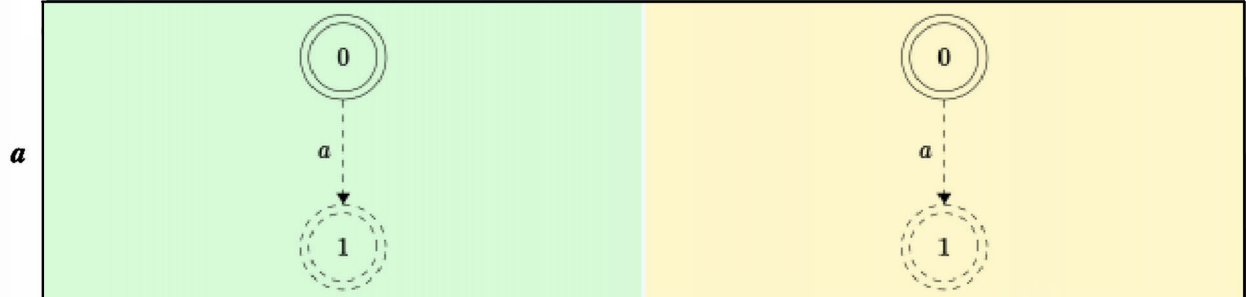
Минимален автомат на наставките

```
function add_letter(x):
    define p = last
    assign last = new_state()
    assign len(last) = len(p) + 1
    while δ(p, x) is undefined:
        assign δ(p, x) = last, p = link(p)
    define q = δ(p, x)
    if q = last:
        assign link(last) = q0
    else if len(q) = len(p) + 1:
        assign link(last) = q
    else:
        define cl = new_state()
        assign len(cl) = len(p) + 1
        assign δ(cl) = δ(q), link(cl) = link(q)
        assign link(last) = link(q) = cl
        while δ(p, x) = q:
            assign δ(p, x) = cl, p = link(p)
```

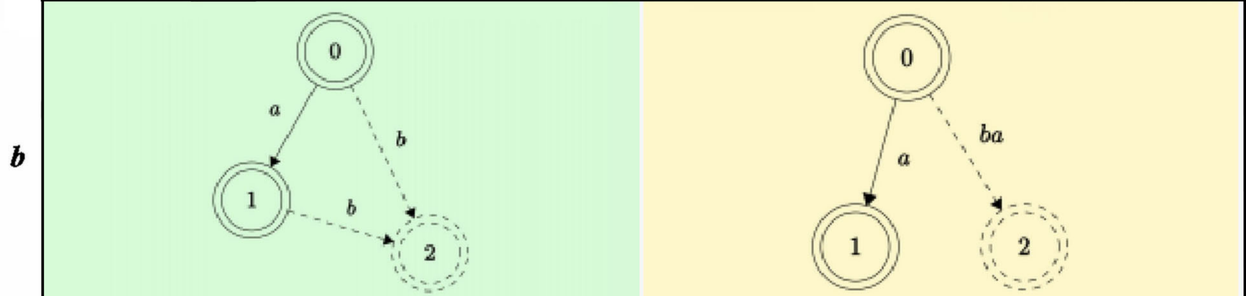
Алгоритъмът за построяване на минимален автомат на наставките е описан подробно в полето вдясно на псевдокод (взет от *Уикипедия*). По-точно, приведеният псевдокод описва процедурата *add_letter(x)*, която стъпка по стъпка променя текущото множество от състояния на автомата и преходи между тях, като прочита поредната буква x от входната дума w, чиито наставки трябва да разпознава автоматът. Множеството от състоянията не е именувано изрично; променливите *last*, p, q, *cl* и q_0 сочат към някакви състояния на автомата; командата *new_state()* създава ново състояние; q_0 е началното състояние на автомата; променливата *last* сочи към последното (току-що създадено) състояние на автомата. Променливите q_0 и *last* са глобални (тоест стойностите им се пазят между извикванията на процедурата *add_letter*), а променливите p, q и *cl* са локални (валидни са само за конкретно извикване на процедурата *add_letter*). Функцията $\delta(p, x)$ описва преходите на автомата, тоест $\delta(p, x)$ е състоянието, в което автоматът се озовава след прочитането на буквата x, ако преди това се е намирал в състоянието p. Ако преходът липсва, то резултатът от извикването $\delta(p, x)$ на функцията е недефиниран; в псевдокода това се отбелязва по два различни начина: със служебната дума `undefined` и с невъзможната стойност *last* (защото към току-що добавеното състояние все още няма преходи). Обозначението $\delta(p)$ без втори аргумент е списъкът на всички преходи, излизащи от дадено състояние p. Пази се и кореново дърво, което има корен q_0 и е представено чрез указатели *link* към родителите; върховете му са точно състоянията на автомата, а *link(p)* е родителят на p. Коренът няма родител, затова $link(q_0) = last$ (служебно). Също $\delta(last, x) = last$.

Отначало автоматът има едно състояние q_0 и дървото има един връх — същото състояние. После процедурата *add_letter* се извиква последователно с буквите x_1 , x_2 , . . . , x_n на w и всеки път актуализира автомата и дървото. Накрая се определят финалните състояния: те са върховете по пътя от *last* (последното добавено състояние) до корена q_0. Той може да се изключи, защото разпознава само празната наставка; *last* разпознава цялата дума w (тя се смята за своя наставка), а може да разпознава и други наставки.

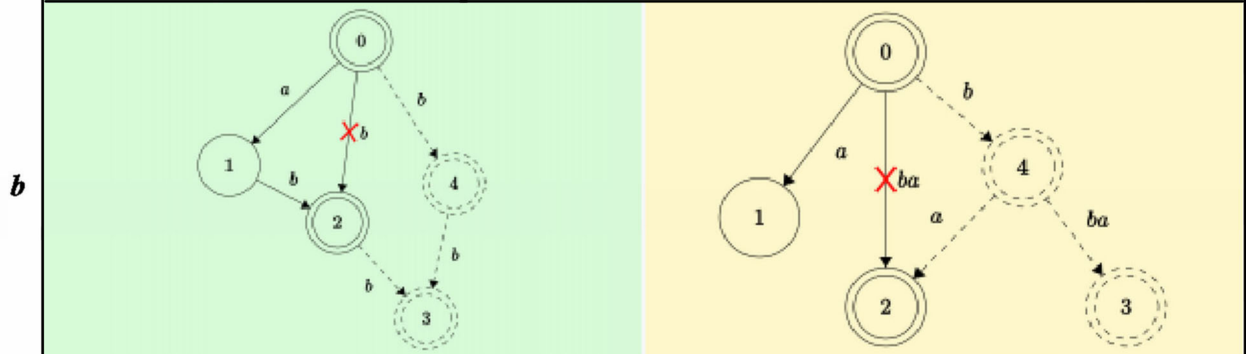
Пример от *Уикипедия*: Автомат на наставките на думата $w = abbcb$, построен стъпка по стъпка.



