Примерни решение на задачите от изпита по Дизайн и Анализ на Алгоритми, КН2, 29.06.2017 г.

Зад. 1 Дадени са n хижи h_1, h_2, ..., h_n. Няма телефонна връзка между нито две от тях. Компанията XYZ трябва да свърже всички хижи в телефонна мрежа чрез телефонни линии така:

- Всяка телефонна линия трябва да свързва директно точно две различни хижи.
- Казваме, че две различни хижи h_i и h_j *имат връзка* тогава и само тогава, когато са директно свързани или има редица от хижи h_{k_1}, h_{k_2}, ..., h_{k_p}, такава че h_i е директно свързана с h_{k_1}, h_{k_1} е директно свързана с h_{k_2}, и така нататък, h_{k_p} е директно свързана с h_j. Релацията "има връзка с" е симетрична – h_i има връзка с h_j тогава и само тогава, когато h_j има връзка с h_i.

  Иска се **всяка двойка** различни хижи да имат връзка.
- В заданието са описани *потенциални телефонни линии* ℓ_1, ℓ_2, ..., ℓ_t. XYZ не е длъжна да изгради всички тях, но трябва да изгражда линии **само** измежду потенциалните. Всяка потенциална линия има краища-хижи, а също така и *тегло*, което е положително число.

Подзадача 1: Тук теглата на потенциалните линии се интерпретират като цени. Предложете колкото е възможно по-ефикасен в асимптотичния смисъл алгоритъм Alg1, чийто вход е множеството от хижите и множеството от потенциалните линии, а изходът е или No, ако не е възможно хижите да бъдат свързани в телефонна мрежа, или подмножество на $\{\ell_1, \ell_2, \ldots, \ell_t\}$, което реализира телефонна мрежа с минимална цена, в противен случай. Цената на мрежата е сумата от цената на линиите в това подмножество. Обосновете отговорите си. Ако алгоритъмът, които предлагате, ползва алгоритъм, който е изучаван на лекции, не е необходимо да обосновавате коректността или сложността на изучавания алгоритъм, но се иска да кажете каква е сложността му по време.

Подзадача 2: Тук теглата на потенциалните линии се интерпретират като дължини. Като част от заданието е и някакво положително число q. Иска се алгоритъм, с който компанията XYZ да реши дали е възможно хижите да бъдат свързани в телефонна мрежа по такъв начин, че нито една телефонна линия да не е по-дълга от q. Тоест, иска се алгоритъм с еднобитов отговор.

- Докажете, че тази подзадача се решава от следния алгоритъм. Първо пускаме алгоритъма Alg1 от предишната подзадача. Ако той върне No, то връщаме No, по този начин показвайки, че няма решение. В противен случай Alg1 връща множество от телефонни линии $T = \{\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_c}\}$. Ако максималното тегло на телефонна линия в T е по-голямо от q, то връщаме No, по този начин показвайки, че няма решение. В противен случай връщаме Yes.
- Предложете алгоритъм с линейна сложност по време за тази подзадача. Обосновете отговора си.

Решение. В първата подзадача очевидно става дума за минимално покриващо дърво (МПД). За пълен отговор обаче това трябва да се обоснове.

Ако има нещо да се обосновава в тази подзадача, то е, че трябва да се намери покриващ подграф на дадения граф, който е именно минимално покриващо **дърво**. Че е зададен тегловен граф с върхове-хижите е очевидно – това следва веднага от изискването директните връзки да са само между две хижи (в реалността при изграждането на телефонна мрежа между отдалечени обекти може да има разклонителни кутии, които не са в обекти, а някъде на терена, което прави задачата по-трудна, но тук е казано, че линиите са само между хижи). Че се търси подграф също е очевидно. Че подграфът трябва да е покриващ (в него да има път между всеки два върха) идва от изискването всички хижи да са имат връзка в телефонната мрежа. Че подграфът, освен покриващ, трябва да е и дърво идва

от изискването за минимална сумарна цена и от това, че цените са положителни. За да се убедим в последното, да разгледаме възможността за отрицателни тегла. Отрицателно тегло на линия значи, че всъщност ни плащат за изграждането ѝ. Очевидно тогава всички линии с отрицателни тегла трябва да са в подграфа, независимо от това дали по този начин в подграфа има цикли. Отново: това, че трябва да се построи покриващ граф, в който няма цикли, идва от положителните тегла и условието за минимална цена; никъде в задачата не се казва експлицитно в телефонната мрежа да няма цикли.

