(Д4) Имаме неориентиран свързан граф
$G = \langle V, E \rangle$, $\omega : E \longrightarrow \{a, b\}$
$0 < a < b$.

Нека r е теглото на някое MST на G.
Как да намерим това r с
линеен алгоритъм, когато е
нужно да намира MST.

Реш/ Нека разгледаме $G' = \langle V, E' \rangle$
подграф, т.ч. $E' \subseteq E$ и за
$E' = E \setminus \{e \mid \omega(e) = a\}$.
G' може да е свързан и тогава
това ST G' е всъщност ST за G.
Не сме сигурни дали е MST
G'. Знаем със сигурност, че
$r \geq (n-1).a$ със сигурност.
Ако G' не е свързан, то със
сигурност имаме няколко компоненти
на свързаност (поне 2). Нека бележим
броят им с $t \geq 2$ и с $V_1, \ldots, V_t$ са
броят върхове във всяка една.

Тогава във всяка компонента може да има излишни ребра.
От i-тата компонента на свързаност взимаме $(v_i - 1)$ ребра стойност a за (прим. dfs пускаме) $i \in \{1, \dots, t\}$, т.с. запазваме свързаността.
100% още $(t-1)$ тежки ребра трябват от тип b ребра да добавим, за да стане свързано дървото.
Всяко едно от добавените свързва точно две компоненти на свързаност.

Тогава

$$r = \sum_{i=1}^{t} (v_i - 1) \cdot a + (t-1) \cdot b =$$

$$= a \cdot \Big(\underbrace{\sum_{i=1}^{t} v_i}_{n} - t \Big) + (t-1) \cdot b =$$

$$= a \cdot (n - t) + (t-1) \cdot b$$

Зад Намерете диаметъра на дърво.

Реш 1 Диаметър на дърво - най-дълъг път в дървото. Как за O(|V|) да го намерим? Първо наблюдение: най-дългият ще бъде винаги м/у две листа. Тогава от произволно връх нека стартираме dfs и запишем върха, който е най-отдалечен от него ви. И е листо и сега нека от него стартираме dfs, за да стигнем до най-отдалеченото от него листо. Това е диаметъра на дървото. Ще ви прикача pdf с псевдокод и обяснения.

CLRS 22.2-8 е номера в int.GraphA4.pdf

Може и с bfs в int.GraphA5.pdf пак за O(|V|) времева сложност

Зад) Модифицирайте алгоритъма на Floyd-Warshall за намира транзитивно затваряне на матрицата на съседство.

Реш:/

```
Floyd-Warshall(G, w)
    r[1,...,n][1,...,n] : reachebility
    init(G, w, r)
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                r[i][j] ←
reach[i][j] v (reach[i][k] & reach[k][j])

init(G, w, r)
    for i ← 1 to n do
        for j 1 to n do
            if <i, j> ∈ E
                r[i][j] ← 1
            else r[i][j] ← 0
```

зад) Даден е ориентиран граф $G=\langle V, E\rangle$ без цикли. Предложете алгоритъм с време $O(|V|+|E|)$, който намира броя пътища между два върха s и t.

Реш: Сортираме топологично G и въвеждаме dist с $|V|$ на брой елемента. Идеята е за $v \in V$, то dist[v] = # пътища от v до t.

Pseudocode:

```
alg(G, s, t):
    e ← topoSort(G)
    paths[|V|] ← 0   // всички са 0 в началото
    paths[t] ← 1   (път от t до t тривиален)
    for u in reverse(e) do
        if u is before t in e
            for v ∈ adj(u)
                paths[u] ← paths[u] +
                           paths[v]
    return paths[s]
```

$O(|V|+|E|)$ за реди

Защо обхождаме върховете

в обратен ред на топологичната сортировка? Какво става с всички върхове, които не са достижими от t?
От върховете недостижими от t не се интересуваме и затова $paths[\cdot] = 0$.
В обратен ред ги обхождаме, защото първо запазваме непосредствените предшественици на t за обработване след това техните на базата на наследниците им и т.н.
Тук се вижда как прилагаме схемата за ДП. Решаваме последователно подзадачи, чието обединение в края ни дава търсеното решение.
Защо $paths[t] = 1$? Тривиален път е от t до t затова и го броим.

* Два класа графи автоматично изключват възможност да имат негативни цикли: DAGs и графи само с положителни тегла.

Зад. Най-къс път и най-дълъг път в DAG: $G = \langle V, E \rangle$ от връх v до всички останали.

Реш.: Можем да направим topoSort на G и след това да посещаваме върховете в реда на топологичното им сортиране и да ъпдейт-ваме ребрата излизащи от тях.

Важното тук е, че във всеки път в един DAG, върховете се появяват във възходяща линейна наредба. Така търсенето на път от v до всички останали в DAG става линейно спрямо $|V|$ и $|E|$.

Модифициране Bellman-Ford
като преди големия цикъл
правим topoSort след което в
него взимаме върховете в реда
определен от topoSort и в
края няма да изпускаме търсенето
за негативен цикъл (в DAG
няма цикли).

Най-дългият път от v до всеки друг
намираме със същото.
но преди това умножаваме по
-1 всички тегла и тогава
пускаме горния алгоритъм или
вместо min да взимаме, ще
взимаме max.

Modified Bellman-Ford +
topologicalSort

```
alg(G, w, s):
  L ← topoSort(G) // O(|V|+|E|)
  dist[|V|]
  for v in V do
      dist[v] ← +∞
  dist[s] ← 0
  for v in L \ {s} do
      for <u,v> in E do
          if dist[v] > dist[u] + w(<u,v>)
              dist[v] ← dist[u] + w(<u,v>)
  return dist
```

Реално $\boxed{dist[v] = \min_{\langle u,v\rangle \in E} \{dist[u] + w(\langle u,v\rangle)\}}$

За най-дълъг път в DAG вместо min, пишем max в тази ф-ла!

Реално алгоритъма решава множество от подпроблеми

$\{dist[v] \mid v \in V\}$ които започваме с най-малкия, най-лесния, чийто отговор знаем: $dist[s] = 0$. Тогава вървим нагоре към по-големите подпроблеми прогресивно по въведената линейна наредба между върховете. Във всеки връх изчисляваме ф-я, която зависи от стойностите ѝ в/у предшествениците. В нашия случай е минимумът от път до и предшественик на $v + w(\langle u,v \rangle)$. Иначе и за е друга.

Навлизаме в темата за динамично програмиране.