

**Зад. 1** Дадени са  $n$  болта, нито два от които не са с еднакъв диаметър. Дадени са и  $n$  гайки, като за всеки болт има точно една гайка, която му пасва. Не е разрешено да се сравняват болт с болт или гайка с гайка. Разрешено е да се правят само сравнения от вида **Сравни болт  $i$  с гайка  $j$** , като резултатът от такова сравнение е точно едно от следните три: или **Болт  $i$  има по-голям диаметър от гайка  $j$** , или **Болт  $i$  има по-малък диаметър от гайка  $j$** , или **Болт  $i$  пасва точно на гайка  $j$** .

- 10 т. а) Докажете, че всеки алгоритъм, който намира кой болт на коя гайка съответства, използвайки само сравнения от посочения вид, работи във време  $\Omega(n \lg n)$ .
- 10 т. б) Предложете алгоритъм, който намира кой болт на коя гайка съответства, използвайки само сравнения от посочения вид, и има **средна** сложност по време  $O(n \lg n)$ . Можете да опишете алгоритъма общо: не с псевдокод, а само да посочите идеята, използвайки наготово факти, които са изучавани на лекции.
- 20 т. в) Предложете итеративен алгоритъм, който намира най-малкия болт и най-малката гайка, използвайки не повече от  $2n - 2$  сравнения от посочения вид. Единственият вид достъп, който Вашият алгоритъм може да прави до болтовете и гайките, е викане на функция `compare(B[i], N[j])`, където  $B[i]$  е  $i$ -ият болт, а  $N[j]$  е  $j$ -тата гайка. Тази функция връща
- 1, ако  $i$ -ият болт е с по-голям диаметър от  $j$ -тата гайка,
  - -1, ако  $i$ -ият болт е с по-малък диаметър от  $j$ -тата гайка,
  - 0, ако  $i$ -ият болт пасва точно на  $j$ -тата гайка.

Този алгоритъм трябва да бъде написан подробно в псевдокод.

**Решение:** а) Всеки болт пасва на точно една гайка по условие, а от елементарни физически съображения следва, че и всяка гайка пасва на точно един болт. Ерго, пасването болтове-гайки е биекция и всеки алгоритъм за тази задача трябва да установи биекция между множеството на болтовете и множеството на гайките. Тези биекции са точно  $n!$ . Следователно, всяко дърво на вземане на решения за тази задача има поне  $n!$  листа. Въпросите, които е разрешено да се задават, имат по три отговора, следователно дървото има разклоненост 3. Оттук височината на дървото е поне  $\log_3(n!)$ . В асимптотична нотация, височината е  $\Omega(n \lg n)$ , използвайки добре известния резултат за асимптотиката на логаритъм от факториел.

б) Задачата лесно може да се реши от алгоритъм със средна сложност по време  $O(n \lg n)$ , ако използваме идеята на QUICKSORT. Идеята на QUICKSORT (ако няма еднакви елементи) е да разбием множеството спрямо някакъв елемент, наречен *pivot*, на множеството от елементите, по-големи от *pivot*, и множеството от елементите, по-малки от *pivot*, и да работим рекурсивно върху двете подмножества. Това обаче може да се приложи директно само ако можем да сравняваме всеки два елемента. В тази задача елементите са болтове и гайки и не е разрешено да се сравнява болт с болт или гайка с гайка. Така че леко усложняваме идеята: б.о.о., избираме произволен болт  $b$  и спрямо него разбиваме множеството от гайките на три подмножества: множеството  $S$  от гайките, които са с по-малък диаметър от  $b$ , гайката (тя е точно една)  $\gamma$ , която му пасва, и множеството от гайките  $L$ , които са с по-голям диаметър от  $b$ . Това правим в линейно време с пробване на всяка гайка с  $b$ .

