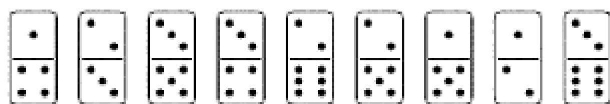
ИЗПИТ ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ" — ЗАДАЧИ ЗА СТУДЕНТИТЕ ОТ СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 1. ПОТОК, (СОФИЙСКИ УНИВЕРСИТЕТ "СВ. КЛИМЕНТ ОХРИДСКИ", ФМИ, 12. 06. 2019 Г.)

Задача 1. Масивът от цели числа $A[1...n]$ се нарича вълнист, ако подмасивът, образуван от първите му $\lceil n/2 \rceil$ елемента, е сортиран (тоест те са наредени в ненамаляващ ред), а подмасивът от останалите елементи е вълнист. За дъно на тази рекурсивна дефиниция служи празният масив ($n = 0$): той се приема за вълнист по определение. Например масивът 10, 20, 30, 40, 17, 39, 28, 14 е вълнист.

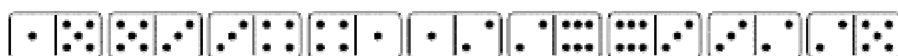
а) Опишете на псевдокод алгоритъм, сортиращ вълнист масив за линейно време. **(6 точки)**
Изпълнете алгоритъма докрай върху дадения примерен масив. **(4 точки)**
Докажете коректността на алгоритъма. **(4 точки)**
Анализирайте времевата сложност на алгоритъма в най-лошия случай. **(6 точки)**

б) Разглеждаме алгоритмичната задача `Вълнист масив:` даден числов масив $A[1...n]$ да се пренареди така, че да стане вълнист. Покажете, че при най-лоши входни данни тази алгоритмична задача има времева сложност $\Theta(n \log n)$, ако се решава чрез сравнения:
— Съставете и опишете с думи алгоритъм, който я решава за време $O(n \log n)$. **(5 точки)**
— Докажете долната граница $\Omega(n \log n)$ чрез подходяща редукция. **(15 точки)**

Задача 2. Дадени са няколко плочки от домино. Комплектът може да бъде нестандартен: допуска се някое от полетата на плочките да съдържа повече от девет точки.



Съставете алгоритъм, който за полиномиално време нарежда дадените плочки в редица (не е задължително краят на редицата да съвпада с нейното начало). Трябва да се спазват правилата на доминото: съседните плочки да се допират по полета с равен брой точки.



Не съставяйте алгоритъм "от нулата"! Вместо това използвайте някой известен алгоритъм. За целта сведете задачата за плочките до изучена задача от теорията на алгоритмите. Не се изисква анализ на сложността на алгоритъма. **(20 точки)**

Задача 3. Предложете алгоритъм, който за време $O(n^3)$ разбива даден непразен низ $S[1...n]$ от n знака на възможно най-малък брой палиндроми. Палиндром (или огледален низ) се нарича всеки текст, който се чете по един и същи начин отляво надясно и отдясно наляво. Например низът S = *ръка* се разбива на четири палиндрома — *р*, *ъ*, *к*, *а* (по-малък брой е невъзможен в случая). Низът S = *капак* се разбива на единствен палиндром — *капак*. Низът S = *мама* се разбива най-малко на два палиндрома — *мам*, *а*.

а) Опишете алгоритъма възможно най-ясно по избран от Вас начин. **(15 точки)**
б) Анализирайте времевата сложност на алгоритъма в най-лошия случай. **(5 точки)**

Задача 4. Колко е средната времева сложност на сортирането чрез вмъкване? **(20 точки)**

РЕШЕНИЯ

Задача 1 (предложена от Стефан Фотев).

а) Вълнист масив може да се сортира за линейно време по следния начин:

```
Сортиране на вълнист масив(A[1...n]: вълнист масив)
if n > 1
   m ← ⌈n/2⌉
   Сортиране на вълнист масив(A[m+1...n])
   B[1...m] ← копие на масива A[1...m]
   C[1...n-m] ← копие на масива A[m+1...n]
   A[1...n] ← Сливане на сортирани масиви(B[1...m],C[1...n-m])
```

Пример: Върху масива 10, 20, 30, 40, 17, 39, 28, 14 алгоритъмът работи така:
1) Сортира рекурсивно дясната половина. Резултат: 10, 20, 30, 40, 14, 17, 28, 39.
2) После слива двата сортирани подмасива 10, 20, 30, 40 и 14, 17, 28, 39.
Резултат: сортираният масив 10, 14, 17, 20, 28, 30, 39, 40.

Коректността на алгоритъма се доказва чрез силна индукция по дължината на масива.

Базата на индукцията е, когато дължината на масива е 0 или 1. В тези случаи алгоритъмът не прави нищо, което е правилно: празният масив и масивите с един елемент са сортирани поначало.

