

Сега ќе продолжим во посока алгоритам за минимизација на KDA.

Нека $M = (K, \Sigma, \delta, S, F)$ е KDA во којто сме можна да видиме недостатици со состојбата. Иде определено, перател $A_n \subseteq K \times \Sigma^*$ како следва: $(q, w) \in A_n \Leftrightarrow (q, w) \vdash_n^* (\ell, \varepsilon)$ за некое $\ell \in F$. Речиси тај перател A_n је дефинирано перател на еквивалентност, но тоги не е и у состојбата.

D. Нека $p, q \in K$. Казаше, што p, q са еквивалентни (имам $p \equiv q$ за KDA $M = (K, \Sigma, \delta, S, F)$) т.т.к. за всички думи $z \in \Sigma^*$ извршена еквивалентноста:

$$(q, z) \in A_n \Leftrightarrow (p, z) \in A_n.$$

Така ќе се докаже p, q са еквивалентни т.т.к. $E_p \cap E_q$ се сдружнат во един и само клас на еквивалентноста \sim_n . Това одате все овој не ни дава алгоритам. За да среќнем го алгоритам ние ќе определимо по групи ператели $b_i \in K$ од перателата \equiv , а конкретно $\equiv_0, \equiv_1, \equiv_2, \dots$ во среќните називи:

9. Една T.T.K. за произволна $z \in \Sigma^*$, така да, за $|z| \leq i$ е узр. еквивалентноста $(q, z) \in A_n \Leftrightarrow (p, z) \in A_n$.

Наша група е перателот \equiv_0 , среќ това \equiv_1, \dots и т.н.

\equiv_0 се состои винати само од једна класа $F \subseteq K \setminus F$.

Освен това бројт на класите ќе може да стане повеќе отколкото број на состојбата на K .

Така, што ние ќе мувимо перателите по-финија докаде \equiv_i и \equiv_{i+1} се вклапаат. Тогава \equiv_{i+1} ќе се вклапа во \equiv .

Последниот етап на алгоритама се дираше от класата за единствената буква.

Нека, за всички ќе се докаже $q, p \in K$ и за всички $i \geq 1$ $p \equiv_n q$ T.T.K. са извршени среќните ќе укажемо:

a) $p \equiv_{n+1} q$; δ) За всички $q \in \Sigma$, $\delta(q, q) \equiv_{n+1} \delta(p, q)$.

D-бо. От дефиниците $p \equiv_n q$, T.T.K. За всички думи $w \in \Sigma^*$ така да, за $|w| \leq n$ е узр., што $(p, w) \in A_n \Leftrightarrow (q, w) \in A_n$, т.е. за всички w , $|w| \leq n$ е узр. $(p, w) \in A_n \Leftrightarrow (q, w) \in A_n$ и за всички думи $|w| > n$ е узр. $(p, w) \in A_n \Leftrightarrow (q, w) \in A_n$

T.e. за било $w, |w| \leq n-1$ е узр. $(p, w) \in A_M \Leftrightarrow (q, w) \in A_M$ и
 за било $w = av$ е узр. $(p, av) \in A_M \Leftrightarrow (q, av) \in A_M$

T.T.K. a) $p \equiv_{n-1} q \wedge \delta(p, a) \vdash_M^*(f, \varepsilon)$ за $f \in F \Leftrightarrow$
 $(q, av) \vdash_M^*(f, \varepsilon)$ за $f \in F$ T.T.K.

a) $p \equiv_{n-1} q \wedge \delta(p, a) \vdash_M(\delta(p, a), V) \vdash_M^*(f, \varepsilon)$ за $f \in F \Leftrightarrow$
 $(q, av) \vdash_M(\delta(q, a), V) \vdash_M^*(f, \varepsilon)$ за $f \in F$ T.T.K.

a) $p \equiv_{n-1} q \wedge \delta(p, a) \equiv_{n-1} \delta(q, a)$ за у. а $\in \Sigma$.