Това обаче не ни дава веднага желаното разбиване, защото множеството от болтовете в момента не е разбито на три, а само на две:  $\{b\}$  и останалите болтове. Вземаме  $\gamma$  и спрямо нея разбиваме множеството от останалите болтове на множеството  $A$  от тези, които имат по-малък диаметър от  $\gamma$ , и множеството  $B$  от тези, които имат по-голям диаметър от  $\gamma$ . Очевидно  $|S| = |A|$  и  $|L| = |B|$ . Нещо повече, съществува биективно съответствие-пасване между болтовете от  $A$  и гайките от  $S$ , и също така има биективно съответствие-пасване между болтовете от  $B$  и гайките от  $L$ . Тогава  $\langle S, A \rangle$  е пример на същата задача, но с по-малък размер; също така и  $\langle L, B \rangle$  е пример с по-малък размер. Ако  $|S|$  е достатъчно голямо, решаваме рекурсивно задачата върху  $\langle S, A \rangle$ , в противен случай решаваме задачата с брутална сила в константно време. Аналогично, ако  $|L|$  е достатъчно голямо, решаваме рекурсивно върху  $\langle L, B \rangle$ , в противен случай решаваме задачата с брутална сила в константно време.

Очевидно, асимптотичния анализ е точно като QUICKSORT с функцията PARTITION. В най-лошия случай, т.е. когато на всяко ниво на рекурсията имаме лошия късмет да изберем екстремален болт, сложността е квадратична; в най-добрия случай е  $\Theta(n \lg n)$ ; в средния случай е пак  $\Theta(n \lg n)$ , което може да се покаже със същото рекурентно уравнение, което използвахме за средната сложност на QUICKSORT. Това, че правим два PARTITION-а, веднъж на гайките спрямо произволен болт и после на болтовете срещу неговата гайка, няма значение за асимптотиката.

в) Идеята е такава: разглеждаме някаква произволна наредба на болтовете, да кажем, масив от болтовете  $B[1, \dots, n]$ , и някаква произволна наредба на гайките, да кажем, масив от гайките  $N[1, \dots, n]$ . За целите на задачата, и болтовете, и гайките се идентифицират с диаметрите си. Б.о.о., нека диаметрите са цели положителни числа. Сравняваме  $B[1]$  с  $N[1]$ . Ако  $B[1]$  е по-голям, отхвърляме го като невъзможен (той не може да е най-малкия болт) и пробваме с  $B[2]$  и  $N[1]$ , и така нататък. Ако  $N[1]$  е по-голяма, отхвърляме я и пробваме с  $B[1]$  и  $N[2]$  и така нататък. Ако  $B[1]$  пасва на  $N[1]$ , не можем да отхвърлим тази двойка, но все още не сме сигурни, че те са най-малките. Запомняме тази двойка като текущ кандидат и минаваме нататък, тоест разглеждаме следващия болт и следващата гайка; ако няма следващ болт или следваща гайка, то край на алгоритъма. Със сигурност ще намерим поне една такава пасваща двойка, но може да има много такива. Следващата двойка пасващи болт и гайка, която ще намерим, ако изобщо има следваща, или ще е от по-малки болт и гайка, при което тя (следващата) ще стане текущата, или ще е от по-големи болт и гайка, при което ще я прескочим. Двойката, която е останала накрая, е минималната. Следният псевдокод реализира тази идея.

```

BOLTS-NUTS( $B[1, \dots, n]$ ,  $N[1, \dots, n]$ : posint)
1  (* Има биекция между двата входни масива *)
2  done  $\leftarrow$  FALSE,  $i \leftarrow 1$ ,  $j \leftarrow 1$ 
3  min  $\leftarrow \infty$ 
4  while not(done) do
5      result  $\leftarrow$  compare( $B[i]$ ,  $N[j]$ )
6      switch
7          case result = 1
8               $i++$ 
9              if  $i = n$ 
10                 done  $\leftarrow$  TRUE
11          case result = -1
12               $j++$ 
13              if  $j = n$ 
14                 done  $\leftarrow$  TRUE
15          case result = 0
16              if  $B[i] < \text{min}$ 
17                  min  $\leftarrow B[i]$ 
18                   $i^* \leftarrow i$ 
19                   $j^* \leftarrow j$ 
20               $i++$ 
21               $j++$ 
22              if  $i = n$  or  $j = n$ 
23                  done  $\leftarrow$  TRUE
24  return  $\langle B[i^*], N[j^*] \rangle$ 

```

**Зад. 2** Разгледайте задачата за намиране на най-къс път в тегловен ориентиран граф с положителни тегла на ребрата. Разгледайте задачата във варианта от даден връх  $s$  до всички останали върхове, като за най-късите пътища се искат само теглата им (а не самите пътища).

