1.2. Поиск подстроки в строке.

Часто приходится сталкиваться со специфическим поиском, так называемым *поиском подстроки в строке*. Его можно определить следующим образом. Пусть задана строка S из N элементов и строка p из M элементов. Описаны они так:

string $S[N]$, $P[M]$;

Задача поиска подстроки P в строке S заключается в нахождении первого слева вхождения P в S, т.е. найти значение индекса i, начиная с которого

$S[i] = P[0]$, $S[i + 1] = P[1]$,…, $S[i + M – 1] = P[M – 1]$.

1.2.1. Прямой поиск подстроки в строке.

Разберем самый очевидный, самый “прямолинейный” алгоритм поиска.

Алгоритм прямого поиска подстроки в строке

1. Установить i на начало строки S, т.е. $i = 0$.
2. Проверить, не вышло ли $i + M$ за границу N строки S. Если да, то алгоритм завершен (вхождения нет).
3. Начиная с i-го символа s провести посимвольное сравнение строк S и p, т. е. S[i] и P[0], S[i+1] и P[1],…, $S[i + M – 1]$ и $P[M – 1]$.
4. Если хотя бы одна пара символов не совпала, то увеличить i и повторить шаг 2, иначе алгоритм завершен (вхождение найдено).

Приведем пример, иллюстрирующий этот процесс. Символы образца, подвергшиеся сравнению, выделены синим цветом. Здесь

S: на дворе трава, на траве дрова;

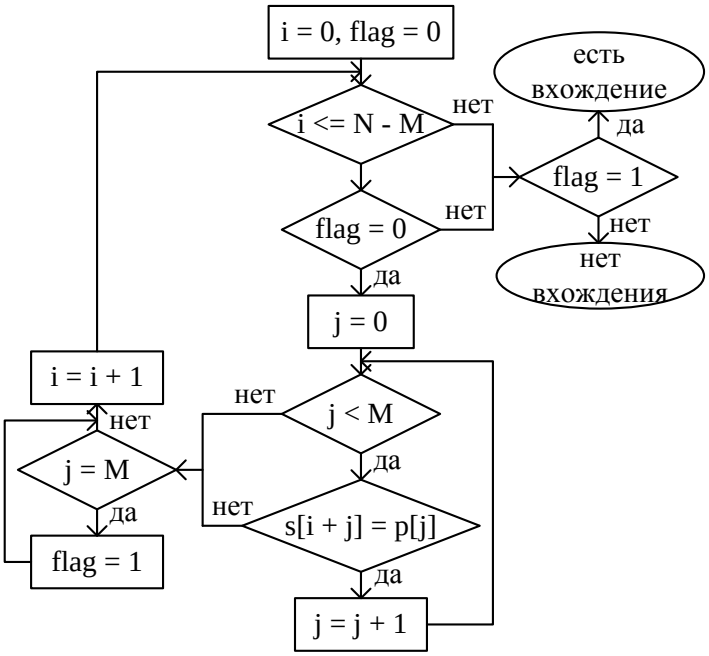
P: траве.

н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
т	р	а	в	е																									
	т	р	а	в	е																								
		т	р	а	в	е																							
			т	р	а	в	е																						
				т	р	а	в	е																					
					т	р	а	в	е																				
						т	р	а	в	е																			
							т	р	а	в	е																		
								т	р	а	в	е																	
									т	р	а	в	е																
									т	р	а	в	е																
									т	р	а	в	е																
									т	р	а	в	е																
									т	р	а	в	е																
										т	р	а	в	е															
											т	р	а	в	е														
												т	р	а	в	е													
													т	р	а	в	е												
														т	р	а	в	е											
															т	р	а	в	е										
																т	р	а	в	е									
																	т	р	а	в	е								

