Малко контролно №2 по ДАА
7 юни 2017г.

Зад.1. (2т.) Разполагаме с n на брой правоъгълни кутийки с разнообразни широчини и дължини. Можем да поставяме кутийка в някоя друга само ако широчината и дължината на първата кутийка са строго по-малки съответно от широчината и дължината на втората кутийка. „Въртене“ на кутийките е забранено и очевидно можем да имаме кутийки, които няма как да поставим една в друга. Опишете алгоритъм, който за $O(n^2)$ време намира колко най-много кутийки от дадените можем да поставим последователно „като матрьошки“ една в друга.

Зад.2 (1,5т.) Отворили сме празен текстови файл и имаме следните опции за работа с него (всяка от тях се счита за едно *действие*):

- Добавяне на един символ, без значение какъв;
- Маркиране на цялото съдържание на файла (Ctrl-A);
- Копиране на всичко маркирано (Ctrl-C, имаме неограничен буфер);
- Поставяне на копираното (Ctrl-V) – след това буферът остава празен!

Естествено, за да поставим някакво съдържание, трябва преди това да сме го маркирали и копирали в буфера. Предложете алгоритъм, който за $O(n)$ време изчислява колко най-много символа може да съдържа файлът след n такива действия.

Зад.3 (2т.) Нека имаме неориентиран свързан тегловен граф с n върха и m ребра, в който теглото на всяко ребро е точно едно измежду реалните числа a и b (нека $0 < a < b$). Предложете *линеен* алгоритъм, който изчислява теглото W на минимално покриващо дърво за графа (не е задължително да намирате такова дърво). За по-бавен алгоритъм ще получите само **1т**.

Упътване: ако премахнем по-тежките ребра от графа, ще получим евентуално несвързан граф. Каква е връзката между броя на неговите свързани компоненти и W?

➔ Обърнете на задната страна за бонус задачите!

Бонус1 (1,5т.) Ами ако дължината на кутийките няма значение при сравненията, т.е. сравняваме само по тяхната широчина? Предложете $O(nlgn)$ алгоритъм, който отговаря на същия въпрос при тази промяна в условието.

Бонус2 (1т.) Какво би се променило със **Зад.2**, ако след поставяне буферът не остава празен, т.е. можем няколко пъти подред да натискаме Ctrl-V? Предложете квадратичен алгоритъм, който взима предвид тази промяна в условието.

Можете ли да решите двата варианта на тази задача *по-бързо*?

Решения (с обяснения):

Зад.1. Виждаме, че имаме описана релация между две кутийки „едната кутийка може да се постави в другата“. Тази релация не е симетрична (и очевидно е бинарна), което означава, че можем да моделираме задачата по следния начин: ще построим граф, в който на всяка кутийка ще съответства точно един връх, а ребро от връх u към връх v ще има само ако кутийката, съответстваща на u, може да се „постави“ по тази дефиниция в кутийката, съответстваща на v. Поради гореспоменатите свойства на тази релация този граф ще бъде ориентиран и ацикличен (няма нужда от доказателство, но ако Ви съмнява, проверете го сами). В задачата се изисква да намерим възможно най-големия брой кутийки, които да можем да поставяме последователно една в друга. Това съответства на намирането на най-дълъг път в построения граф ($LPDAG$) . Дължината на този път в брой върхове ще е търсеният максимален брой кутийки.

Тук можем да спрем и да не пишем алгоритъма, тъй като за целите на това контролно го считаме за „популярен“ и „известен“, но все пак не е лошо да го направим:

$LPDAG(G(V,E): directed\ acyclic\ graph)$
1. $A \leftarrow TopoSort(G)$
2. $M[1..n] \leftarrow array\ of\ integers$
3. $foreach\ u\ in\ reverse\ order\ of\ A$
4. $\quad M[u] \leftarrow 1$
5. $\quad foreach\ v\ in\ Adj(u)$
6. $\quad\quad M[u] \leftarrow \max\{M[u], M[v]+1\}$
7. $return\ Maximum(M[1..n])$

Построението на графа отнема $O(n^2)$ време, тъй като сме длъжни да проверим за всеки две кутийки дали едната не се влага в другата, т.е. дали не трябва в графа да има ребро между всеки два върха. Ясно е, че ребрата ще бъдат $m = O(n^2)$ на брой. След това алгоритъмът за топологическо сортиране отнема $O(n+m)$ време, и допълнителната работа също толкова, което води до обща сложност по време $T(n) = O(n^2) + O(n+m) + O(n+m) = O(n^2)$ и по памет $M(n) = O(n+m) + O(n) + O(n) = O(n+m) = O(n^2)$. Допълнителната памет идва от съхранението на самия граф с най-много $O(n^2)$-та си ребра, паметта необходима на $TopoSort$, и паметта за масива M.

Важно е да се отбележи, че задачата НЕ Е за най-дълга нарастваща подредица, тъй като никъде не е указано, че можем да поставяме кутийка с индекс i в кутийка с индекс j само ако $i < j$, тоест само кутийка по-рано в масива в друга след нея. Ако беше указана такава наредба, тогава получаваме $LongestIncreasingSubsequence$, което се решава с подобен алгоритъм, но в случая това би било грешно решение.