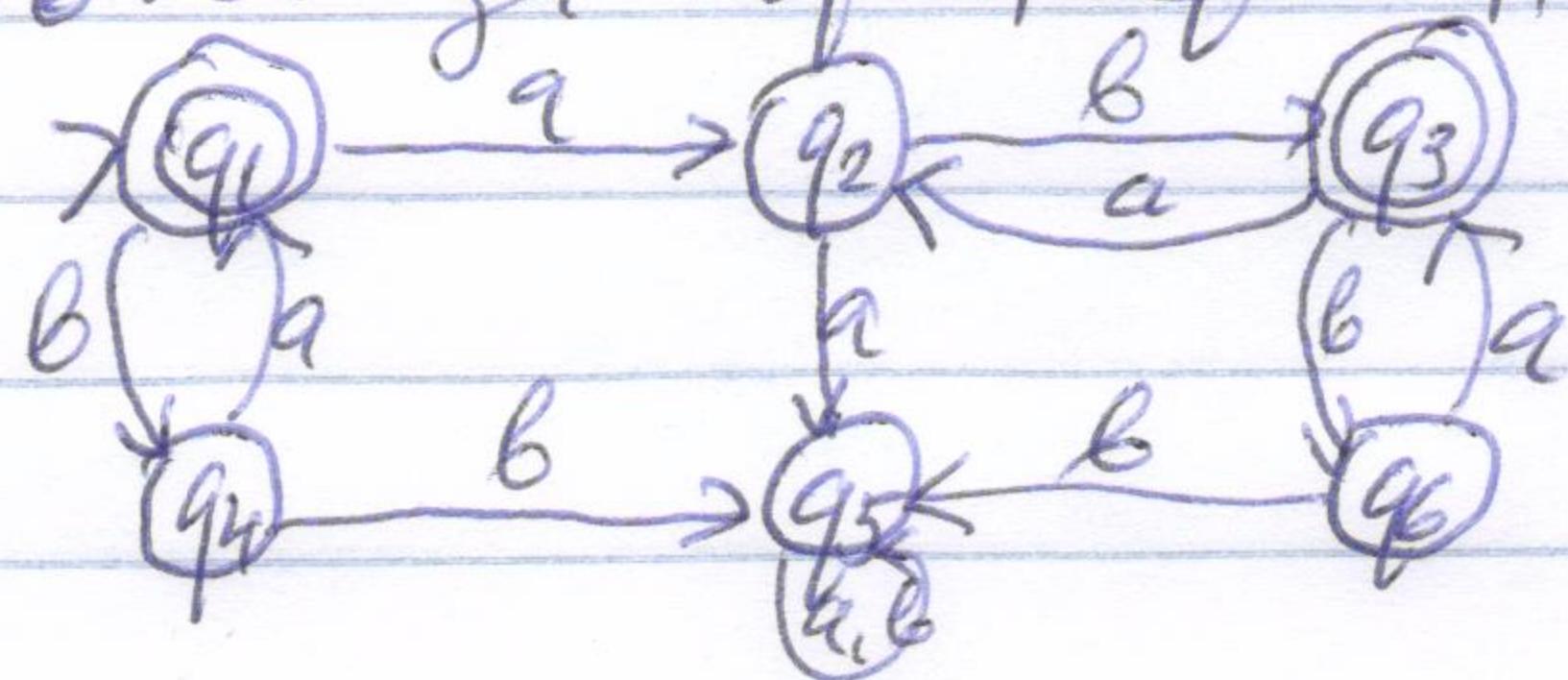
Тази лема ни дава безотрицателност на \equiv_{n-1} да използваме \equiv_n . Сега вече алгоритъмът за минимизация ще дава възможност за използване с класовете $K \setminus F \cup F$ за лев. \equiv_0 . Извоазува ки лемата за едната буква, от \equiv_{n-1} използваме \equiv_n . Но този:

Показване в наследство $F \cup K \setminus F$ га са класовете на \equiv_0
 repeat for $n := 1, 2, \dots$

на първите класовете на \equiv_n от \equiv_{n-1}

until $\equiv_n = \equiv_{n-1}$ съвпадат

Да приложим алгоритъма за премера на автомат, който разглеждане:



Първоначални класовете са $\{q_1, q_3\}, \{q_2, q_4, q_5, q_6\}$

Да пресметнем класовете на \equiv_1 .

$$\delta(q_2, b) = q_3, \delta(q_4, b) = q_5, \delta(q_5, b) = q_5, \text{т.е. } q_2 \neq q_4, q_2 \neq q_5$$

$$\delta(q_4, a) = q_1, \delta(q_5, a) = q_5, \text{т.е. } q_4 \neq q_5. \text{ Следователно,}$$