Индуктивна стъпка: От определението за вълнист масив следва, че подмасивът $A[1...m]$, а значи и масивът $B[1...m]$, е сортиран. Отново по определение подмасивът $A[m+1...n]$ е вълнист, следователно рекурсивното извикване работи върху допустими входни данни. Те представляват масив с по-малка дължина, затова от индуктивното предположение следва, че рекурсивното извикване ще се изпълни коректно, тоест масивът $C[1...n-m]$ ще бъде сортиран. Най-сетне, сливането на два сортирани масива по известния алгоритъм дава пак сортиран масив. Затова в крайна сметка се получава сортиран масив $A[1...n]$.

Анализ на сложността на алгоритъма: Всички случаи за входните данни са еднакво лоши. Времевата сложност $\mathrm{T}(n)$ на алгоритъма удовлетворява рекурентното уравнение

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2}\right) + \Theta(n).$$

Първото събираемо в дясната страна на уравнението е времето за рекурсивното извикване, а второто събираемо е времето за сливане на двата сортирани масива. Решаваме уравнението с помощта на мастър-теоремата и намираме $\mathrm{T}(n) = \Theta(n)$. Тоест алгоритъмът има линейна времева сложност.

б) Алгоритмичната задача *Вълнист масив* може да се реши за време $\mathrm{O}(n \log n)$ например чрез пирамидално сортиране. Получава се сортиран, следователно вълнист масив.

От друга страна, задачата `Вълнист масив` изисква време $\Omega(n \log n)$, което се доказва с редукция от задачата за сортиране на произволен числов масив (не непременно вълнист).

```
Сортиране(A[1...n]: числов масив)
1) Вълнист масив(A[1...n])
2) Сортиране на вълнист масив(A[1...n])
```

Първата команда създава вълнист масив, а втората го сортира. Видяхме, че втората стъпка (сортирането на вълнист масив) може да се извърши за време $O(n) = o(n \log n)$. Нека $T(n)$ е времевата сложност в най-лошия случай на някой алгоритъм за създаване на вълнист масив (първата стъпка). Следователно времевата сложност на получения алгоритъм за сортиране в най-лошия случай е равна на $T(n) + o(n \log n)$. От теорията е известно, че всеки алгоритъм за сортиране, основан на сравнения, има времева сложност в най-лошия случай $\Omega(n \log n)$. Следователно $T(n) + o(n \log n) = \Omega(n \log n)$, откъдето $T(n) = \Omega(n \log n)$. Понеже алгоритъмът от стъпка 1 е произволен, то времевата сложност на задачата `Вълнист масив` е $\Omega(n \log n)$.

Задача 2. Представяме входните данни чрез граф. Всяка плочка от доминото има две полета (квадратчета). Всяко поле съдържа някакви точки (може и нула). Върховете на графа са различните точки, тоест различните стойности на полетата. Ребрата на графа съответстват на дадените плочки. Редицата от плочки, изградена по правилата на доминото, съответства на път в графа, който минава по всяко ребро точно веднъж. Това е т. нар. ойлеров път, който може да бъде и цикъл, ако краят и началото му съвпадат. Намирането на ойлеров път / цикъл е задача от клас P, защото за нея има полиномиален алгоритъм, изучен в теорията: графът се разбива на цикли, които след това се навързват в един голям път / цикъл чрез допирните си точки. Ако искаме само да проверим дали има ойлеров път / цикъл, но построението му не ни интересува, тогава е достатъчно да видим дали графът е свързан и притежава не повече от два върха от нечетна степен. Това става с едно обхождане на графа (което изисква линейно, следователно полиномиално време).

Смисъл на условието, че комплектът от плочки може да бъде нестандартен (да съдържа полета с повече от девет точки): без това условие има само краен (и малък) брой плочки. Следователно дължината на входа не може да расте неограничено и асимптотичните оценки на сложността не са добре определени.

Задача 3 може да се реши по различни начини, но всички те са варианти на една идея: да се направи синтактичен анализ на дадения низ $S[1...n]$. Единият начин е да използваме алгоритъма CYK наготово: изпълняваме го върху съставена от нас безконтекстна граматика в нормална форма на Чомски. Другият начин е да използваме само идеята на алгоритъма, като го пренапишем и приспособим за конкретната задача. Ще покажем и двата начина. Първият от тях съдържа тънкост, без която задачата не може да се реши! Причината е, че тя е оптимизационна задача, а не задача за разпознаване.

Първи начин: чрез алгоритъма CYK, използван наготово. Тъй като всеки низ може да бъде разбит на палиндроми (в краен случай те ще се състоят от по един знак), то съставяме граматика, която поражда всички низове над избраната азбука. В конкретната задача е важно не разпознаването на низовете (граматиката приема всички низове), а тяхното разбиване на възможно най-малък брой палиндроми. За простота приемаме, че даденият низ се състои само от малките латински букви a и b. Една възможна граматика:

$$
\begin{aligned}
S &\to P \\
S &\to PS \\
P &\to a \\
P &\to b \\
P &\to aa \\
P &\to bb \\
P &\to aPa \\
P &\to bPb
\end{aligned}
$$

Началният знак S означава произволен низ. Нетерминалът P означава непразен палиндром. Първите две правила казват, че всеки текст е или един непразен палиндром, или редица от два или повече непразни палиндрома (тоест $S = PPP... P$). Следващите четири правила изброяват палиндромите с една или две букви. Последните две правила казват, че един низ с три или повече букви е палиндром, ако и само ако първата и последната буква на низа са равни и след изтриването им остава непразен палиндром.

