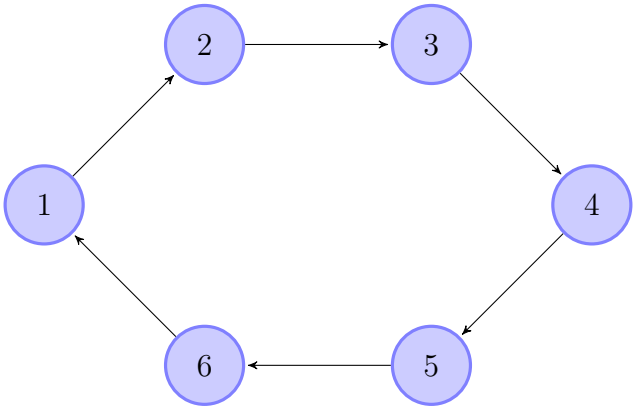

Задача 1: Нека $G = (V, E)$ е ориентиран граф. Казваме, че G е *забележителен*, ако за всеки $u, v \in V$, такива че $u \neq v$, е вярно, че съществува най-много един прост ориентиран път от u до v.

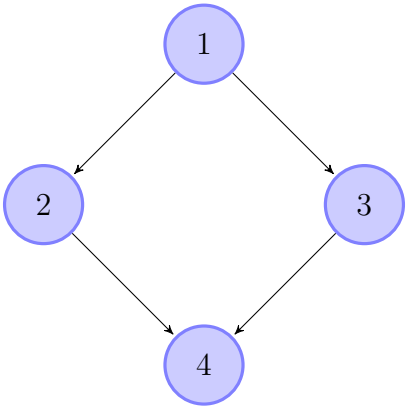
2 т.
- Обяснете с пример защо "забележителен граф" не е същото като "dag".

23 т.
- Предложете колкото можете по-ефикасен алгоритъм, който изчислява дали G е забележителен. Обосновете в общи линии коректността на Вашия алгоритъм.

Решение: В учебника на Cormen, Leiserson и Rivest, този тип ориентирани графи е наречен *singly connected* (стр. 612 в третото издание). Тези графи не са непременно дагове, нито даговете са непременно такива. Всеки прост ориентиран цикъл е такъв граф, без да е даг:



От друга страна, лесно се конструира даг, в който има повече от един път от един връх до друг; да кажем, от връх 1 до връх 4:



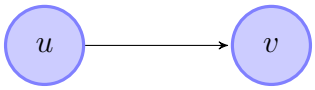
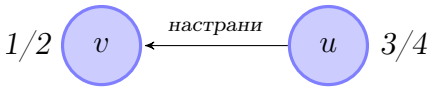
Сега ще конструираме алгоритъм, който изчислява дали даден ориентиран граф е singly connected. Алгоритъмът е много прост: за всеки връх u от графа, пускаме DFS-VISIT(G, u). Ако за поне един връх u, DFS-VISIT(G, u) открие ребро (u, v), което е

- или ребро напред,
- или ребро настрани, но такова, че u и v се намират в едно и също дърво на гората на обхождането,

то G не е singly connected. В противен случай G е singly connected.

Сложността по време очевидно е $\Theta(n(n+m))$, тоест, в най-лошия случай алгоритъмът е кубичен в n. Известен е и *квадратичен в n алгоритъм за тази задача*, но той е прекалено сложен за домашна работа.

Сега да се убедим, че нашият алгоритъм е коректен.

> *Една странична забележка. В посочената статия се казва "If each DFS yields only tree and back edges then the graph is singly connected". Това е вярно, но е еднопосочно твърдение. Конверсното не е вярно: може пускане на DFS да открие ребро настрани и въпреки това графът да е singly connected. Ето съвсем прост пример:*
>
> 
>
>
> *Ако DFS започне върху u, реброто (u, v) е дървесно:*
>
> 
>
>
> *Ако DFS започне върху v, реброто (u, v) е ребро настрани:*
>
> 
>
>
> *А графът очевидно е singly connected. Но краищата на това ребро настрани са от различни дървета от гората на обхождането, а именно $(u, \emptyset)$ и $(v, \emptyset)$.*

Доказателството в едната посока Наличието на ребро напред или ребро настрани, но в едно дърво от гората на обхождането, влече, че графът не е singly connected.

