* 1. Najděte v textu *T* všechny výskyty řetězců, které mají od vzorku *P* Hammingovu vzdálenost rovnou nejvýše *k*. Použijte metodu dynamického programování.

a) *T* = ccacbaabccaccbcabccc, *P* = abcba, *k* = 2

Inicializace tabulky:

		c	c	a	c	b	a	a	b	c	c	a	c	c	b	c	a	b	c	c	c
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	-																				
b	-	-																			
c	-	-	-																		
b	-	-	-	-																	
a	-	-	-	-	-																

Znak ‘-‘ reprezentuje nedefinovanou hodnotu. Tabulku vyplňujeme po řádcích. Hodnota v červeném kolečku je spočítána jako hodnota v levé horní sousední buňce (zelené kolečko) zvětšená o 1, pokud symboly v řádku a sloupci (modrá kolečka) jsou různé. V případě, že symboly se shodují, přepíšeme na pozici červeného kolečka hodnotu ze zeleného kolečka beze změny.

	-	c
-	0	0
a	-	1

	-	a
-	0	0
a	-	0

Vyplněná tabulka:

		c	c	a	c	b	a	a	b	c	c	a	c	c	b	c	a	b	c	c	c
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	-	1	1	0	1	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	1
b	-	-	2	2	1	1	2	1	0	2	2	2	1	2	1	2	2	0	2	2	2
c	-	-	-	3	2	2	2	3	2	0	2	3	2	1	3	1	3	3	0	2	2
b	-	-	-	-	4	2	3	3	3	3	1	3	4	3	1	4	2	3	4	1	3
a	-	-	-	-	-	5	2	3	4	4	4	1	4	5	4	2	4	3	4	5	2

V posledním řádku nalezneme hodnoty menší nebo rovné k = 2 (viz fialová kolečka). Udávají pozice v textu, na kterých končí podřetězec textu délky 5, jež má Hammingovu vzdálenost od vzoru abcba nejvýše k = 2. Jako příklad je v textu oranžově vyznačený podřetěrec s Hammingovou vzdáleností 1.

* 2. Najděte v textu *T* všechny výskyty řetězců, které mají od vzorku *P* Levenshteinovu vzdálenost rovnou nejvýše *k*. Použijte metodu dynamického programování

a) *T* = aacacacbaabbbcbbcacc, *P* = cbbba, *k* = 3.

Postup je podobný, trochu odlišná je ale inicializace, výpočet hodnoty políčka i interpretace výsledku.

Inicializace:

		c	c	a	c	b	a	a	b	c	c	a	c	c	b	c	a	b	c	c	c
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1																				
b	2																				
c	3																				
b	4																				
a	5																				

Tabulku dále vyplňujeme podle úplně stejných pravidel jako když počítáme Levensteinovu vzdálenost dvou slov (viz předchozí cvičení).

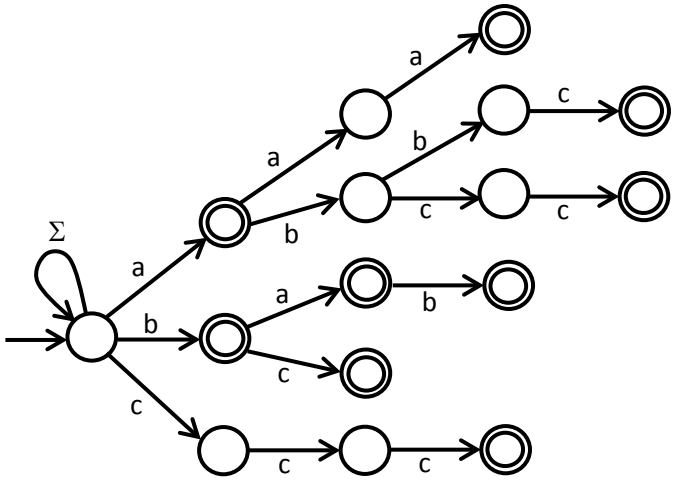
Vyplněná tabulka:

		c	c	a	c	b	a	a	b	c	c	a	c	c	b	c	a	b	c	c	c
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1	1	1	0	1	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	1
b	2	2	2	1	1	1	1	1	0	1	2	1	1	2	1	2	1	0	1	2	2
c	3	2	2	2	1	2	2	2	1	0	1	2	1	1	2	1	2	1	0	1	2
b	4	3	3	3	2	1	2	3	2	1	1	2	2	2	1	2	2	2	1	1	2
a	5	4	4	3	3	2	1	2	3	2	2	1	2	3	2	2	2	3	2	2	2

V posledním řádku jsou pouze první tři hodnoty větší než k = 3 (vyznačené červeným křížkem). To znamená, že na všech ostatních pozicích v textu končí nějaký podřetězec, jehož Levenshteinova vzdálenost od vzoru ‘abcba’ je nejvýše 3. Jako příklad je v tabulce vyznačený podřetězec acba, který má od abcba vzdálenost 1 (což je vidět, stačí insertnout ‘b’ mezi první dva znaky). Všimněme si, že oproti výpočtu Hammingovy vzdálenosti nemáme informaci o délce nalezeného podřetězce na dané pozici, podřetězec je potřeba dohledat.