Зад.2. Знаем по условие, че преди действие Ctrl-V трябва задължително да има Ctrl-C, а преди него задължително Ctrl-A. Тогава, за всяко действие имаме четири възможности: натискане на един символ, натискане само на Ctrl-A, натискане на Ctrl-C веднага след Ctrl-A или Ctrl-V след Ctrl-C след Ctrl-A. Тези последователни натискания се броят към действията, следователно за максималния брой символи след n действия имаме следната рекурентна връзка:

$$f(n) = \max\{f(n-1)+1, f(n-1), f(n-2), 2*f(n-3)\}$$

Четирите случая съответстват на четирите възможни избора за „последно“ действие. Очевидно Ctrl-A и Ctrl-C не увеличават броя символи, а чак след три действия се поставя копираното, тоест съдържанието се удвоява. Можем да мемоизираме това изчисление в едномерен масив, индексиран от 1 до n, който да запълним по следния начин:

$CopyPaste(n: positive\ integer)$
1. $M[1..n] \leftarrow array\ of\ integers$
2. $M[1] \leftarrow 1, M[2] \leftarrow 2, M[3] \leftarrow 3$
3. $for\ i\ \leftarrow 4\ to\ n$
4. $\quad M[i] \leftarrow \max\{M[i-1]+1, M[i-1], M[i-2], 2*M[i-3]\}$
5. $return\ M[n]$

Мемоизираме наивната рекурентна връзка по наивен начин, което води до прост и работещ алгоритъм в $T(n) = M(n) = O(n)$. Първите три стойности в масива можем да изчислим на ръка. По-нататък бихме могли да оптизираме изразходваната памет, т.к. изчислението за всяка позиция в масива използва само предишните три. Тогава ще са ни достатъчни 4 локални променливи, за да държим на всяка итерация текущата стойност и предишните 3, и накрая да върнем същия резултат. Ще получим решение с $M(n) = O(1)$.

Не е трудно човек да забележи, че от гореизброените четири случая само първият и последният са смислени. Иначе казано, няма смисъл да натискаме Ctrl-A без след това да натиснем Ctrl-C и после Ctrl-V (независимо дали маркираното се размаркира и т.н.). Следователно можем да опростим $f(n)$ до $f(n) = \max\{f(n-1)+1, 2*f(n-3)\}$ и ред 4 на $M[i] \leftarrow \max\{M[i-1]+1, 2*M[i-3]\}$.

Зад.3. За тази задача могат да се предложат две различни, но коректни решения. Важат стандартните означения – графът е $G(V,E)$ и $n = |V|;\ m = |E|$:

- **Решение първо:** Следвайки упътването, можем да направим следните наблюдения. Нека построим само графа G', индуциран от по-леките ребра. Тогава, за да направим покриващо дърво на целия оригинален граф, можем първо да направим минималната покриващата *гора* на G' (МПД за всяка негова компонента – както казахме, G' може да не е свързан и да няма минимално покриващо *дърво*). Забележете, че понеже всички ребра в G' са с едно и също тегло, всяко покриващо дърво е минимално, но това не е нужно за решението. След това, за да свържем тези дървета, съответстващи на компонентите на G', е необходимо да използваме от тежките ребра. Разковничето тук е, че за да свържем k на брой компоненти, ни трябват *точно* $k-1$ тежки ребра (представете си k върха, които трябва да обединим в дърво). Няма как с по-малко тежки ребра, защото е възможно някоя компонента да не бъде свързана с общото покриващото дърво. Това значи, че в МПД на G със сигурност ще има точно $k-1$ от „тежките ребра“. Всички ребра трябва да са $n-1$ на брой, откъдето $\boldsymbol{W = (n-k)*a + (k-1)*b}$. Задачата се свежда до това да построим графа G' и да намерим броя негови свързани компоненти (k във формулата), откъдето можем да получим теглото. Построението се извършва за $T(n,m) = M(n,m) = O(n+m)$, а търсенето на свързаните компоненти е възможно с DFS или BFS със сложност $T(n,m) = O(n+m),\ M(n,m) = O(n)$. Окончателно, $T(n,m) = M(n,m) = O(n+m)$.
- **Решение второ:** Можем да използваме алгоритъма на Крускал със следната модификация: вместо да сортираме ребрата за $O(nlgn)$, можем да се възползваме от това, че са само две

възможни стойности и да приложим или $CountingSort$, или $Partition$ (от $QuickSort$). И двата варианта ще отнемат $O(m)$ време, а оттам нататък алгоритъмът продължава работа по нормален начин. След като получим дървото, можем просто да съберем теглата на всички ребра в него, намирайки търсеното W. Сложността по време ще е $T(n,m) = O(m + n.\alpha(n))$, и можем тънко да се възползваме от свойствата на функцията α и да заключим, че това е „линейно“ време. Факт е, че за $\forall n \leq$ от порядъка на $2^{2^{2^{16}}}$ $\alpha(n) \leq 4$, а $2^{2^{2^{16}}}$ е мнооого голямо число и за целите на това контролно този алгоритъм се признава за линеен. Сложността по памет е $M(n,m) = O(n+m)$: n заради Disjoint-Set структурата и m за масива с ребра, който сортираме.