- Нека $e = (u, v)$ е ребро напред. Тогава u и v са в отношение на предшествие в дървото, като u предшества v. Тогава има ориентиран път p от u до v, който не съдържа e, а се състои от дървесни ребра. Реброто e се явява алтернативен път от u до v. Показахме, че има два върха, такива че от първия до втория има два различни пътя.

- Нека $e = (u, v)$ е ребро настрани, но и двата му края са от едно и също дърво T от гората на обхождането, като нито u предшества v в T, нито v предшества u в T. T е кореново дърво-арборесценция. Нека коренът му е r. Тогава в T има уникален път p_1 от r до u и уникален път p_2 от r до v. Очевидно p_1 и p_2 започват от един и същи връх r и до някакъв връх x, който може да съвпада с r, "вървят заедно" (по-академично казано, започват с общ подпът $r \cdots x$), след което се "разделят завинаги", за да стигне p_1 до u, а p_2, до v. p_1 и p_2 нямат други общи върхове освен общото си начало x. Това е така поради свойствата на арборесценциите. Нека подпътят на p_1 от x до u е q_1, а подпътят на p_2 от x до v да е q_2. Тогава в графа има (поне) два различни пътя от x до v: единият е q_2, а другият е q_1, последван от реброто (u, v). Показахме, че има два върха, такива че от първия до втория има два различни пътя.

Доказателството в другата посока Ще покажем, че ако графът не е singly connected, то съществува връх a, такъв че DFS, стартиран от a, ще открие ребро напред или ребро настрани, на което и двата края от едно и също дърво в гората на обхождането, независимо от това какви са списъците на съседство на графа; тоест, независимо от реда на обхождане на съседите. Нека графът не е singly connected. По определение, за два различни върха α и ω има повече от един път от α до ω.

Да разгледаме дървото T на обхождането при стартиране на DFS от α. T е кореново дърво–арборесценция с корен α, като в T съществува ориертиран път p от α до ω. Нека q е алтернативен път от α до ω (q е път в графа, не в дървото; в дървото има точно един път от корена до кой да е друг връх). Пътищата p и q имат общо начало и имат общ край, но не знаем дали имат общи вътрешни върхове или не. Дефинираме връх z като най-отдалеченият от α връх в p, такъв че подпътят на p от z до ω е общ с q. Може $z = \omega$, ако p и q "влизат" в ω през различни ребра, може $z \neq \omega$, ако p и q имат един и същ завършек-подпът с поне едно ребро. Важното е, че z е различен от α, инак p и q биха били един и същи път. Пътят q "влиза" в z през някое ребро e, което не е от p. Нека началото на e е връх x.

Ако x е връх от p, то реброто e е ребро напред, защото началото му x и краят му z са в отношение на предшествие в T, като x е предшественик на z.

Ако x не е връх от p, то реброто e е ребро настрани, като и двата му върха са от едно и също дърво, а именно T. А е ребро настрани, защото в T краят му и началото му не са в отношение на предшествие.

Задача 2: Припомнете си функцията EXTRACT-MAX(S), където S е двоична пирамида. Имплементацията, изучавана на лекции, има сложност по време в най-лошия случай $\Theta(\lg n)$. Предложете по-бърза, в асимптотичния смисъл, имплементация, или докажете, че такава не съществува.