След като сме обосновали това, че се иска МПД, забелязваме, че никъде не е казано, че потенциалните линии образуват свързан граф. Поради това не можем да ползваме директно някой от изучаваните алгоритми за МПД, защото при тях има допускане, че графът е свързан. Но съвсем лесно може да модифицираме алгоритъма на Прим или алгоритъма на Крускал по такъв начин, че да открива дали даденият граф е свързан и ако не е свързан, да индикира това с връщане на някаква особена стойност NO, а ако е свързан, да връщат МПД. Да разгледаме алгоритъма на Прим във варианта CLR, тоест с приоритетна опашка Q. В основния цикъл $\boxed{\textbf{while } \mathrm{NotEmpty}(Q) \textbf{ do}}$ вадим елемент от опашката (като по този начин го слагаме в дървото) чрез $\boxed{x \leftarrow \mathrm{ExtractMin}(Q)}$. Графът не е свързан тогава и само тогава, когато на някаква итерация на **while**-цикъла (щом сме влезли в него, значи има върхове, които са извън дървото) в опашката има само върхове с ключове, равни на ∞. И така, цялата модификация е, че ако ключът на x е ∞, алгоритъмът връща стойност NO и терминира, в противен случай продължава работата си по начина, който знаем – преглежда съседите на x един по един и т. н. Очевидно сложността по време си остава същата.

Сега да разгледаме втората подзадача. Тук пак става дума за покриващо дърво, което е минимално в някакъв смисъл, но минималността е различна от тази на МПД. Тук се теглото на дървото не е сумата от теглата на ребрата, а теглото на най-тежкото ребро. Покриващо дърво с такова минимално тегло се нарича bottleneck spanning tree (BST). В тази подзадача не се иска максимално ефикасен алгоритъм за построяване на BST. Известен е линеен алгоритъм за намиране на BST—алгоритъмът на Camerini—но той е прекалено сложен, за да бъде измислен на изпит, дори с подсказки.

Във втората подзадача се искат две неща. Първо, да се покаже, че всяко МПД всъщност е и BST (обратното **не е вярно**). И второ, да се измисли линеен алгоритъм за доста по-простата задача да се провери дали даден тегловен граф има BST с тегло, не по-голямо от някакво предварително зададено число q. Тази задача е много лесна – достатъчно е да пуснем един DFS или BFS с малката модификация да прескача ребра, които са по-тежки от q. Ако така модифицираният BFS/DFS установи, че графът е свързан, то очевидно има BST с тегло, по-малко или равно на q, а в противен случай такова BST няма. Тази модификация на BFS/DFS не променя асимптотичната сложност на алгоритъма и той си остава линеен. И така, решихме втората част на втората подзадача.

Остава да решим първата част на втората подзадача. Ще докажем, че всяко МПД е и BST. Нека T е МПД на графа. Нека $e = (u, v)$ е ребро с максимално тегло в T. Да кажем, теглото на e е d. Да допуснем, че T не е BST. Нека T' е BST в същия граф. Нека $e' = (u', v')$ е ребро с максимално тегло в T'. Да кажем, теглото на e' е d'. Щом T не е BST, то $d \neq d'$; инак, T би било BST. Веднага отхвърляме възможността $d' > d$: ако беше така, то T' не би било BST, защото има поне едно покриващо дърво, а именно T, което има по-леко най-тежко ребро, а именно e с тегло d. Остава възможността $d' < d$. Разглеждаме T. Съобразяваме, че изтриването на e от T води до появата на две дървета, едното от които съдържа връх u, а другото, връх v. Да наречем съответно тези дървета T_u и T_v. Очевидно $\{V(T_u), V(T_v)\}$ е срез в графа. Щом T' е покриващо дърво, то поне едно ребро от T' прекосява този срез. По допускане, всяко ребро на T' е по-леко от e. Тогава, ако добавим произволно ребро от T', което прекосява среза $\{V(T_u), V(T_v)\}$, към дърветата T_u и T_v, ще получим покриващо дърво със сумарно тегло, по-малко от сумарното тегло на T. Което противоречи на допускането, че T е МПД.