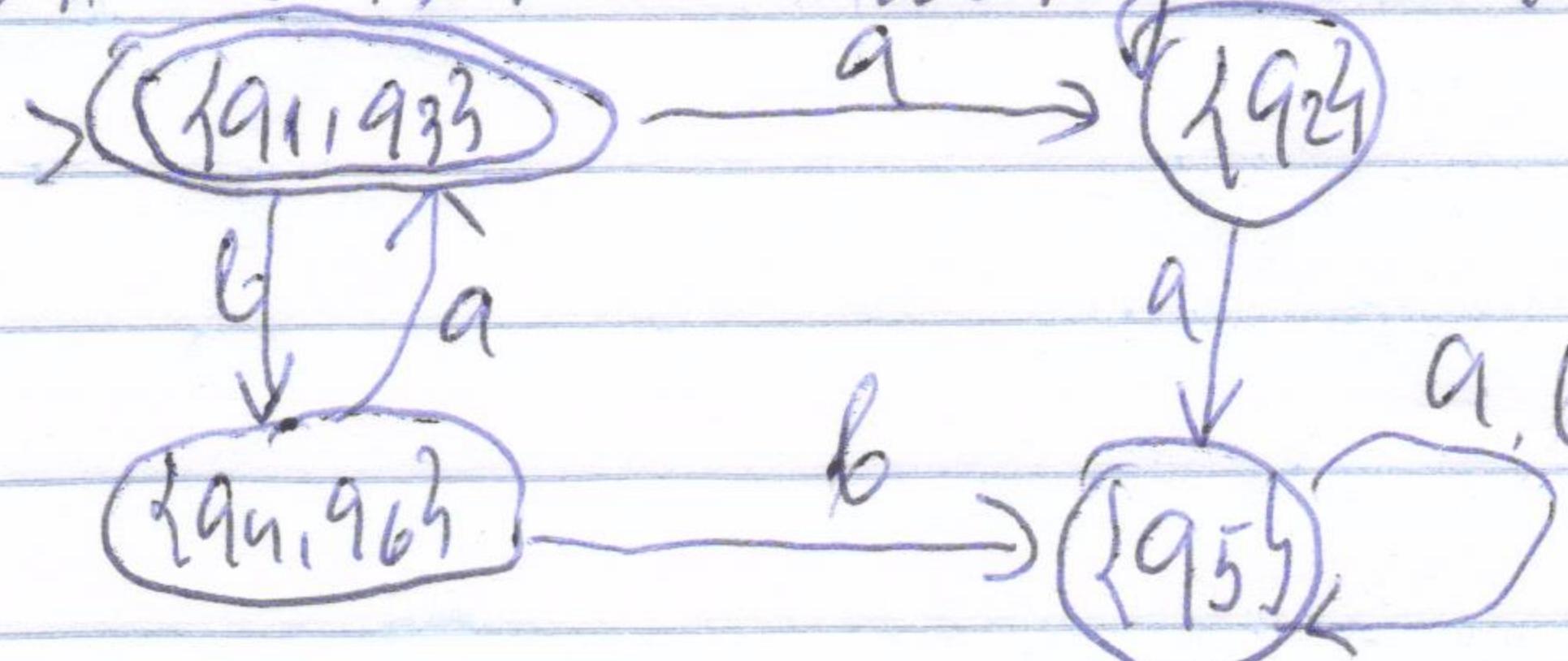
Понижаване на класовете на \equiv_1 са $\{q_1, q_3\}, \{q_2\}, \{q_4, q_5\}, \{q_6\}$

$$\delta(q_6, a) = q_3, \delta(q_2, a) = q_5, \text{т.е. } q_2 \neq q_6 \wedge \delta(q_6, a) = q_3, \delta(q_5, a) = q_5$$

т.е. $q_5 \neq q_6$

По-нататък \equiv_2 не дава нови класове и

около за това то понижаване $\{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}, \{q_6\}$, т.е.



е търсеният за минимиза-
щият автомат.