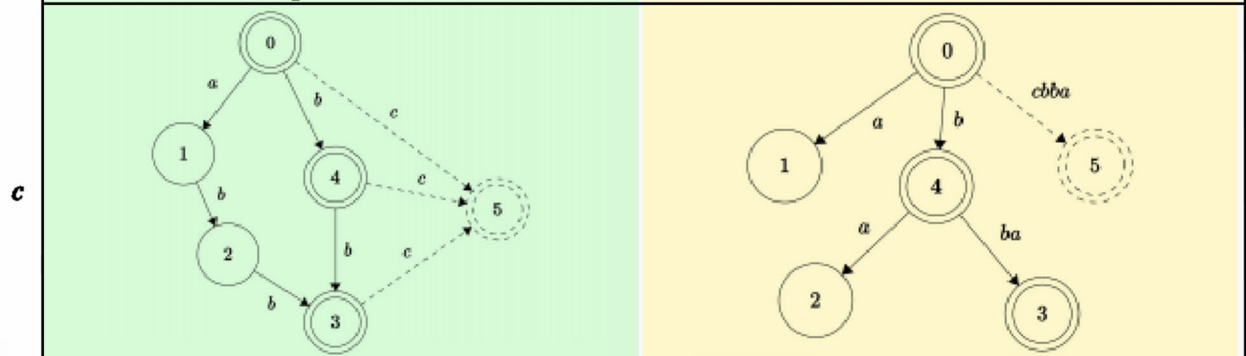
След първия знак се добавя само едно състояние в автомата и само един връх в дървото.



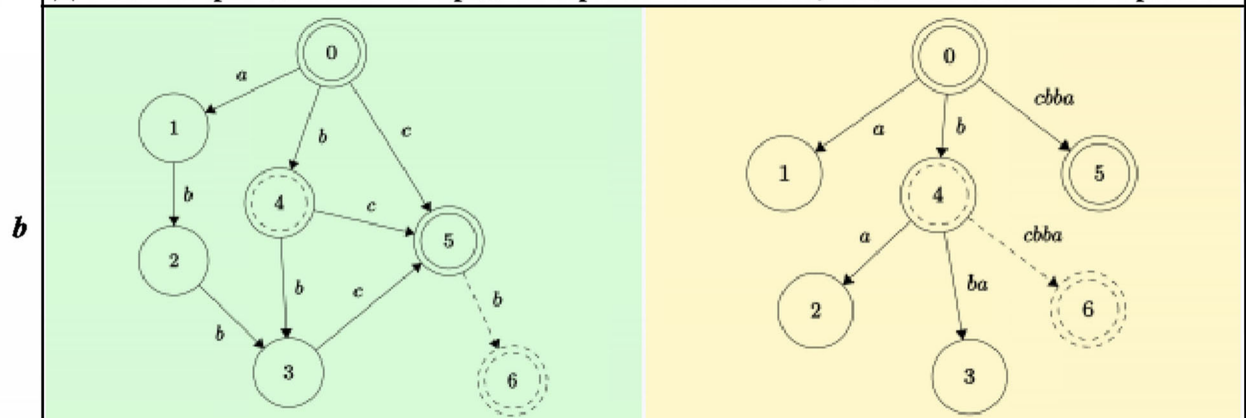
Добавят се преходи от всички предишни финални състояния, защото *b* се появява за пръв път.



Състояние № 2 разпознава *ab* и *b*, но само *b* е наставка, затова *b* се отделя в състояние 4.



Добавят се преходи от всички предишни финални състояния, защото *c* се появява за пръв път.



Състояние № 4 съдържа само думата *b*, която е наставка, затова то не се разделя.

Финалните състояния са означени с двойни кръгчета. Началното състояние има номер 0, тоест $q_0 = 0$.

Веднъж построен, автоматът може да разпознава наставките на думата *w*. Подадена дума α се чете по стандартния начин: от началното състояние q_0 автоматът прочита знаците на α последователно и след всеки прочетен знак преминава в ново състояние по прехода, отбелязан с този знак (в случай че няма такъв преход, автоматът блокира и думата α се смята за неразпозната, тоест α не е наставка на *w*). Думата α е наставка на *w* тогава и само тогава, когато след прочитането на последния знак от α автоматът се намира във финално състояние.

Например *w* = *abbcb* има за наставки себе си, думите *bbcb*, *bcb*, *cb*, *b* и празната дума. Автоматът от примера по-горе ги разпознава съответно чрез финалните състояния 6, 6, 6, 6, 4 и 0, изминавайки съответно пътищата 012356, 04356, 0456, 056, 04 и 0. Финалните състояния са 0, 4 и 6, тоест тези по пътя от последния добавен връх 6 до корена 0 на дървото. Те са отбелязани с двойни кръгчета, но пунктирани, за да се различават от финалните състояния на предишния етап. Алгоритъмът намира финалните състояния само веднъж — след последната прочетена буква на думата *w*; но за удобство на описанието понякога се говори за финални състояния след всеки прочетен знак на думата *w* (например в един надпис на картинката по-горе се споменава за предишни финални състояния). Обратно, думите *bb* и *ac* не са наставки на *w* = *abbcb*. По-точно, *bb* се съдържа в *w*, затова автоматът съумява да прочете *bb* докрай (по пътя 043), но попада в нефиналното състояние 3, затова не разпознава *bb*. А пък *ac* изобщо не се съдържа в *w* като подниз, тоест като подредица от последователни знаци (само началото *a* се съдържа в *w*), поради което автоматът успява да прочете само буквата *a* (по пътя 01), след което блокира, тъй като от състоянието с номер 1 не излиза преход, надписан с буквата *c*.

Дървото е нужно само за построяването на автомата, но не и за неговото използване при разпознаване на наставки. От друга страна, дървото съдържа информация, свързана със следната релация между думи. За всяка дума α се дефинира множество endpos(α), състоящо се от всички позиции, на които думата α се среща в *w* (ако α не е подниз на *w*, то множеството е празно). Става дума за позицията на последния знак от α. Позициите са номерирани с целите числа от 1 до *n* включително. Така например за думата *w* = *abbcb*: endpos(*bbc*) = endpos(*bc*) = endpos(*c*) = {4}; endpos(*abb*) = endpos(*bb*) = {3}, обаче endpos(*b*) = {2 ; 3 ; 5}; endpos(*ac*) = ∅. За празната дума ε се приема по определение, че endpos(ε) = {0 ; 1 ; 2 ; . . . ; *n*}, като невъзможната позиция 0 цели да направи това множество неповторимо. В множеството на всички думи (над входната азбука на автомата) се дефинира следната двучленна релация *R*: $\alpha \, R \, \beta \Leftrightarrow$ endpos(α) = endpos(β). Не е трудно да се провери, че релацията *R* е рефлексивна, симетрична и транзитивна; т.е. *R* е релация на еквивалентност.

Следните свойства са очевидни:

1) Ако от две различни думи α и β никоя не е наставка на другата, то endpos(α) $\cap$ endpos(β) = $\varnothing$.
2) Ако β е наставка на α и β е по-къса от α (тоест те са различни), то endpos(α) $\subseteq$ endpos(β). Равенство има $\Leftrightarrow$ β се среща в w само като наставка на α.