Тази граматика е безконтекстна, обаче не е в нормална форма на Чомски: първото правило и последните четири правила нарушават изискванията. Затова преобразуваме граматиката:

$$
\begin{aligned}
P &\to a \\
P &\to b \\
A &\to a \\
B &\to b \\
P &\to AA \\
P &\to BB \\
P &\to CA \\
P &\to DB \\
C &\to AP \\
D &\to BP \\
S &\to a \\
S &\to b \\
S &\to AA \\
S &\to BB \\
S &\to CA \\
S &\to DB \\
S &\to PS
\end{aligned}
$$

Правилото $S \to PS$ е ключово: броят на палиндромите, на които алгоритъмът CYK разбива дадения низ, е с единица по-голям от броя на прилаганията на това правило по време на синтактичния анализ на низа. Следователно трябва да направим така, че алгоритъмът да прилага това правило възможно най-рядко. Ако алгоритъмът CYK опитва правилата в реда на тяхното срещане, то трябва да поставим споменатото правило на последно място.

От теорията е известно, че времевата сложност на алгоритъма CYK в най-лошия случай е $\Theta(|G| \,.\, n^3)$, където $|G|$ е размерът на граматиката. В нашата задача граматиката и размерът ѝ са фиксирани, така че сложността на алгоритъма е $\Theta(n^3)$.

Втори начин: без да използваме алгоритъма CYK наготово. Съставяме подобен алгоритъм по схемата ***динамично програмиране.*** Обхождаме дадения низ отзад напред и проверяваме всяка негова наставка дали е палиндром. Ако да, тя е последният член на едно възможно разбиване на дадения низ. Останалата част от низа трябва да се разбие на най-малък брой палиндроми. Това е същата задача, но за по-къс низ. Нейното решение, намерено по-рано, се взима от динамичната таблица. От всички наставки палиндроми се взима онази, за която общият брой членове на разбиването е най-малък. Този алгоритъм работи над всяка азбука.

Псевдокод на алгоритъма:

```
Palindromes(S[1...n]: низ)
Dyn[0...n], Last[1...n]: масиви от цели числа;
// Dyn[k] = най-малкият брой палиндроми, на които се разбива
// низът S[1...k]; Last[k] = дължината на последния палиндром.
Dyn[0] ← 0
for k ← 1 to n do
   Dyn[k] ← Dyn[k-1]
   Last[k] ← 1
   for L ← 2 to k do
      if Dyn[k-L] < Dyn[k] and IsPalindrome(S[k+1-L...k])
         Dyn[k] ← Dyn[k-L]
         Last[k] ← L
   Dyn[k] ← Dyn[k]+1
k ← n
while k > 0 do    // възстановяване на решението:
   L ← Last[k]    // палиндромите се отпечатват отзад напред
   print S[k+1-L...k]
   k ← k-L
return Dyn[n]
```

Анализ на алгоритъма: Функцията `IsPalindrome` обхожда подадения низ с дължина L еднократно, затова нейното време е $\Theta(L)$ в най-лошия случай: когато низът е палиндром.

Следователно времето на двата вложени цикъла for е равно по порядък на

$$\sum_{k=1}^{n} \sum_{L=2}^{k} L = \sum_{k=1}^{n} \frac{k^2 + k - 2}{2} = \Theta(n^3).$$

Времето на цикъла while е $\mathrm{O}(n)$, защото променливата k намалява от n до 0 с поне една единица при всяко изпълнение на тялото на цикъла; т.е. то се изпълнява не повече от n пъти.

Окончателно, времето на целия алгоритъм е $\Theta(n^3) + \mathrm{O}(n) = \Theta(n^3)$.

Задача 4. Когато търсим мястото на k-тия елемент, може да се наложи да го разместим със $0, 1, 2, \ldots, k-1$ от предишните $k-1$ елемента, тоест средно със

$$\frac{0+1+2+\cdots+(k-1)}{k} = \frac{k(k-1)}{2k} = \frac{k-1}{2} \text{ елемента.}$$

Средният брой размествания общо за целия алгоритъм е равен на

$$\sum_{k=2}^{n} \frac{k-1}{2} = \frac{(n-1)\,n}{4} = \Theta(n^2),$$

където n е броят на елементите на дадения числов масив, който трябва да бъде сортиран. Броят на сравненията е с едно повече за всяка стойност на k, а общо за целия алгоритъм — със $n-1$ повече. Следователно средният брой сравнения на елементи е равен на

$$\frac{(n-1)\,n}{4} + n - 1 = \Theta(n^2).$$

Окончателно, средният брой операции (сравнения и размествания на елементи) е $\Theta(n^2)$. Това е средната времева сложност на сортирането чрез вмъкване.