Скорост на растежа на функции и анализ на сложността на алгоритми

От математическия анализ знаем за скорост на растежа на редица и функции. За разлика от мат. анализ, където се разглежда растежа на функции при клонене на аргумента към различни точки, в частност при клонене на аргумента към ∞ , тук ще концентрираме вниманието си само към растежа на функции от типа $f: \mathbb{N} \rightarrow \mathbb{N}$, т.е. своя редица от естествени числа. Причината за този интерес е да можем да сравняваме различните алгоритми за ефективност. Затова и нашите разглеждането са по специфична. Този вид ще са нивие брой „степенарни операции“, кие ще разглеждаме само \mathbb{N} .

2. Нека $f: \mathbb{N} \rightarrow \mathbb{N}$. Ред на функцията f , се означава с $O(f)$ и е ред на стъпките на базисни функции $g: \mathbb{N} \rightarrow \mathbb{N}$, такива, че съществуват естествени числа $c, d, (c, d > 0)$ такива, че за всички естествени числа n е извънредно $g(n) \leq c \cdot f(n) + d$. Всъщност, достатъчно е да искаме n за всички естествени числа да бъде извънредно, а за всички естествени числа от известното съдържание, т.е. за $n \geq n_0$ за n_0 фиксирано и. Ако $g \in O(f)$, то също казваме, че степента на растежа на g не надвишава степента на растежа на f (имаме предвид при $n \rightarrow \infty$) и по търсата ще имаме $f \asymp g$. Ако за две функции $f, g: \mathbb{N} \rightarrow \mathbb{N}$ е извънредно $f \in O(g)$ и $g \in O(f)$ то ще пишем $f \asymp g$ и ще казваме, че f и g са обединени и същии ред (порядък), или че съвсем една и съща скорост на растеже. Означаването е, че реда на f е реда на g и във въвеждането. В общия случай, използвайки горната дефиниция, е трудно да се провери дали те са обединени и същии ред. Използвайки средствата на математическия анализ бихме могли по-лесно да проверим това. Ето как се прави това:

Твърдение. Нека $f, g: N \rightarrow N$. Тогава; ако:

a) $\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = K > 0$, то $f \asymp g$;

б) $\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = 0$, то $f \asymp g$, то $f \asymp g$ и $f \asymp g$.

Д-бо. а) Нека $\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = K > 0$. Тогава за $\epsilon = \frac{K}{2} > 0$

имаме, че също. Но: $\forall n \geq n_0$ е няк. $\left| \frac{f(n)}{g(n)} - K \right| < \frac{K}{2}$

$$\text{т.е. } -\frac{K}{2} < \frac{f(n)}{g(n)} - K < \frac{K}{2}, \text{ т.е. } \frac{K}{2} < \frac{f(n)}{g(n)} < \frac{3K}{2}$$

Всички $n \geq n_0$ и, следователно $\frac{K}{2} \cdot g(n) < f(n) < \frac{3K}{2} \cdot g(n)$, за $n \geq n_0$. Тогава $g(n) < \frac{2}{K} \cdot f(n)$ и $f(n) < \frac{3K}{2} \cdot g(n)$ за всички $n \geq n_0$. Нашествие на c_1, c_2 - елементи, че $g(n) \leq c_1 \cdot f(n)$ и $f(n) \leq c_2 \cdot g(n)$ за всички $n \geq n_0$, а след това за всички d_1, d_2 такива че за всички n , $g(n) \leq c_1 \cdot f(n) + d_1$ и $f(n) \leq c_2 \cdot g(n) + d_2$. Това показва $f \asymp g$.

Д-бо по-горе б) можем да покажем за произволното $\epsilon > 0$ че за всички $n \geq n_0$ е изпълнено $\epsilon < \frac{f(n)}{g(n)} < \epsilon$ и можем да използваме също $\frac{f(n)}{g(n)} < \epsilon$, т.е. $f(n) < \epsilon \cdot g(n)$ и така да покажем $f \asymp g$.

Пример 1. Нека $f(n) = 1000 \cdot n^3 + 10n + 5$ и $g(n) = n^3 + 300n^2 + n$. Тогава $\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = \frac{1000}{1} = 1000$ и $f \asymp g$.

Изводът, ако $f \asymp g$ са полиноми от степен m и n , то ако $m = l$, то $f \asymp g$, а ако $m < l$, то $f \asymp g$.