Класовете на еквивалентност на релацията R могат да бъдат отъждествени с всевъзможните множества endpos. Те са подмножества на $\{1 ; 2 ; \ldots ; n\}$ с изключение на endpos(ε) = $\{0 ; 1 ; 2 ; \ldots ; n\}$, обаче обратното не е вярно: не всяко подмножество на $\{1 ; 2 ; \ldots ; n\}$ може да се представи във вида endpos(α). Така например множеството $\{4 ; 5\}$ не съответства на никой клас на еквивалентност на релацията R, породена от думата $w = abbcb$, защото не е възможно един и същи подниз да завършва едновременно на четвъртата и на петата буква от думата (не може последният знак на подниза да бъде едновременно c и b).

Ако $\alpha \, R \, \beta$, на думите α и β може да се съпостави едно и също състояние на автомата. Действително, щом endpos(α) = endpos(β), то за всяка дума γ: $\alpha\gamma$ е наставка на w $\Leftrightarrow$ $\beta\gamma$ е наставка на w. Съществува поне една дума γ, за която $\alpha\gamma$ и $\beta\gamma$ са наставки на w $\Leftrightarrow$ endpos(α) = endpos(β) $\neq \varnothing$. Наистина, ако произволно избрана дума γ притежава дължина k, то $\alpha\gamma$ е наставка на w $\Leftrightarrow n - k \in$ endpos(α) $\Leftrightarrow n - k \in$ endpos(β) $\Leftrightarrow$ $\beta\gamma$ е наставка на w. Очевидно е, че ако endpos(α) = endpos(β) = $\varnothing$, то не съществува дума γ с желаното свойство: щом думите α и β не са поднизове на w, то $\alpha\gamma$ и $\beta\gamma$ също не са поднизове, а значи не са и наставки на w. И обратно, ако endpos(α) = endpos(β) $\neq \varnothing$, то съществува дума γ с желаното свойство, а именно думата, съставена от последните k знака на w, където $n - k$ е най-големият елемент на endpos(α); в този случай думите α и β могат да споделят едно състояние на автомата, защото каквато и дума γ да бъде прочетена после от входната лента, $\alpha\gamma$ и $\beta\gamma$ едновременно са или не са наставки на w, тоест автоматът може да премине в едно и също състояние (финално или не).

Ако ли пък думите α и β не се намират в релацията R, то на тях не може да се съпостави едно и също състояние на автомата. Действително, след като endpos(α) $\neq$ endpos(β), то поне едното множество съдържа елемент r, който не принадлежи на другото множество, затова думата γ, съставена от последните $n - r$ знака на w, е такава, че от думите $\alpha\gamma$ и $\beta\gamma$ само едната е наставка на w (а именно онази от двете думи $\alpha\gamma$ и $\beta\gamma$, чието множество endpos съдържа r). Следователно, ако се допусне, че след прочитане на думите α и β автоматът се оказва в едно и също състояние, то след прочитането на думата γ, започвайки от това състояние, автоматът ще се озове в състояние, което едновременно е и не е финално, а това е противоречие.

И така, състоянията на автомата са поне толкова, колкото са класовете на еквивалентност на релацията R. Автоматът е минимален, в смисъл че има възможно най-малко състояния — колкото са класовете на еквивалентност на релацията R. Всъщност този брой е минимален само ако е задължително автоматът да е напълно определен. Ако обаче се допуска автоматът да блокира, то класът на еквивалентност, съответстващ на празното множество endpos (самият той не е празен), може да се изтрие, защото съответното състояние е поглъщащо (щом една дума α не е подниз на w, то и $\alpha\gamma$ не е подниз на w, каквато и да е думата γ). След изтриването на това състояние и преходите, водещи към него, автоматът отхвърля въпросните думи чрез блокиране, а не чрез попадане в нефинално състояние. Така не се губят думи за разпознаване, защото изтритото състояние е нефинално: финални са точно онези състояния, чието съответно множество endpos съдържа n, а множеството $\varnothing$ не съдържа n.

Празната дума ε е сама в своя клас на еквивалентност: само нейното множество endpos съдържа нулата, затова то не съвпада с множеството endpos на никоя друга дума. Добавянето на нулата е нужно, защото, ако думата w се състои от една буква, повторена неколкократно, например $w = aaaaa$, то endpos(a) = $\{1 ; 2 ; \ldots ; n\}$, а endpos(ε) трябва да е по-широко множество.

От свойство 2 следва, че ако от някоя дума започнат да се изтриват буквите отляво надясно, една по една, то след всяко изтриване множеството endpos или ще се запазва, или ще се разширява. Последователните думи от редицата, които имат едно и също множество endpos, са от един клас на еквивалентност и на тях съответства едно и също състояние на автомата. Разширяването на множеството endpos съответства на преминаване към родител в дървото. Например думата $w = abbcb$ поражда автомата и дървото, изобразени на картинката по-горе, а думата abb поражда чрез изтриване на букви редицата

$$\text{endpos}(abb) = \text{endpos}(bb) = \{3\} \subset \text{endpos}(b) = \{2 ; 3 ; 5\} \subset \text{endpos}(\varepsilon) = \{0 ; 1 ; 2 ; 3 ; 4 ; 5\}.$$

На тази редица от множества съответства (след премахване на повторенията) следната редица от върхове на дървото (и състояния на автомата): 3 ; 4 ; 0. В дървото родителят на връх № 3 е № 4, а родителят на № 4 е № 0.

За всяко състояние алгоритъмът пази цяло неотрицателно число len, равно на дължината на най-дългата дума в класа на еквивалентност на това състояние. Очевидно за всеки автомат $len(q_0) = 0$ (това е дължината на празната дума). За автомата по-горе $len(3) = 3$ (понеже най-дългата дума в състояние 3 е abb), $len(4) = 1$ (най-дългата дума в състояние 4 е b). Числата len не са написани на картинката. Обратно, за прегледност са изобразени самите най-дълги думи, ала компресирано. Дървото е нарисувано с указатели към преките наследници и на всяка стрелка е написана дума, съставена от една или повече букви. Най-дългата дума в дадено състояние се получава при слепването на думите от ребрата на дървото при преход от корена до въпросния връх и обръщане на крайния резултат. Например при слепване на думите b и ba се получава bba, което след обръщане дава abb — най-дългата дума в състояние № 3.

Следователно дължината на думата, с която е надписано някое ребро на дървото, е равна на разликата между стойностите *len* в краищата на реброто, а тази разлика на свой ред е равна на броя на думите в онзи край на реброто, който е наследник на другия край. Например реброто между върховете 3 и 4 на картинката по-горе е надписано с думата *ba* (всъщност *ab* след обръщане), чиято дължина е 2, и състоянието 3 съдържа две думи (*abb* и *bb*), а разликата $len(3) - len(4) = 3 - 1 = 2$ (по определение това е точно разликата на дължините на най-дългите думи от двете състояния). Разликата винаги е цяло положително число; за алгоритъма има важно значение дали то е равно, или е по-голямо от 1.