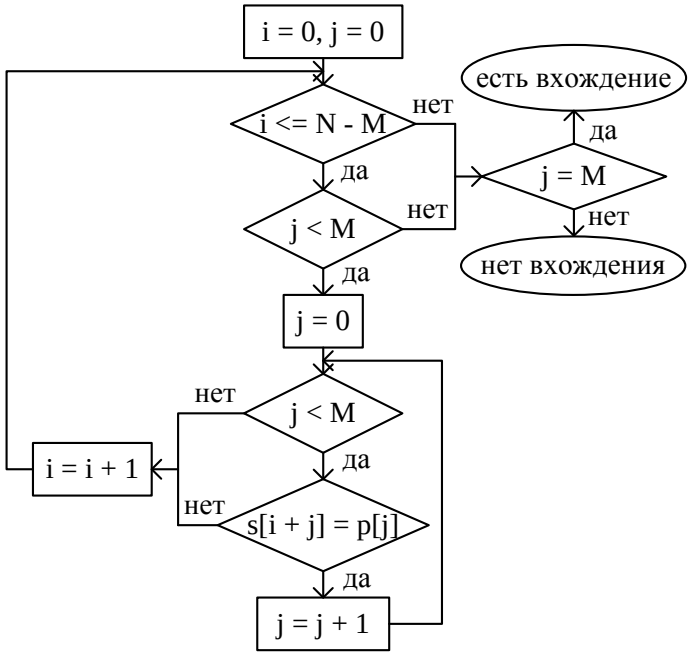
																т	р	а	в	е								
																т	р	а	в	е								
																т	р	а	в	е								
																т	р	а	в	е								
																т	р	а	в	е								

стр

Приведем две **блок-схемы**, описывающие алгоритм. В первой блок-схеме используется дополнительная переменная *flag*, которая явно изменяет свое значение с 0 на 1 при обнаружении вхождения образца *P* в текст *S*.



Во второй блок-схеме используется тот факт, что при $j = M$ мы дошли до конца образца *P*, тем самым обнаружив его вхождение в строку *S*.



стр

Алгоритм работает достаточно эффективно, если при сравнении образца *P* с фрагментом текста *S* довольно быстро выявляется несовпадение (после нескольких сравнений во внутреннем цикле). Это случается довольно часто, но в худшем случае (когда в строке часто встречаются фрагменты, во многих символах совпадающие с образцом) производительность алгоритма значительно падает. Приведем пример. Здесь

S: учиться, учить, учитель

P: учитель

у	ч	и	т	ь	,		у	ч	и	т	ь	с	я	,		у	ч	и	т	е	л	ь
у	ч	и	т	е	л	ь																
у	ч	и	т	е	л	ь																
у	ч	и	т	е	л	ь																
у	ч	и	т	е	л	ь																
у	ч	и	т	е	л	ь																
	у	ч	и	т	е	л	ь															
		у	ч	и	т	е	л	ь														
			у	ч	и	т	е	л	ь													
				у	ч	и	т	е	л	ь												
					у	ч	и	т	е	л	ь											
						у	ч	и	т	е	л	ь										
							у	ч	и	т	е	л	ь									
							у	ч	и	т	е	л	ь									
							у	ч	и	т	е	л	ь									
							у	ч	и	т	е	л	ь									
							у	ч	и	т	е	л	ь									
								у	ч	и	т	е	л	ь								
									у	ч	и	т	е	л	ь							
										у	ч	и	т	е	л	ь						
											у	ч	и	т	е	л	ь					
												у	ч	и	т	е	л	ь				
													у	ч	и	т	е	л	ь			
														у	ч	и	т	е	л	ь		
															у	ч	и	т	е	л	ь	
																у	ч	и	т	е	л	ь
																у	ч	и	т	е	л	ь
																у	ч	и	т	е	л	ь
																у	ч	и	т	е	л	ь
																у	ч	и	т	е	л	ь
																у	ч	и	т	е	л	ь
																у	ч	и	т	е	л	ь

стр

1.2.2. Алгоритм Боуера и Мура

Рассмотрим самый «плохой» для линейного поиска случай:

$S: \underbrace{\mathrm{a}\ldots\mathrm{a}}_{N-1}\mathrm{b},$

$P: \underbrace{\mathrm{a}\ldots\mathrm{a}}_{M-1}\mathrm{b}.$

Подсчитаем для него число сравнений. Так как несовпадение символов происходит на последней букве образца, в каждом внешнем цикле производится M сравнений. Так как образец находится в конце строки, число внешних циклов равно $N - M + 1$, то есть общее число сравнений есть $M(N - M + 1)$. В 1975 году Р. Боуер и Д. Мур предложили алгоритм, который значительно укоряет обработку данного случая. Алгоритм носит название БМ-поиска.

