ДАА семинар 9

Графи

Деф **Граф** $G=(V,E)$ е наредена двойка от

V – множество върхове и
$E \subseteq \{x \mid x \subseteq V \wedge |x|=2\}$ – множество от ребра

Деф **Ориентиран граф** $G=(V,E)$, където V е множ. от върхове, а E е множеството от ребра, т.е

$$E \subseteq (V \times V) \setminus \{(v,v) \mid v \in V\}$$
без примки

Деф **Ориентиран мултиграф** $G=(V,E,f_G)$, където
V – множество върхове
E – множество ребра
f_G – свързваща функция, т.е $f_G : E \to V \times V$

Деф **Свързаност** в граф: Казваме, че $G=(V,E)$ е свързан, ако за вс. $u,v \in V$ съществува път м/у тях.

$\Rightarrow$ чрез релация на еквивалентност, породена от това

$\Rightarrow$ **Свързана компонента** е максимален по включване подграф на G, който е свързан.

Деф **Силна свързаност** в ориентиран граф $G=(V,E)$

1) За $u,v \in V$ казваме, че u и v са **силно свързани**, ако

ма (ориентиран) път от u до v и от v до u.

2) G е силно свързан, ако 1) е изпълнено за $\forall u, v \in V$

3) SC – релация на силна свързаност

4) SCC – силносвързани компоненти – подграфите, индуцирани от класовете на еквивалентност на SC

5) (слабосвързани компоненти са свързаните подграфи в съответния на G неориентиран граф.

Деф. Тегловен граф $G=(V,E,w)$, където V и E са стандартните множества, а $w: E \to \mathbb{R}$ е тегловна функция

Означения: $|V|=n$, $|E|=m$

размер на G: $n+m$

разреден граф: $m \leq n$

Основни задачи за графи:

- свързаност на два върха
- идентифициране на (силно) свързани компоненти
- двуделност
- откриване на цикли
- най-къси пътища
- топологично сортиране

и др.

Представяния на графи в паметта:

1) Матрица на съседство → подходяща за гъсти графи

2) Списъци на съседство → най-добър в средния вариант
→ лесно взимане на съседи

3) Списък от ребра → най-икономичен откъм памет
→ бавна проверка за съседи

	Standard adjacency list (linked lists)	Fast adjacency list (hash tables)	Adjacency matrix
Space	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(V^2)$
Test if $uv \in E$	$O(1+\min\{\deg(u),\deg(v)\}) = O(V)$	$O(1)$	$O(1)$
Test if $u \to v \in E$	$O(1+\deg(u)) = O(V)$	$O(1)$	$O(1)$
List v's (out-)neighbors	$\Theta(1+\deg(v)) = O(V)$	$\Theta(1+\deg(v)) = O(V)$	$\Theta(V)$
List all edges	$\Theta(V+E)$	$\Theta(V+E)$	$\Theta(V^2)$
Insert edge uv	$O(1)$	$O(1)^*$	$O(1)$
Delete edge uv	$O(\deg(u)+\deg(v)) = O(V)$	$O(1)^*$	$O(1)$

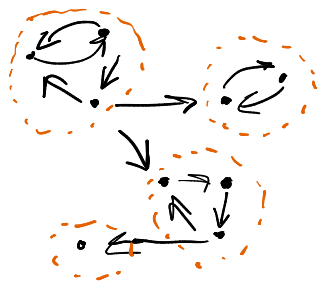
Table 5.1. Times for basic operations on standard graph data structures.

Взето от: https://jeffe.cs.illinois.edu/teaching/algorithms/

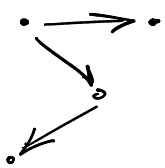
- Добре е да се знаят осн. алг. от лекции - наивно и как го правят
  +
  дефиниции и свойства за графи, вил. от ДС

Пр за „оптимизация" на база свойства на граф:

Ако търсим път м/у 2 върха в ориентиран граф, можем да разглеждаме фактор-графът по релацията на силна свързаност: Дали има път от компонентата на единия до тази на другия?