Автоматът има възможно най-малко състояния, защото всяко едно от тях съответства на клас на еквивалентност на релацията *R* (без класа, съответстващ на празното множество endpos). След всяко прочитане на знак от думата *w* алгоритъмът създава едно или две състояния. Като се вземе предвид и q_0, става ясно, че броят на състоянията е най-малко $n + 1$ (тази бройка се достига например тогава, когато всички знаци на думата *w* са два по два различни). Действително, всички представки на *w*, включително *w* и ε, принадлежат на различни класове на еквивалентност, защото имат различни множества endpos, а това е така, защото най-малките елементи на множествата endpos са различни: най-малкият елемент на множеството endpos на всяка представка на *w* е равен на дължината на представката, а тези дължини са две по две различни (това са всички цели числа от 0 до *n* вкл.). Следователно бройката $n + 1$ на състоянията, която алгоритъмът постига при определени условия, не може да бъде намалена.

От друга страна, състоянията не могат да бъдат повече от $2n - 1$ при $n \geq 2$ (при $n = 1$ състоянията са точно две, а при $n = 0$ има едно състояние — q_0). Наистина, отначало има само едно състояние, след прочитането на първия знак се добавя още едно, а след прочитането на втория знак — пак едно (никога две). При прочитането на всеки от останалите $n - 2$ знака се добавят най-много две нови състояния, ето защо общият им брой не надхвърля $3 + 2(n - 2) = 2n - 1$. Тази горна граница се достига единствено за думите от вида $w = abbbbb \ldots b$.

Че всеки конкретен брой състояния между $n + 1$ и $2n - 1$ включително, постигнат от алгоритъма, е минималният възможен за дадена дума *w*, следва от това, че на всяка стъпка алгоритъмът добавя само толкова нови състояния, колкото е необходимо. Наистина, едно ново състояние е нужно за току-що прочетената представка на *w*. Старите състояния продължават да съответстват на представките, прочетени по-рано. Класът на новото състояние *last* съдържа цялата прочетена до момента част от *w*, затова към него се добавя преход (с току-що прочетената буква *x*) от бившето състояние *last* (сега именувано *p*). Всички доскорошни наставки, удължени със знака *x*, стават отново наставки, затова от предишните финални състояния се добавят преходи с *x* към текущото (последното) състояние *last*. То съдържа всички нови наставки, затова негов родител в дървото става коренът. За това се грижи първият от циклите **while** в процедурата *add_letter* и операторът **if** $q = last$**: assign** $link(last) = q_0$.

Последното разсъждение е безпроблемно, ако знакът x се среща за пръв път в думата w. В противен случай, за да остане автоматът детерминиран, цикълът спира изпълнението си при първото достигане на бивше финално състояние p, от което вече излиза преход със знака x към някакво състояние q. Само дотук обходените от цикъла бивши финални състояния имат нужда от преходи с x към *last*, затова само те ги получават (и съответните наставки са в класа *last*). Ако α е най-дългата дума (и бивша наставка) от състоянието p, то αx е съответната нова наставка, за която обаче не е нужен преход към *last*, защото вече има преход към q. Не е нужно ново състояние, затова алгоритъмът не създава такова, а само прави q родител на *last* в дървото. Точно това е нужно, защото думата αx се получава чрез изтриването на първата буква от последната нова наставка, добавена към *last* чрез последния добавен преход (това следва от командата p = *link(p)* в първия цикъл **while**). По-късите наставки на прочетената дотук част от w са наставки на тази нейна наставка, тоест на αx, значи към тях вече има (от предишните стъпки) верига от указатели *link*, започваща от q, вече предшестващо *last*. Така всички тези състояния са гарантирано финални (към текущия момент).

Разсъжденията в предишния абзац показват, че (чрез текущите си финални състояния) автоматът разпознава всички наставки на прочетената дотук част от думата w. Те не гарантират, че автоматът няма да разпознае думи, които не бива да разпознава. По отношение на старите преходи и указатели *link* не може да има проблем: те са били добавени правилно при някое предишно изпълнение на процедурата *add_letter*. Тук в неявен вид се ползва математическа индукция по броя на прочетените до момента букви от w; фактически се провежда индуктивна стъпка, а базата е очевидна: отначало релацията R има само един клас на еквивалентност, който съдържа непразно множество endpos, и това е множеството $\{0\}$, т.е. алгоритъмът правилно инициализира автомата и дървото с единствен връх — q_0.

Проблем може да възникне с новия указател *link* — този от *last* към q. Тъй като α е най-дългата дума от състоянието p, то дължината на α е равна на *len(p)*. А понеже αx е дума от q, то $len(q) \geq len(p) + 1$. Ако $len(q) = len(p) + 1$, то αx е най-дългата дума от q, следователно q и неговите предшественици в дървото съдържат само наставки на αx, а те са наставки и на прочетената дотук част от w. Затова в този случай не възниква проблем с разпознаването на думи, които не са наставки.

Ако обаче $len(q) > len(p) + 1$, то q съдържа думи, по-дълги от αx. Тогава αx е тяхна наставка, следователно те завършват със знака x. Те не могат да са наставки на прочетената дотук част от w, защото би се оказало, че тя притежава по-дълги наставки, завършващи с x, които биха били открити по-рано от първия цикъл **while** (тоест p не би било първото състояние, от което излиза преход с x). Така състоянието q съдържа думи, които са, и думи, които не са наставки на прочетената дотук част от w, а те имат различни множества endpos (едното

съдържа броя на прочетените дотук знаци от w, а другото — не), следователно тези думи не могат да принадлежат на един клас на еквивалентност на R, а се налага разделянето им в различни класове. Състоянието q се клонира, тоест създава се негово копие cl, като в cl се отделя αx и нейните наставки от q, а в q остават думите, по-дълги от αx. Затова cl става родител на q (тоест cl се вмъква в дървото между q и бившия родител на q) и оттук нататък cl поема ролята, която по-горе играеше q, затова cl става родител и на $last$. Остава само да се пренасочат към cl всички преходи с x, които дотогава са влизали в q и са пораждали думата αx и нейните наставки. Това са точно преходите от p и неговите предшественици, и то само първите няколко: престанат ли веднъж преходите с x към q, значи те са се насочили към по-къси наставки, тоест към предшественици на q и няма никога повече да се върнат към него, ето защо вторият цикъл **while** спира проверките при първата липса на подходящ преход.

Например след прочитане на втората буква на думата $w = abbcb$ се получава автомат с три състояния, означени по-горе с 0, 1 и 2 (0 е началното състояние). Състоянието с номер 2 съдържа думите b и ab, като и двете са наставки към текущия момент (тоест това състояние е финално). С прочитане на третия знак положението се променя. Сега b е наставка на abb, но ab не е наставка на abb. Състоянието с номер 2 не може хем да е финално, хем да не е финално, затова се разделя на две състояния: думата ab остава в него (то престава да е финално), а думата b се отделя в ново състояние (с номер 4), което става финално.

