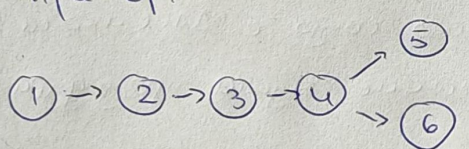
// **Заб.** Съществува топологическо сортиране на G тстк G е ДАГ.

- ако G е неориентиран и съдържа ребро (u,v), то G съдържа и реброто (v,u). Каквато и биекция $h: V \to \{1,\dots,n\}$ да изберем, не е възможно $h(u) < h(v) \wedge h(v) < h(u)$ да е изпълнено
- ако G е ориентиран и има цикъл $a_{i_1} \to a_{i_2} \to \dots \to a_{i_k} \to a_{i_1}$, то каквато и биекция $h: V \to \{1,\dots,n\}$ да вземем, не е възможно $h(a_{i_1}) < h(a_{i_2}) < \dots < h(a_{i_k}) < h(a_{i_1})$ да е изпълнено.

зад (1) Да се напише алгоритъма на Тарян за топ. сортиране

- Същина на алгоритъма: реди върховете по намаляващо време на финализиране (от DFS). Ако G е ДАГ, то той има поне един сифон. Неформално казано, сифоните трябва да имат най-големи h-стойности. Първият открит сифон има най-голяма h стойност. След като сме задали h-стойност на първия открит сифон, остава да направим топологическо сортиране на $G \setminus \{v\}$, което също е ДАГ и има поне един сифон. Процедурата се повтаря докато не се обработят всички върхове.

// Пример:



е представен чрез списъци на съседство:

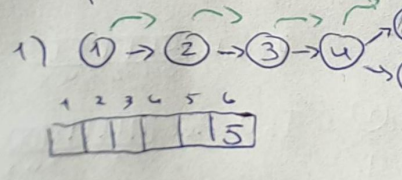
1 → {2}
2 → {3}
3 → {4}
4 → {5, 6}
5 → {}
6 → {}

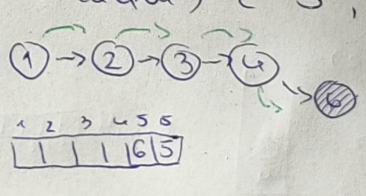
„Пускане" DFS от връх (1)
↳ открива върховете в този ред:

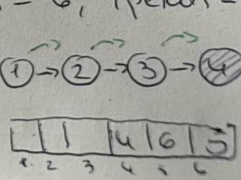
1 → 2 → 3 → 4 → 5 → 6

„→" - път на DFS

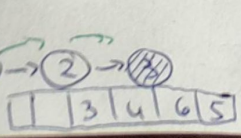
Първият финализиран връх (той е сифон) е 5, вторият – 6, третият – 3 . .

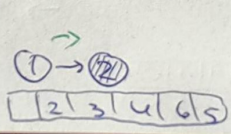
1) 

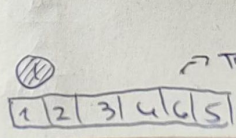
→ 2) 

→ 3) 

→

→ 4) 

→ 5) 

→ 6) 

↱ топ. сорт. на G

Псевдокод:

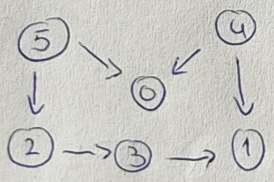
```
TarjanTopSort(G=(V,E)) - чрез сп.н.с)
1. TopSort[1..n], index←n, time←0, color[1..n] // π[1..n]
2. for i←1 to n
3.     color[i]←white //, π[i]←nil
4. for i←1 to n
5.     if color[i]==white
6.         TarjanRec(G, i, color, time, index, TopSort)
7. return TopSort[1..n]

TarjanRec(G=(V,E), s, color[1..n], time, index, TopSort[1..n])
1. time←time+1
2. color[s]←gray, // d[s]←time
3. for y∈adj[s]
4.     if color[y]==white
5.         // π[y]←s, TarjanRec(G, y, color, time, index, TopSort)
6. color[s]←black
7. time←time+1, // f[s]←time
8. TopSort[index]←s
9. index←index-1
```