* 3. Sestrojte nedeterministický automat, který v textu nad abecedou A vyhledá právě každé slovo množiny M.
a) A = {a, b, c}, M = {a, b, ba, bc, aaa, bab, ccc, abbc, abcc } .

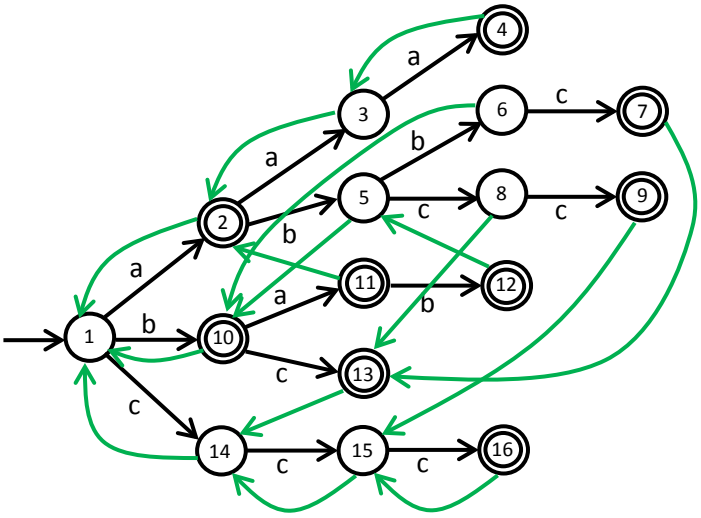
Cílem je vytvořit nedeterministický slovníkový automat pro množinu slov M. Automat má stromovou strukturu kde iniciální stav je kořenem. Každé slovo umístíme na cestu začínající v kořenu. Poslední stav cesty odpovídající slovu je označen jako přijímací. Společné prefixy slov sdílí stejné úseky. Iniciální stav obsahuje nedeterministickou smyčku.



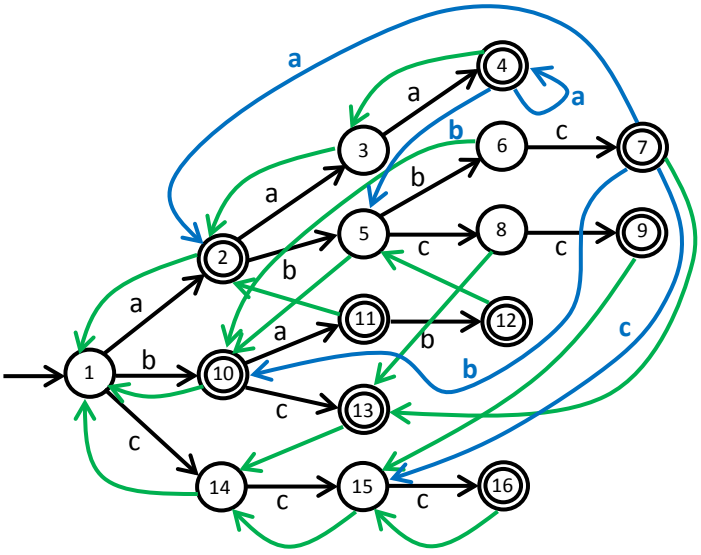
* 4. Sestrojte deterministický automat, který v textu nad abecedou A vyhledá právě každé slovo množiny M z předchozí úlohy.

Deterministický slovníkový automat získáme z předchozího nedeterministického. Stavy zůstávají stejné. Odstraníme nedeterministickou smyčku a potřebujeme v každém stavu dodefinovat přechody pro všechny vstupní symboly (v našem případě a,b,c). Postupujeme podle následujícího algoritmu. Jako mezikrok nejprve pro každý stav (dosažitelný z počátečního stavu čtením neprázdného slova *u*) nalezneme stav dosažitelný po přečtení nejdelšího možného vlastního sufixu slova *u*.