Анализът на всички възможни случаи показа коректността на алгоритъма, както и минималността на броя на състоянията: алгоритъмът създава състояние само когато възниква нов клас на еквивалентност, затова след всяко изпълнение на процедурата *add_letter* броят на състоянията е равен на броя на класовете на еквивалентност на релацията R, породена от прочетената до момента част от думата w.

По-горе беше показано, че броят на състоянията на автомат е между $n + 1$ и $2n - 1$, където n е дължината на думата w, по която се строи автоматът. Има горна граница и за броя на преходите: той не надхвърля $3n - 4$ при $n \geq 3$ (когато n е 0, 1 или 2, максималният брой преходи е съответно 0, 1 или 3).

Наистина, при всеки преход числото len нараства с поне една единица, затова няма цикли, тоест автоматът е ориентиран ацикличен граф. Нека T е множеството на най-дългите пътища в този граф. Тъй като всеки връх съдържа точно една най-дълга дума, T съдържа всички върхове, но не и цикли (дори неориентирани). По-точно, T съдържа точно един път от q_0 до кой да е връх на графа, тоест T е покриващо дърво и q_0 е коренът на T. Понеже всяка част от най-дълъг път също е най-дълъг път, то len нараства с единица по всички ребра на T. Обратно, всяко ребро, по което len нараства с единица, е от T, защото иначе би породило втори най-дълъг път (и оттам втора най-дълга дума) за върха, в който влиза. Ребрата, по които len расте с 1, са точно ребрата на T, затова броят им е равен на броя на върховете минус 1, т.е. не повече от $2n - 2$.

Нека $p \xrightarrow{x} q$ е преход, за който $len(q) > len(p) + 1$. Освен това нека α е думата, прочетена по най-дълъг път до p, следователно от корена на T, тоест от началното състояние. Състоянието q (като всяко друго) съответства на някакъв клас на еквивалентност на релацията R, тоест всички думи от q, включително думата αx, имат едно и също множество endpos. На празното множество endpos по построение не е съпоставено никакво състояние, тогава множеството endpos(αx), съответстващо на състоянието q, е непразно. Нека k е произволно число от множеството endpos(αx). Следователно αx е наставка на думата, образувана от първите k знака на w, а пък $\alpha x\, x_{k+1}\, x_{k+2}\, x_{k+3} \ldots x_n$ е наставка на w. Автоматът може да прочете тази наставка и да я разпознае, попадайки в някакво финално състояние. Пътят, по който я чете, изглежда така: по най-дълъг път от началното състояние q_0 до състоянието p, после по прехода от p към q, след това по някакъв път от q до съответното финално състояние. Първата част от пътя (тази от q_0 до p) се състои само от ребра на дървото T, затова преходът от p към q е първият, при който len се увеличава с повече от 1. Въпреки че е възможно път като описания да съдържа няколко такива прехода, все пак ясно е кой преход е породил пътя — това е първият преход, при който len се увеличава с повече от 1. Следователно на различни преходи от този вид съответстват различни пътища, а оттам и различни наставки (една наставка не може да бъде прочетена по два различни пътя: автоматът е детерминиран). Затова броят на преходите, при които числото len нараства с повече от единица, не надхвърля броя на различните наставки на w, тоест $n + 1$. От тях е нужно да се изключи празната наставка (защото нейният път има нулева дължина, тоест не съдържа никакви преходи) и самата дума w (защото по нейния път има само такива преходи, при които len се увеличава с единица: len расте от 0 до n в резултат на n прехода). Така остават $n - 1$ възможни наставки, тоест броят на преходите, при които len нараства с повече от единица, не надхвърля $n - 1$.

Преходите от двата вида са общо не повече от $(2n - 2) + (n - 1) = 3n - 3$. Но за да се достигне този брой, трябва да се достигне максимум едновременно при двете събираеми, тоест поотделно за всеки от двата вида преходи. Броят на преходите от първия вид достига $2n - 2$ само когато броят на състоянията достига $2n - 1$, тоест само при $w = abbbbb \ldots b$. Но за такава дума w броят на всички преходи е $2n - 1 < 3n - 3$. Следователно броят $3n - 3$ е недостижим. Значи броят на преходите не надхвърля $3n - 4$ (при $n \geq 3$). Този брой се достига при $w = abbb \ldots bbbc$.

Изведените линейни горни граници за броя на състоянията и преходите могат да послужат за анализ на времевата сложност на предложения алгоритъм. Повечето команди на процедурата *add_letter* се изпълняват само по веднъж при отделно нейно извикване, а тъй като тя се извиква общо n пъти, това прави сумарно $\Theta(n)$ операции. Има само три места в алгоритъма, които биха могли да доведат до повече операции.

Едно място е първият цикъл **while**. При отделно извикване на *add_letter* той може да е бавен, но не и сумарно. Всяко изпълнение на тялото на цикъла създава един преход в автомата и тъй като преходи не се премахват никога, а броят им не надхвърля $3n - 4$, то общият брой операции на този цикъл е $\mathrm{O}(n)$.

Друго място е присвояването **assign** $\delta(cl) = \delta(q)$, което е кратко записан цикъл. То копира преходите от q, тоест създава преходи, затова за него важи същото разсъждение и същият извод: общият брой операции, извършени от него за всички извиквания на процедурата *add_letter*, е $\mathrm{O}(n)$.

Третото място е последният цикъл **while**. Той не създава, а само пренасочва вече създадени преходи, поради което за него не важи разсъждението по-горе. Нека w_i е думата, съставена от първите i знака на w. Нека p_i е състоянието, в което автоматът се озовава, след като прочете w_i, започвайки от q_0. (Не може автоматът да блокира при четенето на w_i, тъй като тя е част от наставка на w, а именно самата дума w.) При дефинирането на p_i се има предвид автоматът в своя окончателен вид, но тъй като автоматът се мени по време на изпълнение на алгоритъма, са нужни обозначения и за отделните етапи от този процес. Нека Q_i е множеството от състоянията на автомата непосредствено след добавянето на i-тата буква от w, а пък $S_i(u)$ е множеството от състояния в Q_i, които съответстват на наставките на някоя дума u (това не е някоя определена дума, а променлива, която приема ту една, ту друга стойност).

Очевидно $S_i(w_i)$ се явява множеството от финалните състояния на автомата, какъвто е построен след прочитането на първите i букви от w. Ако $q \in S_i(w_i)$, $q \neq q_0$ и $i \geq 1$, то състоянието q съответства на поне една непразна дума (някоя наставка на w_i), която следователно има вида αx_i, където x_i е i-тата буква на w, а пък α може да е и празната дума. След прочитането на думата αx_i автоматът по определение се озовава в състоянието q, ето защо след прочитането на α (която е наставка на w_{i-1}) и непосредствено преди прочитането на x_i автоматът се озовава в някакво състояние $p \in S_i(w_{i-1})$, от което има преход със знака x_i към състоянието q. Индексът i на S напомня, че става дума за незавършения автомат — този, който се получава след края на извикването *add_letter*(x_i). Именно за него се разсъждава как би прочел една или друга дума.