В отличие от линейного поиска, сравнение в БМ-поиске начинается не с начала, а с конца образца P. Кроме того, образец предварительно анализируется, и благодаря этому образец сдвигается не на один символ, а в общем случае на несколько. Рассмотрим механизм сдвига на примере.

S: `на дворе трава, на траве дрова`

P: `дрова`

Начинаем сравнение с последнего символа образца. Символы, подвергшиеся сравнению, выделены: в строке – красным цветом, в образце – синим. Индекс k указывает на первый сравниваемый символ в строке S (первоначально $k = M - 1$), индексы i и j – на сравниваемые символы в образце и строке соответственно. Сначала $i = M - 1$, $j = k$, затем i и j уменьшаются на единицу, пока не произойдет несовпадение символов $S[j]$ и $P[i]$, либо не закончится образец ($i = -1$).

				k																							
				j																							
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
д	р	о	в	а																							
				i																							

Совпадения не произошло при первом же сравнении, необходимо сдвинуть образец, то есть увеличить индекс k. В БМ-поиске образец сдвигается так, чтобы «под» символом $S[k]$ (в таблице) оказался совпадающий с ним ближайший к концу символ образца (иначе при $j = k$ совпадения $S[j]$ и $P[i]$ точно не произойдет). Значит, в данном случае сдвигаем образец на одну позицию вправо, то есть увеличиваем индекс k на единицу.

					k																						
					j																						
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
	д	р	о	в	а																						
					i																						

Совпадения образца со строкой опять не произошло, сдвигаем образец. Теперь $S[k]$ = ‘о’ и расстояние от конца образца до символа ‘о’ в нем равно двум, значит, индекс k увеличиваем на два.

							k																				
							j																				
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
			д	р	о	в	а																				
							i																				

И опять не произошло совпадения образца со строкой. Теперь $S[k]$ = ‘е’, такого символа в образце нет, поэтому мы можем сдвинуть образец сразу на его длину, то есть увеличить индекс k на M.

												k															

												j																
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
								д	р	о	в	а																	
												i																	

стр

Здесь $S[k]$ = ’в’, расстояние от конца образца до символа ‘в’ равно единице, значит, индекс k увеличиваем на единицу.

													k															
											j																	
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
									д	р	о	в	а																
											i																	

Теперь $S[k]$ = ’а’, такой символ в образце есть, но лишь в последней позиции, в остальной части образца его нет, поэтому увеличиваем индекс k на длину образца M.

																	k											
																	j											
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
														д	р	о	в	а											
																	i											

Символа $S[k]$ = ’ ’ в образце нет, увеличиваем индекс k на M.

																					k							
																					j							
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
																			д	р	о	в	а						
																					i							

Символа $S[k]$ = ’е’ в образце нет, увеличиваем индекс k на M.

																											k		
																											j		
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
																								д	р	о	в	а	
																											i		

Символ $S[k]$ = ’в’ в образце есть, расстояние от него до конца образца равно 1, увеличиваем индекс k на 1.

																													k
																								j					
н	а		д	в	о	р	е		т	р	а	в	а	,		н	а		т	р	а	в	е		д	р	о	в	а
																									д	р	о	в	а
																								i					

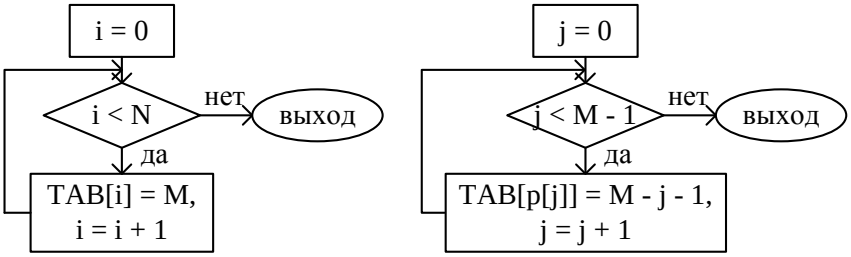
Здесь индекс i = -1, следовательно, образец в строке найден.

В целом индекс k $S[k]$ меняется следующим образом:

- если символа $S[k]$ в образце нет (кроме, может быть, последнего символа!), то k увеличивается на длину образца M;
- если символ $S[k]$ в образце есть, и он не последний, то индекс k увеличивается на расстояние от конца образца до ближайшего символа $S[k]$.