Решение: Ако разполагаме с имплементация на EXTRACT-MAX(S), работеща във време $\Theta(f(n))$ за някаква функция $f(n)$, можем да конструираме HEAPSORT, работещ във време $O(nf(n))$, като първо построим пирамида S с n елемента в линейно време и после n пъти вадим максимален елемент от S с EXTRACT-MAX(S) слагаме извадения елемент на подходящо място в някакъв изходен масив (който пълним отдясно наляво).

Ако $f(n) = o(\lg(n))$, то този HEAPSORT би имал сложност $o(n \lg(n))$, което, както знаем от лекцията за долни граници, е невъзможно.

Следователно, такава имплементация на EXTRACT-MAX(S) не съществува.

Задача 3: Ползвайки обясненията на лекции и написаното в лекционните записки, дайте подробно доказателство за коректността на алгоритъма за намиране на силно свързаните компоненти на Kosaraju.

Решение: Да си припомним алгоритъма на Kosaraju.

```
KOSARAJU(Ориентиран граф G = (V, E))
1  L[1, ..., n] ← TSORT(G)
2  намери G^T
3  SCC(G^T, L)
```

Да дефинираме *силно свързана компонента-сифон* в граф като силно свързана компонента, на която във фактор-графа отговаря връх-сифон, а *силно свързана компонента-източник* в граф като силно свързана компонента, на която във фактор-графа отговаря връх-източник.

Знаем, че силно свързаните компоненти-източници в G точно отговарят (тоест, се индуцират от същите класове на разбиването, породено от силната свързаност) на силно свързаните компоненти-сифони в G^T и обратно. Това бе споменато на лекции като очевидно, понеже източниците на фактор-графа на G са точно сифоните на фактор-графа на G^T и обратното.

Ето кодът на SCC.

```
SCC(G = (V, E); L[1, ..., n]: масив от върховете)
1  S и time са глобални
2  for i ← 1 to n
3      color[i] ← white
4  time ← 0
5  for i ← 1 to n
6      if color[L[i]] = white
7          S ← ∅
8          SCC-REC(G, L[i])
9          print S
```

Лема 1: След изпълнението на ред 1 на алгоритъма на Kosaraju, върхът $L[1]$ е връх от силно свързана компонента-сифон на G^T.

Доказателство: Това твърдение е същото като твърдението, че последният връх, финализиран от TSORT, е от силно свързана компонента-източник на G. Да допуснем противното: този връх е от силно свързана компонента G' на G, която не е компонента-източник на G. Тогава в G има силно свързана компонента G'', такава че от поне един връх b от G'' има поне едно ребро към връх от G'.

Забележете обаче, че има път от b до $L[1]$, но няма път от $L[1]$ до b (и изобщо до никой връх от G''), инак G' и G'' биха били части от една и съща компонента. Това е ключовото в доказателството. Тъй като алгоритъмът открива всички върхове, има точно две възможности за това, кой ще бъде открит преди другия от $L[1]$ и b.

- Ако b бъде открит преди $L[1]$, непременно b ще бъде финализиран след $L[1]$ поради наличието на път от b до $L[1]$.
- Ако $L[1]$ бъде открит преди b, то $L[1]$ ще бъде финализиран преди b поради липсата на път от $L[1]$ до b. Тоест, пак b ще бъде финализиран след $L[1]$.

И в двата случая b бива финализиран след $L[1]$, в противоречие с допускането, че $L[1]$ е последният финализиран връх. Ерго, последното допускане, а именно, че $L[1]$ не е от компонента-източник на G, е грешно. Тогава $L[1]$ е от компонента-източник на G, което веднага влече, че е от компонента-сифон на G^T. □

Лема 2: За произволно $U \subset V$ е вярно, че TSORT$(G - U)$ връща масив, първият връх в който е от силно свързана компонента-сифон на $(G - U)^T$.

Доказателство: Това следва веднага от Лема 1. □

Теорема 1: При всяко достигане на ред 8 на SCC е вярно, че че $L[i]$ е връх от силно свързана компонента-сифон на подграфа на G^T, индуциран от белите върхове.