От разсъжденията в предишния абзац следва, че на всяко $q \in S_i(w_i) \setminus \{q_0\}$ може да се съпостави (както е описано там) състояние $p \in S_i(w_{i-1})$. При това може да се случи на едно q да се съпоставят няколко p-та: едно и също q може да съдържа няколко непразни думи, които да се получават от различни p-та. Но от едно p не може да излизат няколко прехода с една и съща буква (x_i), защото автоматът е детерминиран (на всеки етап от построението). Ето защо q-тата, различни от q_0, са не повече от p-тата, тоест $|S_i(w_i) \setminus \{q_0\}| \leq |S_i(w_{i-1})|$. Горната граница може да се намали още, както предстои да установим.

Ако по време на извикването *add_letter*(x_i), $i \geq 1$, при изпълнение на тялото на последния цикъл **while**, се пренасочва преход, който преди пренасочването сочи от p към q и е отбелязан със знака x, то $x = x_i$ и знакът x се запазва, обаче след пренасочването преходът вече е от p към cl. От инициализирането на p с бившето състояние *last* и от присвояванията $p = link(p)$ в двата цикъла **while** се вижда, че променливата p обхожда само предишни финални състояния, т.е. $p \in S_{i-1}(w_{i-1})$, затова съществува наставка α на w_{i-1}, след чието прочитане предишният автомат (този след края на извикването *add_letter*(x_{i-1}), ако $i > 1$, или този, който се състои само от началното състояние, ако $i = 1$) се озовава в p.

Тъй като $q \in Q_{i-1}$ и преходът от p към q е отбелязан със знака $x = x_i$, то споменатият предишен автомат, започвайки от началното състояние, прочита думата $\alpha x = \alpha x_i$ и се озовава първо в състоянието p, след което преминава по описания преход в състоянието q и именно то е резултатът от прочитането на думата $\alpha x_i = \alpha x$, която се явява наставка на w_i. Следователно $q \in S_{i-1}(w_i)$.

По време на последния цикъл **while** преходи от вида $p \to q$ се пренасочват и стават $p \to cl$. Очевидно $len(q) > len(p) > len(link(p)) > len(link(link(p))) > \ldots$ Ето защо състоянията p са недостижими от q, затуй пътищата, по които се четат думите α, не се променят, тоест съответният път пак завършва в p. Следва, че $p \in S_i(w_{i-1})$. Обаче пътищата, по които се четат думите $\alpha x_i = \alpha x$, се променят заради пренасочването на преходите: сега последният им преход е от p към cl, а не от p към q. Затова $cl \in S_i(w_i)$. А понеже се пренасочват всички преходи с буквата $x = x_i$, които преди са влизали в q, то $q \notin S_i(w_i)$.

Броя на пренасочените преходи (имащи вида $p \to cl$ след пренасочването) да означим със z_i. Очевидно $cl \neq q_0$ (състоянието cl е било създадено току-що, а състоянието q_0 — още преди първото извикване на *add_letter*). Щом $cl \neq q_0$ и $cl \in S_i(w_i)$, то $cl \in S_i(w_i) \setminus \{q_0\}$. В предишния абзац стана ясно, че $p \in S_i(w_{i-1})$. Затова всички преходи $p \to cl$ са от вида, разгледан в края на предишната страница. Понеже към единственото състояние cl сочат z_i прехода от различни p-та от $S_i(w_{i-1})$, то всички $z_i - 1$ състояния p-та след първото сочат към вече посоченото състояние $cl \in S_i(w_i) \setminus \{q_0\}$, не към някое непосочено. Следователно неравенството $|S_i(w_i) \setminus \{q_0\}| \leq |S_i(w_{i-1})|$ може да се усили по следния начин: $|S_i(w_i) \setminus \{q_0\}| \leq |S_i(w_{i-1})| - (z_i - 1)$, тоест $|S_i(w_i) \setminus \{q_0\}| \leq |S_i(w_{i-1})| - z_i + 1$.

Празната дума е наставка, затова началното състояние винаги е финално, тоест $q_0 \in S_i(w_i)$, следователно $|S_i(w_i) \setminus \{q_0\}| = |S_i(w_i)| - 1$. След заместване на този израз в неравенството то приема вида $|S_i(w_i)| - 1 \leq |S_i(w_{i-1})| - z_i + 1$, откъдето следва, че $z_i \leq |S_i(w_{i-1})| - |S_i(w_i)| + 2$.

С това е получена горна граница за броя z_i на преходите, пренасочени от последния цикъл **while** по време на извикването *add_letter*(x_i).

Тъй като преходи не се премахват, а само се пренасочват, то състоянията, разпознаващи наставките на w_{i-1}, не се променят, с евентуалното изключение на състоянията q, *last* и *cl* от последния цикъл **while** (като q може да изгуби свойството да разпознава наставки на w_{i-1}, а новите състояния *last* и *cl* може да придобият това свойство). Следователно $|S_i(w_{i-1})| \leq |S_{i-1}(w_{i-1})| + 2$.

След като току-що доказаното неравенство се замести в горната граница на броя на пренасочените преходи, се получава нова горна граница за броя им: $z_i \leq |S_{i-1}(w_{i-1})| - |S_i(w_i)| + 4$.

След сумиране по i се получава горна граница за общия брой на преходите, пренасочени от всички изпълнения на процедурата *add_letter*, а тя се явява горна граница и на броя на изпълненията на тялото на последния цикъл **while**:

$$
\begin{aligned}
z_1 + z_2 + \ldots + z_n \leq\ & |S_0(w_0)| - \cancel{|S_1(w_1)|} + 4\ + \\
& \cancel{|S_1(w_1)|} - \cancel{|S_2(w_2)|} + 4\ + \\
& \cancel{|S_2(w_2)|} - \cancel{|S_3(w_3)|} + 4\ + \\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ +\\
& \cancel{|S_{n-1}(w_{n-1})|} - |S_n(w_n)| + 4,
\end{aligned}
$$

тоест тялото на последния цикъл **while** от процедурата *add_letter* се изпълнява най-много $4n + |S_0(w_0)| - |S_n(w_n)| \leq 4n$ пъти. Последното неравенство важи, защото $|S_0(w_0)| = |S_0(\varepsilon)| = 1$ (при инициализирането на автомата има само едно състояние — началното, и то разпознава празната наставка), а същевременно $|S_n(w_n)| \geq 1$ (накрая автоматът има поне едно финално състояние — *last*).

Направеният анализ показва, че автоматът на наставките се строи с помощта на $\Theta(n)$ операции при всякакви входни данни, а амортизираният брой операции на неявния и двата явни цикъла за едно извикване на процедурата *add_letter* е $\Theta(1)$.