Разумеется, величины сдвигов вычисляются заранее и заносятся в таблицу TAB[256].

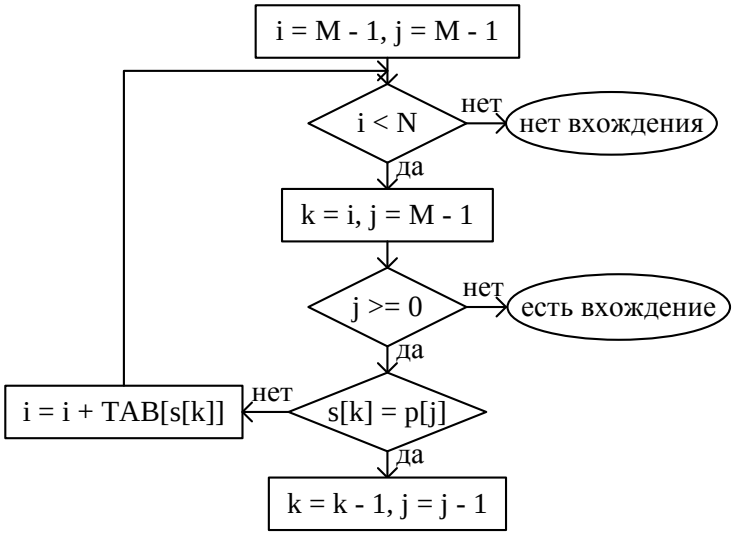
Блок-схема для формирования таблицы TAB очень простая:



Следует особенно отметить, что $j < M - 1$, т.к. TAB[P[$M - 1$]] должен быть равен M, а не 0.

Далее, достаточно очевидно, что образ в строке найден, если индекс j стал меньше 0, и образ в строке не найден, если индекс i дошел до конца строки, т.е. стал больше или равен N (i может стать больше N, так как изменяется “скачками”).

Теперь можно привести **блок-схему** и самого БМ-поиска.



стр

1.2.3. Алгоритм Кнута, Мориса и Пратта

В линейном поиске мы каждый раз сдвигаем образец на одну позицию в строке. В 1970 году Д. Кнут, Д. Морис и В. Пратт изобрели алгоритм, называемый КМП-поиском, который основывается на следующем соображении: если произошло совпадение нескольких символов образца и строки, то мы получили некоторую информацию о строке, которую можно использовать для сдвига образца. Изучим механизм сдвига на примерах.

Пусть в образце все буквы не совпадают с первой, например:

S: `от топота копыт пыль по полю летит`

P: `поле`

Рассмотрим несколько случаев. Символы, подвергшиеся сравнению, выделены. Индексы *i* и *j* указывают на сравниваемые символы в образце и строке соответственно. Сначала *i* = *j* = 0, затем *i* и *j* увеличиваются на единицу, пока не произойдет несовпадение символов *S*[*j*] и *P*[*i*], либо не закончится образец (*i* = *M*).

j																															
о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		П	о		п	о	л	ю		л	е	т	и
п	о	л	е																												
i																															

Пусть уже первый символ образца не совпадает с символом в строке, тогда, не имея информации о составе строки после индекса *j*, мы увеличиваем *j* на единицу и оставляем *i* равным нулю, то есть сдвигаем образец на одну позицию в строке.

	j																														
о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		П	о		п	о	л	ю		л	е	т	и
	п	о	л	е																											
	i																														

Продолжаем действовать так до тех пор, пока не обнаружим совпадения символов образца и строки. Обратим внимание, что индекс *j* лишь увеличивается.

о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		П	о		п	о	л	ю		л	е	т	и
	п	о	л	е																											
		п	о	л	е																										
			п	о	л	е																									
				п	о	л	е																									

Через несколько шагов получаем:

							j																								
о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		П	о		п	о	л	ю		л	е	т	и
					п	о	л	е																								
							i																								

Теперь совпали два символ образца и строки. Так как все символы образца не совпадают с первым символом (в частности *P*[0] ≠ *P*[2]) и *S*[*j*] ≠ *P*[2], то возможно, что *S*[*j*] = *P*[0]. С другой стороны, *P*[0] ≠ *P*[1] и *S*[*j* - 1] = *P*[1], значит, *S*[*j* - 1] ≠ *P*[0], и не имеет смыслы сравнивать эти символы. Поэтому в данном случае индекс *j* не меняется, а индекс *i* полагается равным нулю. Следующие несколько шагов сведем в одну таблицу.