С други думи $n \asymp n^2 \asymp n^3 \asymp n^4 \dots$

Също можем да проверим с горното твърдение, че $n^k \asymp 2^n$ за фиксирано K . За тази цел трябва да проверим, че $\lim_{n \rightarrow +\infty} \frac{n^k}{2^n} = 0$. Първият начин за доказване е да се покаже, че $\sum_{n=1}^{+\infty} \frac{n^k}{2^n} < \infty$.

Вторият начин е да разгледаме последователността $a_n = \frac{n^k}{2^n}$. Да разгледаме за същото $\frac{a_{n+1}}{a_n} = \frac{(n+1)^k \cdot 2^n}{2^{n+1} \cdot n^k} = \frac{1}{2} \left(\frac{n+1}{n} \right)^k$.

Тогава $\lim_{n \rightarrow \infty} \frac{1}{2} \cdot \left(\frac{n+1}{n} \right)^k = \frac{1}{2} \cdot \left(\lim_{n \rightarrow \infty} \frac{n+1}{n} \right)^k = \frac{1}{2} < 1$.

Оттук $\lim_{n \rightarrow \infty} \frac{a_{n+1}}{a_n} = \frac{1}{2}$ и знаем за $q = \frac{1}{2} < 1$ иначе, те съв.

но: За вс. $n \geq n_0$ е възп $\left| \frac{a_{n+1}}{a_n} - \frac{1}{2} \right| < \frac{1}{2}$, т.е. $\frac{1}{2} - \frac{1}{2} < \frac{a_{n+1}}{a_n} < \frac{1}{2} + \frac{1}{2}$.

С други думи за $n \geq n_0$ е възп., че $a_{n+1} < a_n$, т.е. редицата a_1, a_2, \dots е монотоно нарастваща от

уверено и също нараства и $a_n > 0$, т.е. тя е ограничена отгору. Следователно съв. $\lim_{n \rightarrow \infty} a_n = l$. Тогава

$$l = \lim_{n \rightarrow \infty} a_n = \lim_{n \rightarrow \infty} a_{n+1} =$$

$$\lim_{n \rightarrow \infty} \frac{(n+1)^k}{2^{n+1}} = \lim_{n \rightarrow \infty} \left(\frac{(n+1)^k}{2^{n+1}} \cdot \frac{a_n}{a_n} \right) = \lim_{n \rightarrow \infty} \frac{1}{2} \cdot \frac{(n+1)^k}{n} \cdot \lim_{n \rightarrow \infty} a_n = \frac{1}{2} \cdot 1 \cdot l$$

т.е. $l = \frac{1}{2} l$. Оттук $l = 0$, т.е. $\lim_{n \rightarrow \infty} a_n = 0$.

Следователно $\lim_{n \rightarrow \infty} \frac{a_n^k}{2^n} = 0$, и също съвдадено, $n^k \leq 2^n$.

Аналогично можем да проверим, че $2^n \leq 3^n \leq \dots \leq n! \leq n^n$

За да оценяваме ефективността на алгоритмите и да ги сравняваме по ефективност ние ще тръбва да имаме единица (време за изпълнение), с която да мерим сложността на алгоритмите. Такава единица ще настъпва също експериментално. За нас експериментална определящата същност е броя на операциите на алгоритмите. За два експериментални алгоритма ще съществува съпоставка на броя на операциите, които са ювти или не, събиране, изваждане, умножение, деление на числа. Издигнато тръбва да обединим, че различните типове къмки от типа for, while, until, -- не бива да не можем да ги считаме за експериментални операции, защото (което и бързи да са начините технически средства) времето за изпълнението им е недопредвидено, а може и да бъде дълъг. Докато за експерименталните операции съществуват компютри изразходват част от килосекунди. Естествено, времето е основният фактор при оценката на сложността на алгоритмите и основният мерачим съпоставими като време ефект на друга.

Не започнем с алгоритми, които често използвани също използвани от тук наред. Такива са:

1) Рекурсивно и транзитивно заобикарение на една релация.