Зад. 2 Налага се да пътувате с лодка надолу (тоест, по течението) по някаква река. Тръгвате от пристанище P_1 и трябва да стигнете до пристанище P_n. За съжаление, не разполагате със собствена лодка. За щастие, между P_1 и P_n има междинни пристанища P_2, P_3, $\ldots$, P_{n-1}; за всеки две P_i, P_j, такива че $1 \leq i < j \leq n$, пристанището P_j е след P_i по реката. Освен това, за всеки две P_i, P_j, такива че $1 \leq i < j \leq n$, можете да наемете лодка от P_i до P_j на цена $c(i, j)$. Предложете колкото е възможно по-ефикасен алгоритъм, който намира минималната цена за пътуване с наети лодки от P_1 до P_n. Цената на цялото пътуване е сумата от цените на отделните наемания на лодки. Цените $c(i, j)$ са произволни положителни числа.

Решение. Това е задачата за най-къс път в ориентиран тегловен граф във варианта от даден връх (P_1) до друг даден връх (P_n). Съществено е, че графът е dag, защото всички възможни вземания на лодка са от дадено пристанище до друго пристанище, което е надолу по реката. Иска се само цената на най-евтин път, а не самият път. Линеен алгоритъм за тази задача е разглеждан на лекции. Забележете, че в тази задача dag-ът има точно един източник, а именно P_1, и точно един сифон, а именно P_n, така че неизбежно началният връх след топологическото сортиране ще е P_1, а крайният ще е P_n.

DAG-Shortest-Path(G: dag, P_1: unique source, P_n: unique sink, w: weight function)
1 TopoSort(G)
2 **for** $v \in V(G)$
3 $\quad d[v] \leftarrow \infty$
4 $d[P_1] \leftarrow 0$
5 **for** $x \in V(G) \setminus \{P_1\}$ in the topo-order
6 $\quad d[x] \leftarrow \min\{d[y] + w(y,x) \mid (y,x) \in E(G)\}$
7 **return** $d[P_n]$

Зад. 3 При дадени три стринга $x = x_1x_2\cdots x_m$, $y = y_1y_2\cdots y_n$ и $z = z_1z_2\cdots z_{m+n}$ над някаква азбука Σ, казваме, че z е *смесване на* x *и* y тогава и само тогава, когато z се състои точно от буквите на x и y с единственото ограничение, че буквите на x в z са в същото взаимно разположение, както в x, и буквите на y в z са в същото взаимно разположение, както в y.

Ето пример. Нека $\Sigma = \{0,1\}$, нека $x = \mathbf{1100111}$ и нека $y = 001001$. Тогава $z_1 = \mathbf{1100}001\mathbf{11}0011$ е смесване на x и y. За Ваше улеснение, стринговете x и y са написани с различни цветове и в различни шрифтове, като тези атрибути са запазени в z_1, за да е ясно "кой откъде идва". От друга страна обаче, $z_2 = 1110011001001$ не е смесване на x и y, въпреки че съдържа точно 7 единици и 6 нули.

Задачата е, предложете ефикасен алгоритъм, който при вход стрингове $x = x_1x_2\cdots x_m$, $y = y_1y_2\cdots y_n$ и $z = z_1z_2\cdots z_{m+n}$ над $\Sigma = \{0,1\}$, да връща Y, ако z е смесване на x и y, или N, в противен случай. Обосновете накратко алгоритъма и изследвайте сложността му по време и памет.