о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		П	о		п	о	л	ю		л	е	т	и
							п	о	л	е																						
								п	о	л	е																					
									п	о	л	е																				
										п	о	л	е																			
											п	о	л	е																		

стр

Остановимся на случае, когда совпал один символ.

												j																		
о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		п	о		п	о	л	ю		л	е	т	и
												п	о	л	е																
												i																			

Так как все символы в образце не совпадают с первым, и $S[j] = P[1]$, значит возможно, что $S[j] = P[0]$. Поэтому, как и в предыдущем случае, индекс j не меняется, а индекс i полагается равным нулю. Следующие несколько шагов опять сведем в одну таблицу.

о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		п	О		п	о	л	ю		л	е	т	и
													п	о	л	е																
														п	о	л	е															
															п	о	л	е														
																п	о	л	е													
																	п	о	л	е												
																		п	о	л	е											
																			п	о	л	е										
																				п	о	л	е									
																					п	О	л	е								
																						п	о	л	е							

Теперь совпало три буквы образца.

																										j					
о	т		т	о	п	о	т	а		к	о	п	ы	т		п	ы	л	ь		п	О		п	о	л	ю		л	е	т	и
																								п	о	л	е					
																											i					

Если провести рассуждения, аналогичные предыдущим, то получим, что индекс j не меняется, а индекс i полагается равным нулю.

Можно заметить, что индекс j не меняется, если совпала хотя бы часть образца, и увеличивается на единицу, если ни один символ не совпал. Значит, нет смысла запоминать позицию в строке, где начинается образец, как это делается в линейном поиске (индекс k). Сдвиг образца в КМП-поиске происходит за счет изменения индекса i (это можно видеть в сводной таблице внизу, где выделены рассмотренные символы и показан сдвиг образца). Если хотя бы часть образца совпала, индекс i обнуляется. Затем $P[i]$ сравнивается с $S[j]$. Если же ни один символ не совпал, i остается равным нулю, а j увеличивается на единицу. То есть можно считать, что $i = -1$, а затем сравнить $P[i + 1]$ с $S[j + 1]$.

j							*j*						*j*															*j*					
о	т		т	о	п	О	т	а		к	о	п	ы	т		п	ы	л	ь		п	о		п	о	л	ю		л	е	т	и
п	о	л	е		п	О	л	е				п	о	л	е									п	о	л	е					
	п	о	л	е			п	о	л	е			п	о	л	е												п	о	л	е		
i							*i*						*i*															*i*					

Всю эту информацию можно занести в массив D, где $D[k]$ – новое значение индекса i при условии, что совпало kсимволов.

k	0	1	2	3
D[*k*]	–1	0	0	0
P[*k*]	п	о	л	е

Очевидно, что массив имеет подобный же вид во всех случаях, когда ни один из символов образца не совпадает с первым.

k	0	1	…	*m*
D[*k*]	–1	0	0	0

стр

Общие правила построения массива *D* таковы:

- *D*[0] = -1
- Если перед *P*[*j*] нет фрагмента, который совпадает с началом образца, и *P*[*j*] ≠ *P*[0], то *D*[*j*] = 0

Рассмотрим теперь ситуацию, когда насколько первых символов образца повторяются в образце еще раз.

S: каравай каркнуть каркать

P: каркас

Здесь зеленым цветом выделены повторяющиеся фрагменты образца.