Нека имаме крайно множество A и елементи и релация $R \subseteq A$. Ако си имам за двойката $\langle A, R \rangle$ като ориентиран граф, тие разглеждате идейни съесции как може да се интерпретира рефлексивното и транзитивното заобикарение - като добавим прилики на всички връхове, ч, ако има маркирът от един връх до друг, тие добавяме думата маркирът от началото на маркирата като края на маркиръта. Или иначе предвид тази практика, тие можем да разгледаме следния алгоритъм: Разгледаме всички връхове b_1, \dots, b_i с i -ти и проверяваме дали е ориентиран маркирът (без прилики). Ако е така, добавим (b_1, b_i) във R^* . В противен случаи ниво не извърши. Какви елементи ти се изразиха навсякъде? - Изпроверяваме дали $(b_1, b_1), \dots, (b_i, b_i)$ юз R^* и съответно добавим (b_1, b_i) във R^* . Следователно за всеки i , имаме най-много n^i елементарни операции или общо $n + n^2 + \dots + n^n = n(1 + \dots + n^{n-1})$, което е фундаментално от $O(n^{n+1})$. Този алгоритъм не е ефективен. Тук изпушахме добавянето на (a_i, a_j) $i=1, \dots, n$ из R^* , но това никога нямаше да попадне тук.

За същата задача можем да използваме и друг алгоритъм:

$$R^* := R \cup \{a_i, a_j \mid a_i \in A\}, \quad A = \{a_1, \dots, a_n\}$$

while съществуват ед. $a_i, a_j, a_k \in A$ такива, че

$$(a_i, a_j) \in R^*, (a_j, a_k) \in R^* \text{ и } (a_i, a_k) \notin R^* \text{ добави } (a_i, a_k) \text{ във } R^*.$$

Несто се вижда, че имаме най-много n^2 двойки (a_i, a_k) за добавяне. Следователно трябва най-много n^2 изпушта да използваме while-цикла. За всички while-цикли ни трябва най-много из операции. Така, че общият брой на елементарните операции е $O(n^5)$.

Нека обрнати внимание, че тук не можем да кажем колко точно са всички елементарни операции, а само даваме горна гранична на броя елементарни операции. И така за една и съща задача имаме два алгоритма с различна ефективност. За всички е $O(n^5)$, че броя ръст алгоритъм

е не-эффективен.

За свидета загара можем да посочим и трет алгоритм:

$$R^* := R \cup \{(a_i, a_j) \mid a_i \in A\} \quad (\text{стапка } j=0)$$

for $j = 1$ to n do

 for $i = 1$ to n do

 for $k = 1$ to n do

 if $(a_i, a_j) \in R^*$ и $(a_j, a_k) \in R^*$ и $(a_i, a_k) \notin R^*$ then
 добави (a_i, a_k) към R^* .

Сложността на този алгоритм е очевидна от 3 вложени

for-цикла. Една очевидна граничка на врем.
онерасим е $3n^3$, т.е. врем. на сложността е $O(n^3)$.

За да покажем, че алгоритъм, посочен по-горе е коректен
той га дадем една дефиниция. Мисленето си за (A, R)
като за граф, ориентират мрежа $G(A, R)$. Наричаме подгра-
фия $a_{i_0}, a_{i_1}, \dots, a_{i_k}$ такава, че за всички $p=1, \dots, n$ е изпъл-
нено $(a_{i_{p-1}}, a_{i_p}) \in R$. За всички $m \geq 1$ (направи предвид орци-
тирането) от a_{i_0} до a_{i_m} , възможно е $i_p \leq n$, факт на този път
наричаме последователността на индексата i_0, i_1, \dots, i_{m-1} . Тук може
да имаме и привидни незначителни приколи.

Решетка на този незначителни приколи е 0 по дефиниция, а
затворено те имат леки десни бъзи. Сега сме готови да
покажем следното твърдение:

Тв. За всички $j=0, 1, 2, \dots, n$ след изваждането на $H_{i_0}-безъ-$
множи j -тия член R^* създаващта всички гребици (a_i, a_k)
такива, че свидетелствва за всички p , за $p \leq j$.