Решение. Общият вид на подзадачите е: за $0 \le i \le m$ и $0 \le j \le n$, решете дали $z_1z_2\cdots z_{i+j}$ е смесване на $x_1x_2\cdots x_i$ и $y_1y_2\cdots y_j$. При $i = 0$ или $j = 0$ очевидно става дума за празния стринг. Удобно е да приемем, че празният стринг участва във всяко смесване. Тези подзадачи са само полиномиално много на брой, а именно $(m+1)(n+1)$, така че ако можем да пресмятаме по-големите подзадачи от по-малките ефикасно, ще имаме полиномиален алгоритъм, изграден по схемата *динамично програмиране*. Нека $T[i,j]$ е истина тогава и само тогава, когато $z_1z_2\cdots z_{i+j}$ е смесване на $x_1x_2\cdots x_i$ и $y_1y_2\cdots y_j$. Следното рекурентно уравнение е очевидно вярно и то ни дава директно ефикасен алгоритъм:

$$T[i,j] = \begin{cases} \text{True}, & \text{if } i = 0 \wedge j = 0 \\ \text{False}, & \text{if } x_i \neq z_{i+j} \wedge y_j \neq z_{i+j} \\ T[i-1,j], & \text{if } x_i = z_{i+j} \wedge y_j \neq z_{i+j} \\ T[i,j-1], & \text{if } x_i \neq z_{i+j} \wedge y_j = z_{i+j} \\ T[i-1,j] \vee T[i,j-1], & \text{if } x_i = z_{i+j} \wedge y_j = z_{i+j} \end{cases}$$

Тъй като всяка следваща подзадача се решава в $\Theta(1)$ от вече решени подзадачи, то сложността по време е асимптотично същата като размира на таблицата. И така, сложността по време и по памет е $\Theta(mn)$. Таблицата може да се попълва по редове. След попълването на таблицата, алгоритъмът връща $T[m,n]$.

---

15 т. **Зад. 4** Всяко от числата $1, 2, \ldots, 2n$ е произволно оцветено или в бяло, или в черно, при единственото ограничение, че n от числата са оцветени в бяло, а останалите, в черно. Трябва да разбием множеството $\{1,2,\ldots,2n\}$ на n подмножества $A_1, A_2, \ldots, A_n$, такива че:

- За всяко i, такова че $1 \le i \le n$: $|A_i| = 2$.

- За всяко i, такова че $1 \leq i \leq n$: в A_i съществува бял елемент и в A_i съществува черен елемент.
- Сумата

$$\sum_{i=1}^{n} \sum_{\substack{x,y \in A_i \\ x \neq y}} |x - y|$$

е минимална.

Предложете алгоритъм с линейна сложност по време, който генерира такова разбиване, и обосновете коректността му. Не е необходимо подробно доказателство на коректност (от типа доказателство с инвариант). Задачата е оптимизационна, затова трябва просто да покажете, че Вашият алгоритъм връща решение, което е оптимално.

Решение. Условието на задачата е изложено твърде формално и многословно – нарочно, разбира се. Задачата може да се формулира по-просто: дадени са n бели и n точки (dots), наредени в редица, така че съседите са на единица разстояние един от друг, и трябва да свържем точки по двойки бяла с черна (очевидно има точно n такива двойки) с жици (wires), така че общата дължина на използваните жици да е минимална.

Задачата е добре известна и е добре известно нейно решение с greedy стратегия, която свързва i-тата бяла с i-тата черна точка. Очевидно това е алгоритъм с линейна сложност по време и константна сложност по памет, така че единственото, което трябва да обосновем, е оптималността на решението, което той намира.

Нека най-лявата бяла точка е w_1, следващата бяла точка отляво надясна е w_2, и така нататък, и най-дясната бяла точка е w_n. Напълно аналогично, нека черните точки отляво надясно са b_1, b_2, $\ldots$, b_n. За всяка точка дефинираме нейната *съответна* така: ако точката е w_i, нейната съответна е b_i, и обратното. Твърдението е, че свързването на съответните точки по двойки е оптимално.