Первые три символа в образце различны. Поэтому если несовпадение образца и строки произошло при *i* = 0, 1, 2, надо действовать, как описано выше. Рассмотрим случай, когда совпало три символа, то есть несовпадение обнаружено при *i* = 3.

			j																			
к	а	р	а	в	а	й		к	а	р	к	н	у	т	ь		к	а	р	к	а	т	ь
к	а	р	к	а	с																	
			i																			

Часть образца *P*[0]*P*[1] (ка) совпадает с частью *P*[3]*P*[4]. Несовпадение обнаружено при *i* = 3, то есть *S*[*j*] ≠ *P*[3]. С другой стороны, *P*[3] = *P*[0], значит, *S*[*j*] ≠ *P*[0], и образец сдвигается так:

			j																			
к	а	р	а	в	а	й		к	а	р	к	н	у	т	ь		к	а	р	к	а	т	ь
к	а	р	к	а	с																	
				к	а	р	к	а	с													
			i																			

Значит, если совпало три символа, то индекс *i* = -1. Обратим внимание, что *P*[3] = *P*[0]. Общее правило таково:

- Если перед *P*[*j*] нет фрагмента, который совпадает с началом образца, и *P*[*j*] = *P*[0], то *D*[*j*] = -1

Рассмотрим случай, когда совпало четыре символа.

												j											
к	а	р	а	в	а	й		к	а	р	к	н	у	т	ь		к	а	р	к	а	т	ь
								к	а	р	к	а	с										
												i											

Часть образца *P*[0]*P*[1] (ка) совпадает с частью *P*[3]*P*[4]. Обратим внимание, что в данном случае ситуация такова: символу *P*[4] предшествует символ *P*[3] = *P*[0], то есть

фрагмент длины 1, совпадающий с началом образца. При этом $P[4] = P[1]$, то есть для $i = 4$ верны те же рассуждения, что и для $i = 1$.

Несовпадение символов произошло при $i = 4$, то есть $S[j] \neq P[4]$, а так как $P[4] = P[1]$, значит, $S[j] \neq P[1]$. При этом $S[j - 1] = P[3] = P[0]$. Значит, так как $P[0] \neq P[1]$, возможно, что $S[j] = P[0]$, и образец надо сдвинуть следующим образом:

												j										
к	а	р	а	в	а	й		к	а	р	к	н	у	т	ь		к	а	р	к	а	т	ь
								к	а	р	к	а	с									
												к	а	р	к	а	с					
												i										

Значит, если совпало четыре символа, то индекс j не меняется, а индекс i равен нулю.

стр

На основании этих наблюдений можно сформулировать **правило:**

- Если перед $P[j]$ есть фрагмент длины k, который совпадает с началом образца, и $P[j] = P[k]$, то $D[j] = D[k]$

Рассмотрим ситуацию, когда совпало пять символов.

																				j	
к	а	р	а	в	а	й		к	а	р	к	н	у	т	ь		к	а	р	к	а	т	ь
																	к	а	р	к	а	с	
																						i	

Несовпадение символов произошло при $i = 5$, то есть $S[j] \neq P[5]$, и при этом $S[j - 1] = P[4] = P[1]$ и $S[j - 2] = P[3] = P[0]$. Так как $P[2] \neq P[5]$ и $S[j] \neq P[5]$, возможно, что $S[j] = P[2]$, и образец надо сдвинуть следующим образом:

																				j					
к	а	р	а	в	а	й		к	а	р	к	н	у	т	ь		к	а	р	к	а	т	ь				
																	к	а	р	к	а	с					
																				к	а	р	к	а	с	
																						i				

Образец сдвигается так, чтобы совместить совпадающие части образца и строки (выделенные зеленым цветом). Здесь индекс j не меняется, а индекс i равен 2 (числу символов в совпадающих частях образца). Еще раз подчеркнем, что нет необходимости сдвигать назад индекс j, так как в ходе предыдущих сравнений получено, что $S[j - 2]S[j - 1] = P[3]P[4] = P[0]P[1] =$ ка.

Общее **правило** здесь такое

- Если перед $P[j]$ есть фрагмент длины k, который совпадает с началом образца, и $P[j] \neq P[k]$, то $D[j] = k$

Всю эту информацию можно занести в массив D, где $D[k]$ – новое значение индекса i при условии, что совпало kсимволов.

k	0	1	2	3	4	5
$D[k]$	–1	0	0	–1	0	2

$P[k]$	к	а	р	к	а	с

В общем случае, когда первые q символов образца повторяются затем, начиная с l-го символа ($P[0]\ldots P[q-1] = P[l]\ldots P[l+q-1]$), массив D имеет вид:

k	0	1	…	$l-1$	l	$l+1$	…	$l+q-1$	$l+q$	$l+q+1$	…	$M-1$
$D[k]$	–1	0	0	0	–1	0	0	0	q	0	0	0