↳ ако G не е ДАГ не сигнализира

зад ② Да се напише алгоритъма на Kahn за топологическо сортиране

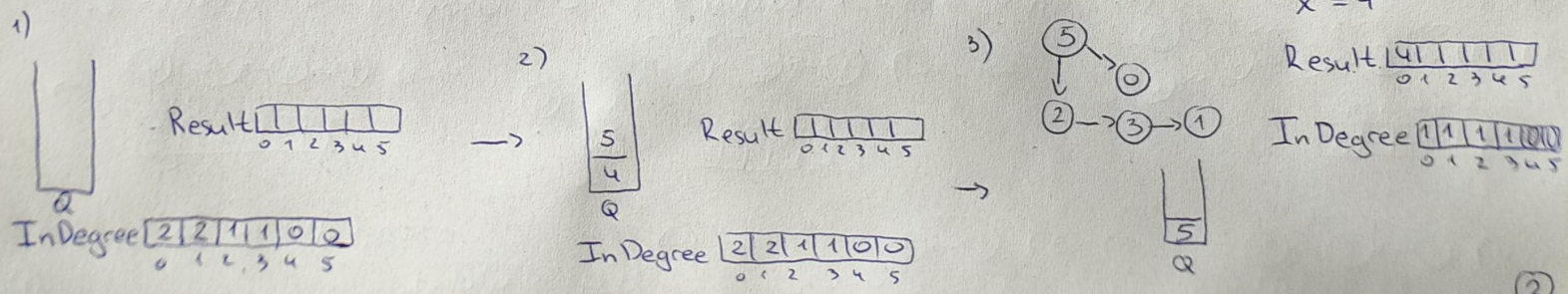
- Същина на алгоритъма: реди върховете като започва от източниците. Ако G е ДАГ, то той има поне един източник. Неформално казано, източниците трябва да имат най-малки h-стойности. Първият „открит" източник има най-малка h-стойност. След като сме задали h-стойност на v, остава да направим топ. сортиране на G\{v}, което също е ДАГ (ако са останали върхове) и има поне един източник. Прилагаме същата процедура докато не обработим всички върхове.

// пример:



представен чрез:

0→{}
1→{}
2→{3}
3→{1}
4→{0,1}
5→{0,2}



4) ⓪ → ② → ③ → ①

x = 5

Result: [4|5| | | | |] (0 1 2 3 4 5)

InDegree: [0|1|1|1|0|0] (0 1 2 3 4 5)

Q: [2 | 0]

→

5) x = 0

② → ③ → ①

Result: [4|5|0| | | |] (0 1 2 3 4 5)

InDegree: [0|1|0|1|0|0] (0 1 2 3 4 5)

Q: [2]

→

→ 6) ③ → ① x = 2

Result: [4|5|0|2| | |] (0 1 2 3 4 5)

InDegree: [0|1|0|0|0|0] (0 1 2 3 4 5)

Q: [3]

→

7) ① x = 3

Result: [4|5|0|2|3| |] (0 1 2 3 4 5)

InDegree: [0|0|0|0|0|0] (0 1 2 3 4 5)

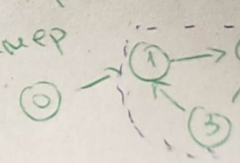
Q: [1]

→ 8) Result: [4|5|0|2|3|1] (0 1 2 3 4 5) ✓

Псевдокод:

```
KahnTopSort(G=(V,E) - чрез списъци на съседство)
1.  S ← ∅ , InDegree[1...n]{0} , color[1..n]
2.  for x ← 1 to n
3.      for y ∈ adj[x]
4.          InDegree[y] ← InDegree[y]+1
5.  for i ← 1 to n
6.      if InDegree[i] == 0
7.          S ← su{i}                          → T(n,m) ≈ Θ(n+m)
8.          color[i] ← gray
9.      else
10.         color[i] ← white
11. index ← 1
12. while S.notEmpty()
13.     x ← S.popFront() // приемаме, че S е опашка в случая
14.     TopSort[index] ← x
15.     index ← index+1
16.     for y ∈ adj[x]
17.         InDegree[y] ← InDegree[y]-1
18.         if InDegree[y] = 0
19.             S ← su{y}
20.             color[y] ← gray
21.     color[x] ← black   // в хода на цялия алгоритъм трябва да сме открили точно и върха, които в някакъв момент са били източници в необработената част на графа, ако G е ДАГ
22. if index < n   →
23.     cout << "G is not a DAG"
24. else
25.     return TopSort[1..n]
```