Да допуснем, че това не е така. Тогава съществува поне един пример P, върху който поне една двойка съответни точки не са свързани (очевидно такава двойка не може да е единствена, но да не избързваме) и който има по-малко тегло от свързването на съответните точки. Да разгледаме най-лявата точка в P, която не е свързана със съответната си точка. Без ограничение на общността, нека тази точка е черна. Нека я наречем b_i. По допускане, b_i не е свързана с w_i. Тогава b_i е свързана с някоя w_j. Ще покажем, че w_j е вдясно от w_i. Всички $b_1, \ldots, b_{i-1}$ са свързани със своите съответни $w_1, \ldots, w_{i-1}$. Това, че $b_1, \ldots, b_{i-1}$ са вляво от b_i е от дефинирането на b_i, но също така е вярно, че $w_1, \ldots, w_{i-1}$ са вляво от w_i – инак, те нямаше да са съответни на $b_1, \ldots, b_{i-1}$. Тогава $w_1, \ldots, w_{i-1}$ са свързани със своите съответни и тъй като w_j не е свързана със своята съответна, то w_j не е измежду $w_1, \ldots, w_{i-1}$. Тогава w_j е е вдясно от w_i и взаимното разположение е:

$$\cdots\cdots b_i \cdots\cdots w_i \cdots\cdots w_j \cdots\cdots$$

Но w_i е свързана с някоя точка b_k. Четирите точки b_i, w_j, w_i и b_k допринасят към общото тегло на решението $|b_i - w_j| + |b_k - w_i|$.

Следните три възможности за b_k са изчерпателни. Първо, b_k може да е вдясно от w_j:

$$\cdots\cdots b_i \cdots\cdots w_i \cdots\cdots w_j \cdots\cdots b_k \cdots\cdots$$

Тогава очевидно можем да свържем b_i с w_i и w_j с b_k, като при това приносът на четирите точки ще е $|b_i - w_i| + |b_k - w_j| < |b_i - w_j| + |b_k - w_i|$. Следва, че P не е оптимално.

Второ, може b_k да е между w_i и w_j:

$$\cdots\cdots b_i \cdots\cdots w_i \cdots\cdots b_k \cdots\cdots w_j \cdots\cdots$$

Тогава отново можем да свържем b_i с w_i и w_j с b_k, като при това приносът на четирите точки ще е $|b_i - w_i| + |b_k - w_j| < |b_i - w_j| + |b_k - w_i|$. Пак следва, че P не е оптимално.

Трето, може b_k да е между b_i и w_i:

$$\cdots\cdots b_i \cdots\cdots b_k \cdots\cdots w_i \cdots\cdots w_j \cdots\cdots$$

В този случай, ако свържем b_i с w_i и b_k с w_j, ще получим свързване, което има същото тегло като P, понеже в този случай $|b_i - w_i| + |b_k - w_j| = |b_i - w_j| + |b_k - w_i|$.

И така, видяхме, че или P не е оптимално, или има свързване със същото тегло като P, при което b_i е свързано с w_i. Очевидно, ако продължим по този начин – намирайки следващата точка отляво надясно, която не е свързана със своята съответна, и прилагайки аналогична конструкция, или ще покажем, че P не е оптимално, или ще намерим свързане със същото тегло като P, но в което са свързани само съответни двойки точки.

Зад. 5 Професор Оптимистов твърди, че задачата за намиране на минимално върхово покриване (Vertex Cover) на неориентиран граф е в класа на сложност **P**. Професор Песимистова твърди, че задачата за намиране на максимално независимо множество върхове (Independent Set) в неориентиран граф не е в класа на сложност **P**, но е в класа на сложност **NP**.

Накратко обяснете какво са **P** и **NP**. Допуснете, че $\mathbf{P} \neq \mathbf{NP}$ и кажете дали и двамата професори може да са прави? Обосновете отговорите си съвсем кратко. За да получите пълен брой точки не е необходимо да излагате резултати за сложността на тези две изчислителни задачи, които резултати не са изучавани на лекции.

Решение. Няма как и двамата да са прави. Задачите Vertex Cover и Independent Set се трансформират една в друга с полиномиални трансформации, и това е съвсем тривиално: върху един и същи граф, допълнението на всяко независимо множество е върхово покриване, и обратното. Това се доказва елементарно от дефинициите на "независимо множество" и "върхово покриване". И така, ако $(G = (V, E), k)$ е пример на Vertex Cover, то $(G = (V, E), |V| - k)$ е пример на Independent Set и обратното. Тогава, ако едната от тези задачи е в **P**, то и другата задължително ще е в **P**.