1 т.      а) Напишете алгоритъма на Dijkstra за тази задача.

19 т.      б) Докажете коректността на този алгоритъм.

**Решение** Това се преподава на лекции.

**Зад. 3** Представете си лабиринт, който се състои от стаи и коридори между стаите. Коридорите са еднопосочни: във всеки коридор може да се върви само в едната посока и никога в обратната посока. Знае се, че в този лабиринт е невъзможно да се въртим в кръг; тоест, ако напуснем дадена стая през някакъв коридор, излизащ от нея, няма как да се върнем в нея повече. Даден е лабиринтът като множеството от стаите  $\{r_1, r_2, \dots, r_n\}$  и множеството от коридорите, като всеки коридор е наредена двойка  $(r_i, r_j)$  от началната си и крайната си стая.

Предложете ефикасен алгоритъм, който по дадени стаи  $r_p$  и  $r_q$  връща броя на различните начини да тръгнем от стая  $r_p$  и да пристигнем в стая  $r_q$ , движейки се в лабиринта. Обосновете накратко коректността и сложността по време.

**Решение:** Задачата може да се моделира с ориентиран граф, чиито върхове са стаите, а коридорите са ребрата. Невъзможността да се въртим в кръг казва, че графът е даг. Трябва да направим ефикасен алгоритъм, който намира броя на ориентираните пътища от даден връх  $s$  до друг даден връх  $t$  в дага.

NUM-PATHS-DAG( $G = (V, E) : \text{dag}, s, t \in V$ )

```

1  for  $x \in V$ 
2      num[x]  $\leftarrow 0$ 
3  num[s]  $\leftarrow 1$ 
4  TOPOSORT( $G$ )
5  for  $x \in V$  in the topo order
6      for  $y \in \text{Adj}[x]$ 
7          num[y]  $\leftarrow \text{num}[y] + \text{num}[x]$ 
8  return num[t]
```

Коректността може да се обоснове чрез следната рекурсия. Нека  $n(x)$  означава броя на пътищата от  $s$  до  $x$ .

$$n(x) = \begin{cases} 1, & \text{ако } x = s \\ \sum_{(y,x) \in E} n(y), & \text{в противен случай} \end{cases}$$

Щом няма цикли, очевидно има точно един път от  $s$  до  $s$ . За всеки друг връх  $x$ , броят на пътищата от  $s$  до  $x$  е сумата по всички предшественици  $y$  на  $x$  на броевете на пътищата от  $s$  до  $y$ .

Сложността по време е  $\Theta(n + m)$ . От лекции знаем, че сложността на топологичното сортиране е  $\Theta(n + m)$ , а цикълът на редове 5–7 също има сложност  $\Theta(n + m)$ , защото той минава през всеки списък на съседства по веднъж.

**Зад. 4** Контекстно-свободна граматика (КСГ) е наредена четворка  $\Gamma = \langle \Sigma, N, S, R \rangle$ , където  $\Sigma$  е крайно множество от терминали ( $\Sigma$  е азбуката),  $N$  е крайно множество от нетерминали, такова че  $N \cap \Sigma = \emptyset$ ,  $S \in N$  е фиксиран нетерминал, наречен начален нетерминал, и  $R$  е множеството от правила (още се наричат продукции) от вида  $A \rightarrow X$ , където  $A$  е нетерминал, а  $X \in (\Sigma \cup N)^*$ . Формално,  $R$  е бинарна релация  $R \subseteq N \times (\Sigma \cup N)^*$ . Дефинираме бинарната релация  $\Rightarrow \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$  така:

$$x \Rightarrow z \stackrel{\text{def}}{\iff} \exists A \in N \exists x_1, x_2 \in (\Sigma \cup N)^* (x = x_1 A x_2 \text{ и има правило } A \rightarrow \alpha \text{ и } z = x_1 \alpha x_2)$$