→ пример



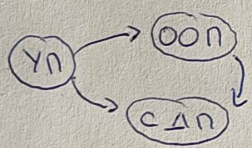
→ след обработването на връх 0, който е източник, в необработената част на графа има източници ⇒ графът е ДАГ

Зад ③ Учебна програма

Дадени са N предмета, именовани от 1 до N. Някои предмети имат изисквания; например, за да се вземе предмет/курс с номер K, първо трябва да се вземе курс с номер P (за да се вземе ООП, първо трябва да се вземе УП или)
Изискванията са представени чрез двойки (p, k) // примера горе.
По дадена N и списък от изисквания да се изведе възможен график за последователността на вземане на курсовете, ако такъв съществува, в противен случай да се изведе съобщение за грешка

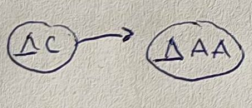
// пример
Изискванията са:
(УП, ООП) // за да вземем ООП първо трябва да сме взели УП
(УП, СДП)
(ООП, СДП)
(ДС, ДАА)
(ДИС1, ДИС2)

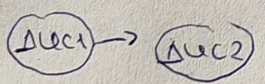
↳ в нашия график първо трябва да се появи УП, за да се появи, например, ООП ...
↳ конструираме следния граф



и намиране някое топологическо сортиране, то ще отговаря на валиден график, например:

[УП, ООП, СДП, ДС, ДАА, ДИС1, ДИС2]





Псевдокод

```
Schedule (N, Requirements - списък от двойки)
1. AdjLists[1...N] // вектор от вектори
2. for i ← 1 to Requirements.size()
3.     AdjList[Requirements[i].first].pushBack(Requirements[i].second)
4. KahnTopSort(AdjLists)
```

Зад (4) Извънземен речник

Даден е извънземен речник (сортиран) от N извънземни думи, състоящи се от първите K символа на тяхната азбука.

Да се намери правилна последователност на тези K символа в извънземната азбука (може да има повече от една правилна последователност - да се върне коя да е, ако няма правилна последователност, да се изведе съобщение, че такава азбука не съществува).

// Пример

$N=5$, $K=4$

Речник:

baa, abcd → b е преди a в извънземната азбука

abca → d е преди a "—"

cab → a е преди c "—"

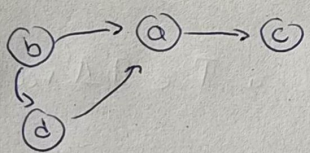
cad → b е преди d "—"

правилен ред е: (b, d, a, c)

Решение:

Елементите в задача (извънземните букви) са в релацията на предшестване → $\forall x,y \in \{\text{извънземна азбука}\}$ $xR^{\text{предшестване}}y \leftrightarrow$ x се появява преди y в азбуката. Ще конструираме релацията по подадения като вход речник. Ако тя се окаже антирефлексивна и съответния ѝ граф няма цикли, кое да е нейно топологическо сортиране ще ни свърши работа.

// От примера по-горе получаваме този граф:



// ако някоя от буквите не се среща в речника, я приемаме за изолиран връх (в случая, ако $k=5$, то "е" ще да е изолиран връх).

Ще представяме всяка буква с число $\in \{1,\dots,k\}$ и за всеки две съседни думи в речника ще намираме разликата, която ще фактор. Нека u,v са две съседни думи в речника като u е (непосредствено) преди v. ($u=u_1\dots u_m$, $v=v_1\dots v_t$) Нека $u_x \neq v_x$ е първата позиция, в която двете думи се различават. В този случай добавяме ребро (u_x, v_x) към графа.

Псевдокод

↗ списък от думи (стрингове)

AlienAlphabet(N, k, Dictionary)

1. graph[1...k] // fies1,..k} graph[i] е списъка на съседство на i // vector<int> graph[k];
2. for i ← 1 to ?-1
3. word1 ← Dictionary[i]
4. word2 ← Dictionary[i+1]
5. for j ← 1 to min(word2.size(), word1.size()) — a съответства на 0
6. if word1[j] != word2[j] ↗ b „–" 1
7. graph[word1[j]-'a'].push_back(word2[j]-'a');
8. break;
9. KahnTopSort(graph)

// няма такава извънземна азбука, ако, например, в азбуката има съседните двойки:

1) a b c d
   a b c

   или

2) a b c
   b a d
   a d c

a < b < a