Бонус1: Ако проверката дали някоя кутийка може да се постави в друга е само по широчините на кутийките, това означава, че за всеки две кутийки можем да поставим едната от тях в другата, стига широчините им да не са точно равни (ще са или $<$, или $>$). Тогава можем да започнем от кутийката с най-малка широчина към тази с най-голяма, поставяйки всяка кутийка в следващата по широчина, и пропускайки единствено тези кутийки, които съвпадат по широчина. Алгоритъм, който намира търсения брой кутийки, би изглеждал така:

$Boxes(A[1..n]: array\ of$ кутийки$)$
$1. Sort(A[1..n])$
$2. counter \leftarrow 1$
$3. for\ i \leftarrow 2\ to\ n$
$4.\quad if\ A[i-1].width \neq A[i].width$
$5.\quad\quad counter \leftarrow counter + 1$
$6. return\ counter$

Естествено, сортирането на ред 1 е по широчината на кутийките в масива. Не е ограничение да приемем, че можем да достъпваме тяхната широчина и дължина така, както на ред 4. Можем да приемем, че използваме $HeapSort$ за сортиращ алгоритъм и получаваме $T(n) = O(nlgn)$.

Бонус2: Тази модификация в условието означава, че трябва да разглеждаме повече случаи в рекурентното отношение за максималния брой символи, получени след n действия. Ако досега имахме само две възможности за „последните“ действия, а именно или едно натискане на символ, или комбинация Ctrl-A, Ctrl-C, Ctrl-V, сега може тези n действия да са завършили с Ctrl-A, Ctrl-C, Ctrl-V, Ctrl-V например. С това бихме получили 3 пъти повече символи, отколкото сме имали преди 4 действия. Ако имаме три последователни копирания, значи ще получим 4 пъти повече символи, отколкото сме имали преди 5 действия и така нататък за всеки възможен брой последователни копирания. Рекурентната връзка би изглеждала така:

$$f(n) = \max\{f(n-1)+1, 2*f(n-3), 3*f(n-4), \dots, (n-2)*f(1)\}$$

Тези възможности са $O(n)$ на брой за всяко n и можем да ги изброим с прост цикъл:

$CopyPasteN(n: positive\ integer)$
$1. M[1..n] \leftarrow array\ of\ integers$
$2. M[1] \leftarrow 1, M[2] \leftarrow 2, M[3] \leftarrow 3$
$3. for\ i \leftarrow 3\ to\ n$
$4.\quad M[i] \leftarrow M[i-1] + 1$
$5.\quad for\ k \leftarrow 2\ to\ i-2$

$$6. \quad M[i] \leftarrow \max\{M[i], k * M[i-k-1]\}$$
$$7. return\ M[n]$$

Очевидно вложеният цикъл на ред 6 изброява всички възможности в горната рекурентна връзка, и всичко работи за $O(n^2)$ време. Няма как да не използваме целия масив и оставаме с $O(n)$ сложност по памет.

Бонус към бонуса (дали двата варианта могат да се решат по-бързо): За първия, при позволено еднократно натискане на Ctrl-V, можем да направим следното наблюдение: за $n \leq 5$ оптималната стратегия е да добавим n пъти по един символ, получавайки n символа. За по-големи стойности оптималната стратегия е да добавим *няколко* символа в началото, след което да натискаме само Ctrl-A, Ctrl-C, Ctrl-V, Ctrl-A, Ctrl-C, Ctrl-V, и т.н. Очевидно така за всеки 3 действия си удвояваме броя символи, което води до горе-долу експоненциален брой символи. Остава да прецизираме какво имаме предвид под *няколко* символа – между 3 и 5, в зависимост какъв остатък при деление на 3 дава n. Останалите действия ще бъдат само $\approx \left\lfloor \frac{n}{3} \right\rfloor$ повторения на схемата Ctrl-A, Ctrl-C, Ctrl-V. Всяко едно от тях ще ни удвои броя символи, значи ще получим $\approx 2^{\left\lfloor \frac{n}{3} \right\rfloor}$ символа. По-точно:

$$CopyPasteFast(n: positive\ integer)$$
$$1. if\ n < 6$$
$$2. \quad return\ n$$
$$3. k \leftarrow 3 + n(mod 3)$$
$$4. return\ k * 2^{\frac{n-k}{3}}$$

Бързото на този метод е, че повдигането на степен може да се извърши в $O(lgn)$ време по метода на бързото експоненциране, или ако приемем побитовите операции (bitwise left shift, все пак смятаме степени на двойката) за константни, в $\boldsymbol{T(n) = O(1)}$.

Аналогично разсъждение може да се приложи и за втория вариант – не е оптимално да се повтарят повече от 5 или 6 (не съм сигурен за константата) последователни Ctrl-V. Това значи, че възможностите в рекурентната връзка пак стават константен брой и можем да изчислим бройката в линейно време и константна памет. Но това вече са още по-дълбоки разсъждения и изчисления. ☺