При этом не обязательно $l > q - 1$! Например, если образец имеет вид $P: \underbrace{\text{a}\ldots\text{a}}_{m-1}\text{b}$, то массив (фрагмент $P[1]\ldots P[M-2]$ совпадает с началом образца $P[0]\ldots P[M-3]$), массив D выглядит так:

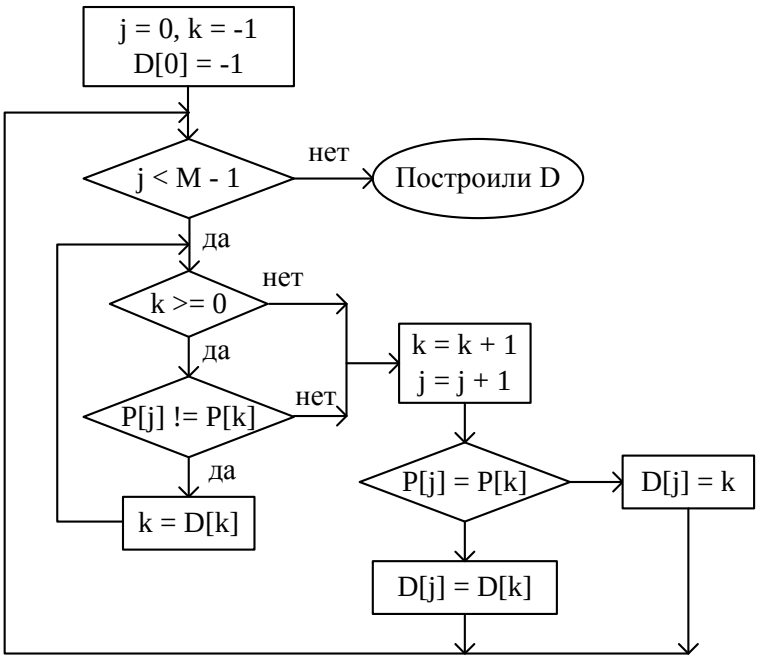
k	0	…	$M-2$	$M-1$
$D[k]$	–1	…	–1	$M-2$
$P[k]$	a	…	a	b

стр

Перечислим здесь все выведенные нами правила формирования массива D:

- $D[0] = -1$
- Если перед $P[j]$ нет фрагмента, который совпадает с началом образца, и $P[j] \neq P[0]$, то $D[j] = 0$
- Если перед $P[j]$ нет фрагмента, который совпадает с началом образца, и $P[j] = P[0]$, то $D[j] = -1$
- Если перед $P[j]$ есть фрагмент длины k, который совпадает с началом образца, и $P[j] \neq P[k]$, то $D[j] = k$
- Если перед $P[j]$ h есть фрагмент длины k, который совпадает с началом образца, и $P[j] = P[k]$, то $D[j] = D[k]$

Таким образом, чтобы построить массив D, необходимо исследовать образец и найти в нем фрагменты, совпадающие с его началом, то есть, по сути, решить задачу поиска подстроки в строке. Это можно осуществить с помощью того же алгоритма КМП-поиска. Приведем **блок-схему** построения массива D.



Пример (по гиперссылке)

стр

k	0	1	2	3	4	5	6	7	8
$D[k]$	–1	0	0	0	–1	0	0	3	–1
$P[k]$	a	b	c	d	a	b	c	e	f

Здесь зеленым цветом выделены фрагменты образца, подвергшиеся сравнению в предыдущем цикле, синим – в текущем цикле.

S	a	b	c	a	b	k	a	b	c	d	a	a	b	c	d	a	b	c	d	e	a	b	c	d	a	b	c	e	f	h
P	a	b	c	d	a	b	c	e	f																					
i				3																										
P				a	b	c	d	a	b	c	e	f																		
i						2																								
P						a	b	c	d	a	b	c	e	f																
i						0																								
P							a	b	c	d	a	b	c	e	f															
i												5																		
P												a	b	c	d	a	b	c	e	f										
i																			7											
P																a	b	c	d	a	b	c	e	f						
i																				4										
P																					a	b	c	d	a	b	c	e	f	
i																														9