Релацията  $\Rightarrow^*$  е рефлексивното и транзитивно затваряне на  $\Rightarrow$ . Ако  $x \Rightarrow^* y$ , казваме, че има *дериация* от  $x$  до  $y$  в  $\Gamma$ . За всеки  $y \in \Sigma^*$  казваме, че  $y$  има *дериация* в  $\Gamma$ , ако  $S \Rightarrow^* y$ .

КСГ е в Нормална Форма на Чомски (пишем кратко КСГНФЧ), ако всички правила са от вида  $A \rightarrow BC$  или  $A \rightarrow a$  или  $S \rightarrow \epsilon$ , където  $A, B$  и  $C$  са нетерминали,  $a$  е терминал, а  $\epsilon$  е празният стринг.

Предложете ефикасен алгоритъм, който по дадена КСГНФЧ  $\Gamma = \langle \Sigma, N, S, R \rangle$  и  $y \in \Sigma^*$  връща 1, ако  $y$  има дериация в  $\Gamma$ , или 0 в противен случай. Дайте съвсем кратка обосновка на коректността и сложността по време на предложението от Вас алгоритъм. Приемете, че размерът на  $\Gamma$  е  $O(1)$ .

**Решение:** Ще построим алгоритъм по схемата динамично програмиране. Този алгоритъм е добре известен и се нарича СΥΚ на имената на създателите си Cocke, Younger и Kasami.

Нека  $y = y_1 y_2 \dots y_n$ . Б.о.о., нека  $n \geq 1$ . Ако за всеки  $y_i \dots y_j$ , където  $1 \leq i \leq j \leq n$ , намерим множеството  $X_{i,j} = \{A \in N \mid A \Rightarrow^* y_i \dots y_j\}$ , то ние сме готови! Тогава връщаме 1, ако  $S \in X_{1,n}$ , и връщаме 0 в противен случай. Това, което позволява да направим ефикасен алгоритъм е, че при  $i < j$  можем да пресмятаме ефикасно всяко  $X_{i,j}$  от множествата  $X_{i',j'}$ , където  $i \leq i' \leq j' \leq j$  и  $j' - i' < j - i$ .

Базата на рекурсията са множествата  $X_{i,i}$  за  $1 \leq i \leq n$ :

$$\forall i \in \{1, 2, \dots, n\} : X_{i,i} = \{A \in N \mid \text{има правило } A \rightarrow y_i\}$$

защото в КСГНФЧ, ако по време на дериацията в сентенциалната форма се появи нетерминал, той никога не изчезва в смисъл, че в крайния стринг ще има поне един съответстващ на него терминал; ерго, правилата от вида  $A \rightarrow BC$  никога не могат да стартират дериация на еднобуквен стринг.

При  $i < j$ ,  $X_{i,j}$  е обединението, по всички нетривиални факторизации  $\langle y_i \dots y_k, y_{k+1} \dots y_j \rangle$  на стринга  $y_i \dots y_j$  (тоест, по всички  $k \in \{i, \dots, j-1\}$ ) на тези нетерминали, които са  $A$ -то в продукции от вида  $A \rightarrow BC$ , такива че  $B \Rightarrow^* y_i \dots y_k$  и  $C \Rightarrow^* y_{k+1} \dots y_j$ . Но тези нетерминали  $B$  и  $C$  са елементите на множествата  $X_{i,k}$  и  $X_{k+1,j}$ . При изчисляването на  $X_{i,j}$  множествата  $X_{i,k}$  и  $X_{k+1,j}$  са вече известни.

Самият алгоритъм може да се реализира по начин, аналогичен на реализацията на алгоритъма за MATRIX-CHAIN MULTIPLICATION, изучаван на лекции.