Броят на операциите ще бъде времева сложност, при условие че всяка от тях се изпълнява за константно време. Проблем може да възникне единствено с функцията на преходите. Тя може да се реализира за константно време, ако преходите се пазят в масив, чийто индексен тип съвпада с входната азбука. В такъв случай времето за построяване на автомата е $\Theta(n)$, а нужната памет е от порядък $\Theta(m)$ за всяко състояние и $\Theta(m\,n)$ общо за целия автомат, където m е размерът на азбуката: за всяко състояние се пази масив от m указателя, които представляват преходите от това състояние (ако за някоя буква липсва преход, съответният указател е нулев). В оценката $\Theta(n)$ за времето за построяването на автомата е пренебрегнато времето за инициализиране на масивите с нулеви указатели. Ако се добави и то, оценката за времето за построяване на автомата ще стане от порядък $\Theta(m\,n)$. Само $\Theta(k)$, без множител m, ще бъде порядъкът на времето за прочитане на дума от k знака, след като автоматът е построен.

Има друга реализация: действителните преходи от всяко състояние се пазят в двоично дърво за търсене. Тогава необходимата памет е пропорционална на броя на състоянията и преходите, т.е. $\Theta(n)$, и не зависи от размера на азбуката.

Втората реализация пести памет, като не пази нулеви указатели. Преходите се пазят в двоично дърво за търсене; за ключове служат буквите на преходите. По този начин обаче се губи повече време за търсене на преходите. Това време е пропорционално на височината на съответното двоично дърво за търсене. Тя може да се намали с помощта на балансирани двоични дървета. Не е нужен идеален баланс, стига частното от дължините на най-дълъг и най-къс път да е ограничено отгоре от някаква константа. В такъв случай времето за търсене на преход от дадено състояние е пропорционално на логаритъма от броя на преходите, тоест $O(\log m)$. Времето за построяването на автомата е $O(n \log m)$. След като автоматът е построен, времето за прочитане на k-буквена дума ($k \leq n$) е $O(k \log m)$, като тази оценка, за разлика от предишната, се подобрява лесно: ако k е много по-малко от n, то полученото време $O(k \log m)$ и така не е голямо; ако ли пък k е от порядъка на n, то оценката $O(k \log m) = O(n \log m)$ може да се замени с по-малка, като се съобрази, че е невъзможно всички състояния да имат m изходящи прехода, нито близък до този брой (заради горната граница на броя на всички преходи в автомата). В последния случай добра оценка се получава със следния подход. Времето за прочитане на дадена k-буквена дума е равно на $c \log b_1 + c \log b_2 + \ldots + c \log b_k$, където $b_1, b_2, \ldots, b_k$ са бройките преходи от всички състояния, преминати при четенето на думата, без последното. Може да се добавят липсващите събираеми и да се получи следната горна граница: $c (\log b_1 + \log b_2 + \ldots + \log b_K)$, където K е броят на състоянията на автомата. От неравенството между средното аритметично и средното геометрично следва, че времето за прочитане на дадена дума (след като автоматът е построен) $\leq$ $c (\log b_1 + \log b_2 + \ldots + \log b_K) = c \log (b_1 b_2 \ldots b_K) \leq c \log [(b_1+b_2+\ldots+b_K)/K]^K$ $\leq c \log [(3n-4)/(n+1)]^{2n-1} = c\,(2n-1) \log [\Theta(1)] = \Theta(n)$, като константата c представлява някакво характерно време за изпълнението на отделна команда (например прочитането на стойността на елемент на масив по даден индекс) и не зависи от размера m на азбуката. Следователно времето за прочитането на дадена дума е $O(n)$. От двете получени оценки — $O(k \log m)$ и $O(n)$ — трябва да се вземе по-малката, тъй като важат и двете (а те са горни граници).

Забележка: Предполага се, както обикновено, че основата на логаритъма е произволна, но по-голяма от 1. Точната стойност не е важна за порядъка.

Често се приема, че размерът m на входната азбука е константа. Тогава всички изчислени сложности по време и по памет (и за двете реализации) се свеждат до $\Theta(n)$ — памет за автомата и време за построяването на автомата, и $\Theta(k)$ — време за прочитането на дадена k-буквена дума. По този начин се получават линейни порядъци за сметка на пренебрегването на множителите m и $\log m$. От практическа гледна точка е необходимо те да са малки числа. За логаритъма това обикновено е така. Обаче отнапред не е никак сигурно, че множителят m е пренебрежимо малък: входната азбука може да е голяма. Например кодът ASCII съдържа 256 знака, докато версията 15.0 на Unicode (от септември 2022 година) съдържа 149186 знака.

Автоматът на наставките има множество приложения:

1) *Проверка за подниз*. Даден е текст T и едно множество P от низове. Трябва да се провери кои низове от P се срещат в T (и къде точно се срещат). Решението е да се построи автомат на наставките на T, после за всеки низ от P да се направи опит да бъде прочетен от автомата. Ако низ блокира автомата, то низът не се съдържа в T (множеството endpos на проверявания низ е празно). Ако низ бъде прочетен докрай от автомата (без значение дали накрая автоматът се озовава във финално състояние), то прочетеният низ се явява подниз на T, понеже неговото множество endpos е непразно. Множеството endpos на онова състояние, в което автоматът завършва четенето, показва всички позиции, на които се среща провереният низ.

Забележка: Ако автоматът блокира след прочитането на k-тия знак от низа, то най-дългата представка на низа, която е подниз на T, има дължина k.

2) *Брой на различните поднизове*. Даден е низ S. Търси се броят на всички различни поднизове на S. Решението е да се построи автомат на наставките на S и да се преброят пътищата, започващи от началното състояние на автомата. Всеки такъв път съответства на подниз на S и обратно. Тъй като автоматът е ориентиран ацикличен граф, то броят на пътищата, започващи от даден връх, се намира с динамично програмиране. По-точно, върху графа се извършва топологично сортиране, после върховете се обхождат отзад напред и на всеки връх се приписва число по следния начин: 1 + сбора на числата във върховете, към които има преходи от текущия връх. Ако няма преходи, сборът е празен, т.е. нула. Числото във всеки връх е броят на пътищата, започващи от него; сборът е равен на броя на пътищата, започващи от текущия връх и минаващи през някой от върховете, към които има преходи от него; единицата брои пътя, съставен само от текущия връх, без преходи (път с дължина нула). Най-накрая се пресмята числото в началното състояние; то показва броя на всички пътища, започващи от него; това е също броят на всички поднизове на S.

3) *Обща дължина на различните поднизове*. Даден е низ S. Търси се общата дължина на всички различни поднизове на S. За целта се строи автомат на наставките на S, прилага се топологично сортиране, както по-горе. След това за всеки връх се пресмятат отзад напред две числа: броят на пътищата, които започват от този връх, и сборът от дължините им. Първото число се пресмята, както е обяснено в т. 2. Второто число на всеки връх е сборът от всички числа (и първите, и вторите) на върховете, към които има преход от текущия връх: към дължините на пътищата, започващи от върховете, към които има преход, се добавя единица заради прехода — толкова единици, колкото са пътищата.

4) *Първото по азбучен ред кръгово отместване* на даден низ S се търси, като се построи автомат на наставките на $S + S$ (той съдържа всички кръгови отмествания на S) и от началното състояние се направят $|S|$ прехода алчно, тоест на всяка стъпка се избира преходът с възможно най-малката буква.