Компонентен граф:



Обхождане на графи → BFS и DFS

$T(n,m) = \Theta(n + m.T)$, където T е времето за добавяне в съответния контейнер.

($T = 1$)

$S(n,m) = \Theta(n)$ // И двете са само ползване списъци на съседство.

Зад. Даден е граф $G=(V,E)$. Да се състави ефикасен алгоритъм, който проверява дали G е двуделен.

Нека БОО $V=\{1,\dots,n\}$ и G е свързан
↳ ако не е, извикваме този алг. за отделните компоненти

Решение:

```
isBipartite( G=(V,E) )
  colours[1..n] <- [red, Nil, ... Nil]
  q <- празна опашка
  q.enqueue(1)
  while not q.isEmpty()
      u <- q.dequeue()
      for each (u,v) ∈ E
          if colour[v] = Nil
              if colour[u] = red
                  colour[v] = black
              else
                  colour[v] = red
              q.enqueue(v)
          else if colour[u] = colour[v]
              return False
  return True
```

Зад. Дадена е шахматна дъска с размер $n \times n$, върху която има кон, топ и царица. Топът и царицата не се местят. С колко най-малко стъпки може конят да ги вземе?

Решение: Ще построим граф:
- върховете са различните полета на дъската // $n \geq 2$, за да има място
- ребрата са възможните движения на коня.

- м/у 2 върха има ребро ТСК между съответните полета конят може да се придвижи за 1 ход.

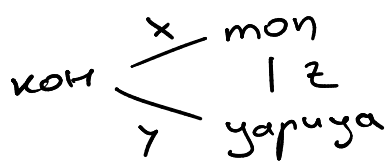
Този граф е неориентиран и $|V| = n \cdot n = n^2$, а

$|E| = 4(n-1)(n-2) = \Theta(n^2)$ // Knight's graph

Ако намерим най-къс път между съответните полета, ще намерим най-малък брой стъпки, както се иска в условието.

1) Чрез BFS намираме най-къс път до другите две фигури.

2) BFS от топ (царица) до царица (топ)



Няма как да пропуснем 2, защото търсим по-добрия път: $\min(x,y) + 2$

$T(n) = \underbrace{O(n^2 + n^2)}_{\text{върхове}} + \underbrace{\Theta(n^2+n^2)}_{\text{BFS}} = \Theta(2n^2) = \Theta(n^2)$ (ребра)

$S(n) = \underbrace{\Theta(n^2 + n^2)}_{\text{създаване}} + \Theta(n^2) = \Theta(n^2)$

- В задачата се възползваме от това, че BFS строи дърво на най-късите пътища между стартовия връх и останалите достижими върхове в претеглен граф.

Зад Дадени са стаи с номера от 1 до n. Всяка стая държи ключ, за да бъде посетена. Целта е да се посетят всички. Във всяка стая има набор от различни ключове.

Предложете алг., който по даден масив $A[1..n]$, т.ч. $A[i]$ съдържа списък на ключовете от стая i за $1 \le i \le n$, и стартова стая връща True ако всички стаи могат да бъдат посетени; False иначе.

Решение:

Може да използваме и BFS, и DFS за тази задача. Ще използваме BFS. Защо е по-добрия вариант?

```
canVisit(A[1..n], start)
1   visited ← ∅
2   toVisit ← празна опашка
3.  enqueue(toVisit, start)
4.  while not isEmpty(toVisit)
5       x ← dequeue(toVisit)
6       foreach y ∈ A[x]
7           if y ∉ visited
8               enqueue(toVisit, y)
9   return |visited| = n
```

Правим лека модификация на BFS. Вместо к съседи, разгл. само тези, за които имаме ключ. Няма нужда да си „носим" ключовете докато обхождаме в отделен масив.

DFS също работи, но може да предизв. стека (на практика), ако n е много голямо. Също, трябва да запомним кои ключове имаме.

Сложността е като на BFS.