Д-бо. За $j=0$ твърдението е логично, защото на "съзлока"
към R^* са добавени всички незначителни приколи към R^* .
Допускай, че твърдението е валидно за $j \leq n$. Иде го покажем
за $j=n+1$. Нека га разгледаме сега $m \geq a_{i_0}, a_{i_1}, \dots, a_{i_n}$,
за които факта се доказва също в същата точка a_{i_0}, \dots, a_{i_n} ,
 $i_0=n+1$ и за всички i_1, \dots, i_{n-1} разликите от i_0 е изпълнено
то, че $i_0 \leq m$. Това означава, че нито a_{i_0}, \dots, a_{i_n} има

дати $\leq m$ и нито a_{i_0}, \dots, a_{i_n} има дати $\leq m$. След това $\Pi^{i_0}(a_{i_0}, a_{i_1})$
 $\dots, \Pi^{i_n}(a_{i_0}, a_{i_n})$ са добавени към R^* при $(n+1)$ -ия H_{i_0} -безъ-

нест цикъл. Съговаряно, (a_{i_0}, a_{i_k}) се добави към R^* , отгас-
то алгоритъма) след $(k+1)$ -ият ивиц-блок на цикъл.
Ние проверихме само ако първа се покриа само 1
този. Прехода от 1 ивица към 2 ивици е лесна, а ището,
Нека $a_{i_0} \rightarrow a_{i_1} \rightarrow a_{i_2} \rightarrow a_{i_3} \rightarrow a_{i_k}$ е ивица към то
 $i+1, 1 \leq i \leq k-1$, съговаряно, $p = i$ или $p = i+1$, т.е. първа се
покриа само 2 ивици. Това га разглеждаме нито
 $a_{i_0}, a_{i_1}, \dots, a_{i_k}$ (затова показватът съвсем, но оу-
трабане, за $(a_{i_0}, a_{i_k}) \in R^*$ и $(a_{i_k}, a_{i_0}) \in R^*$ и, съговаряно,
 $(a_{i_0}, a_{i_k}) \in R^*$. Така ищем с ивиц. Относно s , га покри-
хнем, за (a_{i_0}, a_{i_k}) се добави към R^* ако първа се покриа
този 1s ивици. С това показваме, че алгоритъмът е
коректен.

Очевидно е, че Третият алгоритъм, който има $\Theta(4^n)$
е не-ефективен от боравъл.

За отдельният, че наименство на че се интересуваме
да има един алгоритъм към инициалната или
експертната съдътност. Користи един алгоритъм
има $\Theta(DP)$, където Р е инициалното, то казбане, за
алгоритъмът има инициалната съдътност.

Един алгоритъм има експертната съдътност ако
той има $\Theta(2^n), \Theta(3^n), \dots, \Theta(p^k)$, за всекивало
 $p \in N$.

Съговарящ алгоритъм, който че разглеждаме за съдът-
ност е за наридане на задача P на инициалното от-
носно $(k+1)$ -ата редача. Да наричаме наименование
дефинирамът:

д. Нека A е множество с инициалната, $B \subseteq A$ и $R \subseteq A^{k+1}$.

Зададено B^* на B относно R се нарича съговаряща:

а) Ако $a \in B$, то $a \in B^*$;

б) Ако $a_1, \dots, a_k \in B^*$ и $(a_1, \dots, a_k, b) \in R$, то $b \in B^*$.

Алгоритъмът, който че опишем е съговарящ:

$B^* := B$

while съвсем съвсем $(a_1, \dots, a_k, b) \in R$ и $a_1 \rightarrow a_k \in B^*$ то

$b \notin B^*$ do: го дава в $\kappa_B B^*$.

Сложността на алгоритма се определя от близкото времето за го даване на b и е линеен, когато е $O(n)$ и след това от while-циклика, която при експоненциални операции има сложността n^{k+1} . Следователно, сложността е $O(n^{k+2})$.

Да разгледаме и обобщената дефиниция.

Д. Нека A е матрическо с непустота, $B \subseteq H \cup R_i \subseteq A^{r_i}$, $i=1, \dots, K$. Зададено B^* на B от H и R_1, \dots, R_K се определя като сърдечка:

a) Ако $a \in B$, то $a \in B^*$;

б) Ако $(a_1, \dots, a_{i-1}, a_i) \in R_i$ и $a_1, \dots, a_{i-1} \in B$, то $a_i \in B^*$,

$i=1, \dots, K$.

Тук, аналогично на то-тозе, се вижда, че сложността на алгоритма (както и предвид изобилният на носещи резултати то-тозе) има сложността $O(n^{r+1})$, където $r = \max\{r_1, \dots, r_K\}$.