другой пример

стр

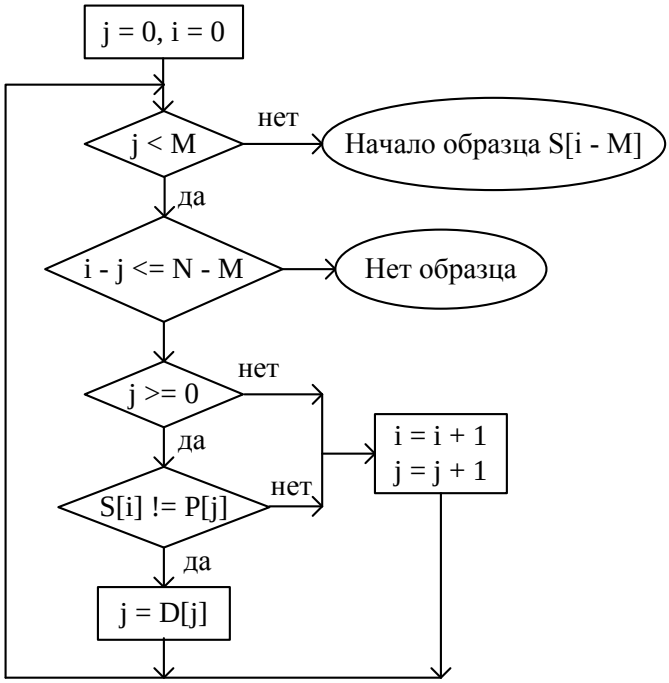
S: varevo varvara varvaram varila

P: varvary

S	в	а	р	е	в	о		в	а	р	в	а	р	а		в	а	р	в	а	р	а	м		в	а	р	и	л	а
P	в	а	р	в	а	р	ы																							
i				3																										
P				в	а	р	в	а	р	ы																				
i				0																										
P					в	а	р	в	а	р	ы																			
i						1																								
P						в	а	р	в	а	р	ы																		
i						0																								
P							в	а	р	в	а	р	ы																	
i							0																							
P								в	а	р	в	а	р	ы																
i														6																
P											в	а	р	в	а	р	ы													
i														3																
P															в	а	р	в	а	р	ы									
i															0															
P																в	а	р	в	а	р	ы								
i																						6								
P																			в	а	р	в	а	р	ы					
i																						3								
P																							в	а	р	в	а	р	ы	
i																							0							
P																								в	а	р	в	а	р	ы
i																								0						

стр

Блок-схема собственно КМП-поиска имеет вид:



Обсудим преимущества данного алгоритма. Как вы могли заметить на примерах, индекс j, который передвигается по строке, может только увеличиваться. В связи с этим можно выделить два важных момента.

Во-первых, рассмотрим самый плохой случай для линейного поиска.

$S: \underbrace{\text{a}\ldots\text{a}}_{N-1}\text{b},$

$P: \underbrace{\text{a}\ldots\text{a}}_{M-1}\text{b}.$

Как было показано (ссылка на БМ-поиск), число сравнений в линейном поиске равно $M(N - M + 1)$. Рассмотрим, как в этом случае работает КМП-поиск.

k	0	…	M - 2	M - 1
$D[k]$	–1	…	–1	M - 2
$P[k]$	a	…	a	b

S	a	…	a	a	a	…	a	…	a	b
P	a	…	a	b						
i				M - 1						
P		…	a	a	b					
i					M - 1					
…						…				
P							a	…	a	b
i										M - 1

При значениях индекса $i = 0, 1, \ldots, M - 2$ символы строки и образца совпадают: $S[j] = P[i]$. При $i = M - 1$ обнаруживается несовпадение, и образец сдвигается так, что $i = M - 2$, то есть на одну позицию, при этом символ $S[j]$ сравнивается с символом $P[M - 2]$. Таким образом, символы строки, начиная с $j = M - 1$ (кроме последнего), сравниваются сначала с $P[M - 1]$, а затем с $P[M - 2]$, то есть подвергаются двум сравнениям. Остальные символы строки сравниваются с символами образца лишь один раз. Таким образом, общее число сравнений есть $2N - M + 1$, что в общем случае существенно меньше, чем в линейном поиске.

Во-вторых, индекс j, движущийся по строке, может только увеличиваться. Это позволяет искать образец в строке, заданной не массивом, а файлом, не используя дополнительных массивов для хранения информации.