Доказателство: Ще докажем това по индукция по j.

База: При първото достигане на ред 8 на SCC, очевидно $i = 1$. Тогава всички върхове са бели, така че подграфът на G^T, индуциран от белите върхове, е самият G^T. Съгласно Лема 1, $L[1]$ е връх именно от силно свързана компонента-сифон на G^T.

Поддръжка: Да допуснем, че твърдението е вярно при някое достигане на ред 8 на SCC. Нека това достигане е в момента t. Тогава рекурсивната функция открива силно свързана компонента-сифон на подграфа на G^T, индуциран от тези върхове, които в момента t са бели, но не открива нищо друго, защото от тази силно свързана компонента не може да се "излезе"; за всяко ребро от неин връх към връх z от друга силно свързана компонента е вярно, че z е черен.

При излизането от рекурсивното викане на ред 8, а това става в някакъв момент $t' > t$, е вярно, че върховете на въпросната силно свързана компонента вече са черни и се намират в множеството S. Твърдим, че най-левият бял връх в този момент е от силно свързана компонента-сифон на подграфа на G^T, индуциран от върховете, които са бели в момента t'. Но това следва от Лема 2, понеже черните върхове са все едно изтрити от графа (и SCC, и SCC-REC ги прескачат). □

Задача 4: Разгледайте функцията `bar`, написана на `C`. Допуснете, че n е естествено число.

```
1  int bar(int n) {
2      int s = 1;
3      for (i = 1; i < n; i ++) {
4        for (j = 1; j <= floor(n/s); j ++) {
5          s ++;
6        }
7      }
8
9  return s;
10 }
```

bar.c

2 т.

- Какво връща тя?

23 т.

- Докажете това. Няма нужда доказателството да е много формално и прецизно, няма нужда да се ползва инварианта. Дайте смислен аргумент, който не е прекалено многословен.

Решение: Ако $n = 0$, функцията връща 1. Ако $n \in \{1, 2, 3\}$, функцията връща n. Ако $n \geq 4$, функцията връща $n + 1$.

Ето кратка аргументация. Ако $n = 0$ или $n = 1$, тялото на външния for цикъл не се изпълнява изобщо. Ерго, s остава със стойност 1, каквато е получила на ред 2.

Ако $n = 2$, тялото на външния for цикъл се изпълнява, при което тялото на вътрешния for цикъл се изпълнява само веднъж, s става 2 и вътрешният цикъл не се изпълнява повече, след това външният цикъл не се изпълнява повече и s остава 2.

Ако $n = 3$, тялото на външния for цикъл се изпълнява два пъти, като при първото изпълнение вътрешният for се изпълнява точно веднъж и s става 2, при второто изпълнение вътрешният for също се изпълнява точно веднъж и s става 3, и s остава 3.

Да допуснем, че $n \geq 4$. Броят на изпълненията на тялото на външния цикъл не зависи от s. Очевидно i приема стойностите $1, 2, \ldots, n-1$, така че външният for "се извърта" $n - 1$ пъти.

За $i = 1$, вътрешният for "се извърта" поне два пъти, защото

- при първото достигане на ред 4 булевото условие $j \leq \lfloor n/s \rfloor$ е $1 \leq \lfloor n/1 \rfloor$, което е истина, s става 2 и
- при второто достигане на ред 4 булевото условие $j \leq \lfloor n/s \rfloor$ е $2 \leq \lfloor n/2 \rfloor$, което е истина и тялото на вътрешния for се изпълнява.

Само от тези съображения е ясно, че s се инкрементира поне n пъти, с което s става $n + 1$. В момента, в който s стане $n + 1$, $\lfloor n/s \rfloor$ става нула, булевото условие на ред 4 става лъжа и тялото на вътрешния for не се изпълнява повече. Ерго, s остава $n + 1$ до края на алгоритъма.