DERIVATION( $\Gamma = \langle \Sigma, N, S, R \rangle, y \in \Sigma^*$ )

```

1  n ← |y|
2  for i ← 1 to n
3      M[i, i] ← {A | (A → y_i) ∈ R}
4  for diag ← 2 to n
5      for i ← 1 to n - diag + 1
6          j ← i + diag - 1
7          M[i, j] ← ∅
8          for k ← i to j - 1
9              for B ∈ M[i, k]
10                 for C ∈ M[k + 1, j]
11                     for P ∈ R
12                         if P = (A → BC)
13                             M[i, j] ← M[i, j] ∪ {A}
14  if S ∈ M[1, n]
15      return 1
16  else
17      return 0
```

Обосновката на коректността следва от обосновката на рекурсията. Сложността по време е  $\Theta(n^3)$ , ако граматиката има константна сложност – тогава множествата  $M[a, b]$  имат големина, ограничени от константа, така че и трите най-вътрешни цикъла “въртят” само константен брой пъти.

**Зад. 5** Докажете, че следната задача е **NP**-пълна. Дадено е множество  $A$  от софтуерни приложения. Дадено и е множество  $C$  от “способности” (capabilities), като всяко приложение притежава една или повече от тези способности. Дадено е и число  $k$ . Пита се дали има подмножество  $X \subseteq A$ , такова че обединението от способностите на всички приложения в  $X$  е точно  $C$  и  $|X| \leq k$ .

**Решение:** Задачата е най-известна под името “SET COVER”: при дадено опорно множество и множество от негови подмножества, дали има  $\leq q$  от тези подмножества, които покриват опорното множество. Общият пример е  $\langle S, T, q \rangle$ , където  $S$  е опорното множество,  $T \subseteq 2^S$ , а  $q \in \mathbb{N}$ . Съответствието със **Зад. 6** е следното:  $A$  съответства на  $T$  и  $C$  съответства на  $S$ .

Очевидно  $SC \in \mathbf{NP}$ : има недетерминирана машина на Тюринг, която отгатва подмножество  $T' \subseteq T$  и после в полиномиално време проверява, че  $T' \subseteq T$ ,  $|T'| \leq q$  и  $T'$  покрива  $S$ .

Ще покажем, че  $VERTEX COVER \propto SET COVER$ . При даден пример  $\langle G = (V, E), k \rangle$  на  $VC$ , правим в полиномиално време пример  $\langle S, T, q \rangle$  на  $SC$ , в който  $q = k$ ,  $S = E$  и  $T = \{I(v) \mid v \in V\}$ , където “ $I(v)$ ” означава множеството от ребрата на  $G$ , инцидентни с връх  $v$ . В термините на **Зад. 6**, съответствието е такова: ребрата от примера на  $VC$  са способностите от **Зад. 6**, а всяко множество ребра с общ връх дава точно едно приложение от **Зад. 6** (за целите на това, което правим, идентифицираме приложенията със способностите, които имат).

Ако  $\langle G = (V, E), k \rangle$  е ДА-пример на  $VC$ , то по дефиниция има  $U \subseteq V$ , такова че на всяко ребро от  $E$  поне единият край е в  $U$  и  $|U| \leq k$ . Всеки връх от  $U$  задава точно едно множество-елемент на  $T$ . За съвкупността от тези множества, да я наречем  $T'$ , е вярно, че всеки елемент на  $S$  е в някое от тях; с други думи,  $T'$  покрива  $S$ . Освен това,  $|T'| = k = q$ .

В обратната посока, ако  $\langle S, T, q \rangle$  е ДА-пример на  $SC$ , то има подмножество  $T' \subseteq T$ , такова че  $T'$  покрива  $S$  и  $|T'| = q$ . В примера на  $VC$ , на  $T'$  съответстват  $q = k$  различни множества от ребра, като във всяко от тях ребрата имат общ край. С други думи,  $T'$  задава върхово покриване с размер не повече от  $k$  на графа.

Очевидно е, че конструкцията може да се извърши в полиномиално време.