ДАА Семинар 10

Графи:
- обикновени или мултиграфи
- ориентирани или не
- с или без примки
- тегловни или не

- 2 вида свързаност: силна и слаба за ориентирани

- Различни представяния в паметта:
  - матрица на съседство → подходяща за гъсти графи
    → константно време за проверка дали има ребро
  - списъци на съседство → най-добър в средния вариант (за по-редки графи)
    → лесно взимане на съседи
  - списъци от ребра → най-икономичен откъм памет
    → бавна проверка за ребро/съседи

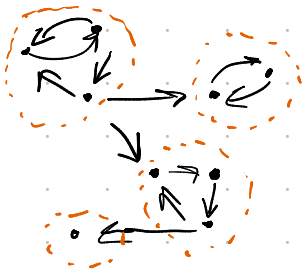
- Основни задачи:
  - свързаност на два върха
  - идентифициране на (силно) свързани компоненти
  - двуделност
  - откриване на цикли
  - най-къси пътища
  - мостове, срязващи върхове
  - топологично сортиране

  и др.

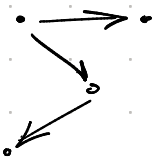
- Да се знаят основните алг. от лекции - как и какво правят
  + дефиниции/свойства за графи

Пр за „оптимизация" на даден 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:



- За $G=(V,E)$ означаваме $|V|=n$, $|E|=m$

Размер на G: $m+n$

Разреден граф: $m \leq n$

Обхождане в матрица и в далбочина:

$T(n,m) = \Theta(n+m)$ $\quad S(n,m) = \Theta(n)$ //Ако използваме списъци на следство

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

Решение: Ще построим граф:
- върховете са различните полета на дъската
- ребрата са възможните движения на коня.

//$n \geq 2$, за да има място

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

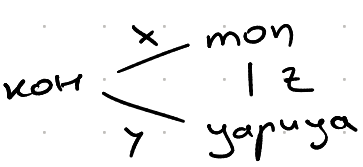
Този граф е неориентиран и $|V| = n.n = n^2$, а

$|E| = 4(n-1)(n-2) = \Theta(n^2)$ // 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 (дама) до дама (топ)



Дали $x+z < y+z$?
Отг. е $\min(x,y) + z$

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

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

Зад Дадени са стаи с номера от 1 до n. Всяка стая съдържа 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.ж $A[i]$ 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:

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

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

Правим лека модификация на BFS. Вместо ∀ 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 препълни 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: Ребро м/у два върха има ТСТК имаме ключ за съответните стаи.

Зад Условие → файл week10-LongTask.pdf

Идея: Построяване 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.

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

Решение: isBipartite (G=(V,E)) $V = \{1..n\} \leq 500$

```
1   colours[1..n] ← [red, Nil, ... Nil]
2   q ← пустая очередь
3   q.enqueue(1)
4   while not q.isEmpty()
5       u ← q.dequeue()
6       for each (u,v) ∈ E
7           if colour[v] = Nil
8               if colour[u] = red
9                   colour[v] = black
10              else
11                  colour[v] = red
12              q.enqueue(v)
13          else if colour[u] = colour[v]
14              return False
15  return True
```