Коректността следва от коректността на BFS, тъй като разглеждаме граф, който описва достижимите стаи: Редро м/у два върха има тстк имаме ключ за съответните стаи.

Задача:

НАЦИОНАЛНА ОЛИМПИАДА ПО ИНФОРМАТИКА
Областен кръг, 18 март 2016 г.
Група В, 9 – 10 клас

ЗАДАЧА В2. ГРАД

Автори: Добромир Кралчев и Емил Келеведжиев

В един град всички улици са еднопосочни. В града има n кръстовища, номерирани с целите числа от 1 до n. Даден е списък на двойки кръстовища p и q, които са свързани с еднопосочна отсечка от улица, така че по тази отсечка от улица няма други кръстовища. За всеки две кръстовища p и q съществува най-много една еднопосочна отсечка от улица в посока от p към q, или в посока от q към p.

При спазваме правилата за движение в града, невинаги е възможно да отидем с кола от кръстовище a до кръстовище b. С колко най-малко нарушения, обаче, може да се придвижим от a до b? Всяко навлизане в посока обратна на разрешената в еднопосочна отсечка от улица се брои за едно нарушение.

Напишете програма **`town`**, която намира минималния брой на нарушенията.

Вход

В първия ред на стандартния вход са зададени стойностите на три цели числа n, a и b, където n е броят на всички кръстовища, a е номерът на кръстовището, от което тръгваме и b е номерът на кръстовището, в което трябва да отидем. Следват толкова на брой редове, колкото са еднопосочните отсечки от улици. Във всеки от тези редове са дадени по две числа p и q – номерата на кръстовищата, за които съществува еднопосочна отсечка от улица с посока от p към q.

Изход

На стандартния изход програмата трябва да изведе едно цяло число – намерения минимален брой. Ако е възможно да се премине без нарушения – програмата трябва да изведе 0. Ако въобще не е възможно да се премине – програмата трябва да изведе главната латинска буква `X`.

Ограничения

$1 \leq n \leq 200\,000$; броят на еднопосочните отсечки от улици не е по-голям от 400 000.

Примери

Пример 1

Вход

```
4 1 4
4 3
3 2
1 2
4 2
```

Изход

```
1
```

Пример 2

Вход

```
4 1 4
4 3
3 2
4 2
```

Изход

```
X
```

Решение:

Идея: Построяваме граф:
- върховете са кръстовищата
- за всяка дадена улица от u към v добавяме в графа ребро (u,v) с тегло 0 и ребро (v,u) с тегло 1.

Използваме модифициран BFS, който използва приоритетна опашка - по минимално тегло, вместо обикновена.

Допълнително, в модифицираната трябва да се вложи проверка за подобряване на разстоянието.

Реализирайте задачата на C++.

→ можем и да използваме deque, като ако теглото е 0, добавяме в началото, ако е 1 – в края.

Този модификация на BFS се нарича 0–1 BFS и е частен случай на алгоритъма на Dijkstra

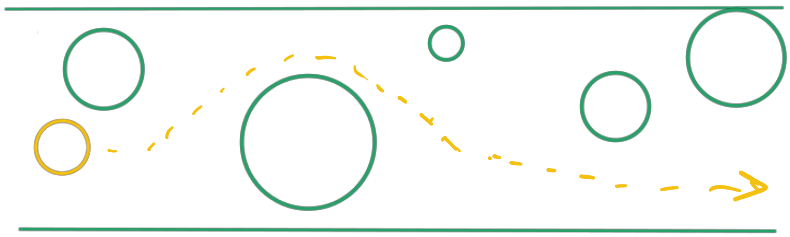
- BFS може да се използва и за откриване на цикъл в граф.

  → Ако от текущия връх има ребро до вече посетен връх, който не е негов сосед/родител, то значи има цикъл.