5) *Търсене на k-тия подниз по азбучен ред*. Даден е низ S. Търси се k-тият подниз на S по азбучен ред. За целта се построява автомат на наставките на S и се решава задачата от т. 2. След това, започвайки от началното състояние, алгоритъмът обхожда преходите му по азбучен ред на буквите на преходите и сумира бройките на поднизовете по всички преходи (тези бройки са написани във върховете, към които отиват преходите). Сумата се натрупва, докато стане равна или по-голяма от k. Алгоритъмът минава по последния сумиран преход към ново състояние, изважда предпоследната сума от k и наново решава същата задача за новото състояние и намаленото число k.

6) *Брой срещания на подниз*. Даден е текст T и множество P от низове. За всеки низ от P се търси колко пъти се среща в T. Тази задача се решава, като по дадения низ T се построява автомат на наставките, а после всеки низ от P се подава на автомата за четене. Ако той блокира, значи съответният низ от P не е подниз на T. Ако подаденият низ от P бъде прочетен от автомата докрай, значи е подниз на T и се среща в T толкова пъти, колкото е броят на елементите на множеството endpos на състоянието, в което автоматът е завършил четенето. Мощностите на множествата endpos се пресмятат предварително, веднага след построяването на автомата. Във всяко състояние се записва отначало числото 1 с изключение на състоянията, получени чрез клониране (в тях се записва нула). След това се обхожда от листата към корена дървото на автомата, образувано от указателите *link*, сочещи от всеки връх към неговия родител. По време на обхождането числото от всеки връх (може би увеличено на предишните стъпки) се прибавя към числото на родителя.

Обяснение на процедурата: Множествата endpos на преките наследници на един и същи връх q не се пресичат две по две, а множеството endpos на q е тяхното обединение (оттук сумирането) плюс може би допълнителни елементи. Най-дългият низ на q може да се срещне (без да е наставка на по-дълъг низ) още само в началото на T. Затова се прибавя единица, а тъй като T се чете отляво надясно, то q е създаден преди преките си наследници (не е клониран). Обратно, ако най-дългият низ на q не се среща в началото на T, не се прибавя 1; в този случай q е бил създаден след някой пряк наследник, тоест бил е клониран от него. Коренът се създава преди всички други върхове (не чрез клониране), затова и при него е налице събираемо 1; то съответства на нулевата позиция в endpos на корена (празният низ се среща и преди първия знак на T).

7) *Първо срещане на подниз*. Даден е текст T и множество P от низове. За всеки низ от P се търси първото му срещане в T (указано чрез позицията на последния знак). За целта се строи автомат на наставките на T и се изчислява първият елемент на всяко множество endpos (изчислението се прави в момента на създаване на всяко състояние): числото *len*, ако състоянието не е клонирано; иначе първият елемент на endpos на състоянието, от което се клонира текущото. След създаването на автомата се прочита всеки низ от P; резултатът е първият елемент на множеството endpos на състоянието, в което е завършило четенето.

8) *Всички срещания на подниз*. Даден е текст T и множество P от низове. За всеки низ от P се търсят всичките му срещания в T (указани чрез позициите на последния знак). За целта първо се построява автомат на наставките на T и се изчисляват (както в т. 7) първите, тоест най-малките, елементи на всички множества endpos. След това низовете от P се четат един по един и за всеки низ се отбелязва състоянието, в което е завършило четенето. Това състояние пази най-малкия елемент на своето множество endpos; този най-малък елемент е позицията на първото срещане на подниза. Всяко друго срещане е като наставка на по-дълъг низ, затова другите срещания се получават с обхождане на преките и косвените наследници на състоянието в дървото на автомата. Най-малките елементи на множествата endpos на клонираните състояния дублират стойности от наследниците си, така че могат да се пренебрегнат. За целта е достатъчно да се пази по един допълнителен бит за всяко състояние (дали е клонирано). В дървото на автомата трябва да се пазят и указатели към преките наследници.

9) *Най-къс липсващ подниз*. По дадена входна азбука и низ S да се намери най-късият низ, съставен от букви на входната азбука, който не е подниз на S. Тази задача се решава с помощта на автомат на наставките на S. Тъй като автоматът е ориентиран ацикличен граф, използва се динамично програмиране. Извършва се топологично сортиране на графа, после върховете се обхождат отзад напред и във всеки връх (състояние) се записва едно число — дължината на най-късия низ, който не може да бъде прочетен при започване на четенето от този връх. За всеки връх, за който липсва изходящ преход с поне една буква от азбуката, това число е 1, защото липсва именно низът, състоящ се само от тази буква, а той има дължина 1. За останалите върхове числото е единица плюс най-малкото измежду числата на състоянията, към които съществуват преходи от текущия връх; единицата се добавя заради буквата на прехода (тя удължава въпросния най-къс низ с един знак). Отговорът на задачата е числото, записано в началния връх на автомата (то се изчислява последно). В случай че се търси не само дължината, а и самият най-къс низ, то във всеки обходен връх може да се пази допълнително по една буква (за върховете от първия вид това е буквата на липсващ преход; за останалите върхове това е буквата на прехода, сочещ към състояние с минимално записано число). Тези букви, прочетени от началното състояние, образуват търсения низ (най-къс низ, липсващ в S); всяка запазена буква указва преход към следващо състояние; четенето завършва при блокиране (тоест при липсващ преход).

Забележка: Ако от едно състояние липсват няколко прехода или съответно има няколко минимума сред състоянията, към които има преход, няма значение буквата на кой точно измежду тези преходи ще бъде запазена.

Пример: За $S = abbcb$ автоматът беше построен в началото. Една възможна топологична сортировка е тази: 0, 1, 2, 4, 3, 5, 6. Изчислени отзад напред, допълнителните числа (дължини) и букви (преходи) на всеки връх са следните: 6 — a, 1; 5 — a, 1; 3 — a, 1; 4 — a, 1; 2 — a, 1; 1 — a, 1; 0 — a, 2. Входната азбука е $\{a, b, c\}$. Един най-къс липсващ подниз е aa (с дължина 2).

10) *Най-дълъг общ подниз на два низа*. По дадени два низа S и T се търси най-дългият им общ подниз. За целта се построява автомат на наставките на S. Всеки общ подниз се съдържа някъде в S и е наставка на някоя представка на T. Инициализират се две променливи: $q := q_0$ и $L := 0$. После знаците на T се четат един по един. За всеки прочетен знак на T се прави преход към състояние q (ако има такъв) и $L := L + 1$. Ако няма преход, $q := link(q)$, после $L := len(q)$. В новото състояние q пак се търси преход (със същата буква на T); ако няма, отново се преминава към родител (по указателя *link*); това продължава, докато се намери преход (и се преминава по него) или се установи липса на преход (значи току-що прочетеният знак от T не се среща в S, затова $q := q_0$ и $L := 0$). Търсената дължина е равна на максималното L. Ако е нужен и самият подниз, то трябва да се запази не само дължината му, а и състоянието q, в което е достигната тази дължина. При предварителната обработка е необходимо също за всяко състояние да се изчисли и запази най-малкият елемент на endpos. По дължината на подниза и позицията на първото му срещане в S не е трудно да се намери и самият подниз.