- Когато BFS използва приоритетна опашка, то алг. се нарича Best-First Search

$$T(n,m) = O(n + m \lg m)$$

Зад. В двумерното пространство е дадена тръба с препятствия – кръгове с център (x,y) и радиус r. Знаем и ширината на тръбата. Хамстер е в кръг с даден радиус. Може ли хамстерът да премине от единия край до другия край на тръбата?



Решение: Построяваме граф с по 1 връх за всяко препятствие, както и 2 върха за стените.

Между два върха има ребро тогава разстоянието между съответните геометрични обекти е по-малко от размера на хамстеровия кръг. (диаметъра)

Отгоре, хамстерът може да премине тогава няма път между върховете съответно за двете стени.

$$T(n) = \underbrace{\Theta(n^2)}_{\text{граф}} + \underbrace{\Theta(n^2)}_{\text{BFS}} = \Theta(n^2)$$

$$S(n) = \underbrace{O(n^2)}_{\text{граф}} + \underbrace{\Theta(n^2)}_{\text{BFS}} = O(n^2)$$

DAG и Топосортиране

- DAG = Directed Acyclic Graph
  - поне един източник, поне един сифон

- Топологическо сортиране за $G = (V, E)$

дискретна $h: V \to \{1..n\}$, т.ж. $\forall (u,v) \in E \quad h(u) < h(v)$

- G е DAG $\Leftrightarrow$ $\exists$ топосортиране

- Можем да намерим топологическо сортиране за $\Theta(n+m)$ време и $\Theta(n)$ памет

Пример. Намиране на най-голям път в DAG.

```
LongestPathDAG( G=(V,E) - DAG)
1  A ← Topsort(G)
2  d[1..n] - ако е попълнена d[i] съдържа дължина
           на най-голям път, започващ в съотв. връх.
3  foreach v ∈ reverse(A)
4     d[v] ← 0
5        foreach u ∈ adj[v]
6           d[v] ← max(d[v], d[u]+1)
7  return max(d[1..n])
```

// Лесно може да се модифицира да връща върха

- Как модифицираме алг., ако G е претеглен?

Зад. Колко най-много ребра може да има в DAG?

Решение: $(n-1) + (n-2) + \dots + 1 + 0 = \frac{n(n-1)}{2} = \Theta(n^2)$

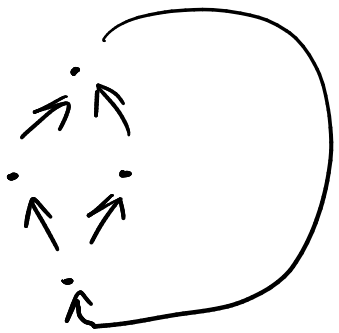
- Нека $G=(V,E)$ е ориентиран граф. Компонентният му граф е DAG (ако има цикли, те са в една силносвързана компонента)

Зад. Докажете или опровергайте:

а) 1 SCC $\Leftrightarrow \exists$ HC

SCC – силносв. комп.
HC – Хамилтонов цикъл
HP – Хам. път

($\Rightarrow$) Не е вярно:



→ За SCC може да има пресичащи се пътища.

За HC не може.

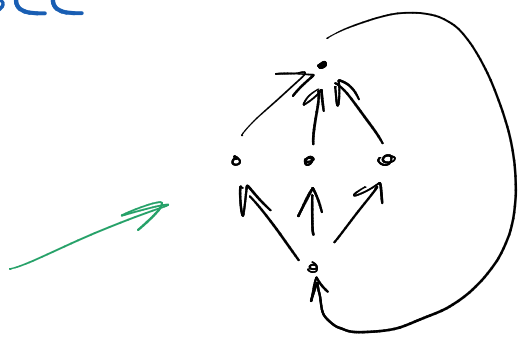
($\Leftarrow$) Да, вярно е:

От всеки връх може да стигнем до всеки друг, използвайки цикъла

Т.е.: $\exists$ HC $\Rightarrow$ 1 SCC

б) 1 SCC $\Leftrightarrow \exists$ HP

($\Rightarrow$) Не, както в а)



($\Leftarrow$) Не, контрапример:

Има HP, но 2 SCC