Výsledkem jsou zelené hrany na obrázku níže. Například, stav 7 je dosažitelný z počátečního stavu po přečtení abbc. Nejdelším vlastním sufixem tohoto slova je bbc, pro toto slovo však neexistuje cesta z počátečního stavu. Druhým nejdelším sufixem je slovo bc. To je reprezentováno stavem 13. Proto ze stavu 7 vede zelená šipka do stavu 13. Jiný příklad, stav 8 reprezentuje slovo abc. Nejdelším vlastním sufixem je slovo bc, proto zelená šipka směřuje ze stavu 8 do stavu 13.

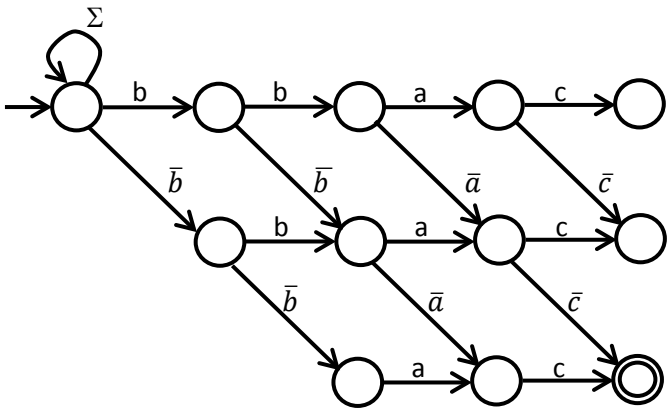


Nyní můžeme dodefinovat všechny přechody. Pokud máme stav *s* a symbol *d*, pro který není přechod z *s* definovaný, přejdeme po zelené šipce do stavu reprezentujícího vlastní sufix (stav *p*). Pokud je v tomto stavu definován přechod pro symbol *d* (do stavu *r*), doplníme přechod pro *d* z *s* do *r*. Pokud definován není, zkusíme další stav v pořadí, na který ukazuje zelená šipka ze stavu *p*. V případě, že se dostaneme až do iniciálního stavu a přechod stále není definovaný, bude přechod pro *d* z *s* směřovat do iniciálního stavu.
Příklad: Uvažujme stav 7 a symbol *c*, pro který není přechod v nedeterministickém automatu definovaný. Po zelené šipce jdeme do stavu 13. Ani zde přechod pro *c* není definovaný, jdeme tedy po zelené šipce do stavu 14. Zde máme přechod pro *c* do stavu 15. Z toho důvodu doplníme pro deterministický automat přechod pro *c* ze stavu 7 do stavu 15.
Algoritmus je výhodné provádět tak, že bereme stavy podle jejich hloubky ve stromu. To má za následek, že se po zelené šipce vracíme pouze jednou, do stavu s menší hloubkou, který již bude mít všechny přechody definované.
Jiný příklad: Uvažujme stav 4 a symbol *a*. Přechod není definovaný, jdeme po zelené šipce do stavu 3. Vidíme, že zde přechod pro *a* směřuje do stavu 4. Proto ze stavu 4 po přečtení symbolu *a* přejdeme v deterministickém automatu opět do stavu 4.
Několik dalších přechodů deterministického automatu je ilustrováno níže (modré šipky).

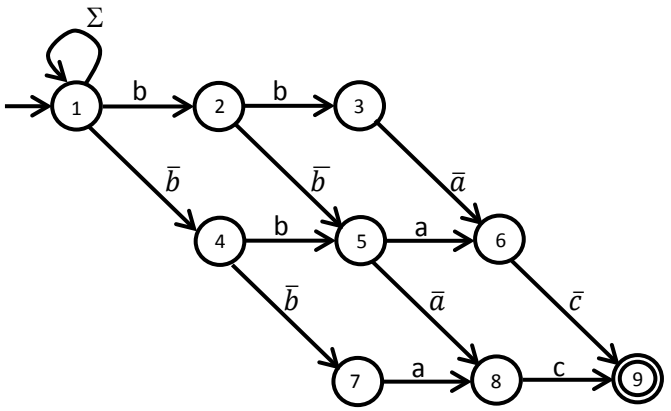


* 5. Sestavte tabulky pro simulaci činnosti vyhledávacího automatu metodou bitového paralelizmu pro daný text *t*, vzorek *p* a Hammingovu vzdálenost *k*
a) *t* = abcbcaaccbbaa, *p* = bbac, *k* = 2

Nejprve sestrojíme nedeterministický automat, který počítá Hammingovu vzdálenost 2 od daného vzoru *p*.



Automat dále zjednodušíme odstraněním stavů, ze kterých se nedostaneme do přijímacího stavu.



Pro simulaci tohoto automatu „bitovým paralelismem“ budeme potřebovat bitové vektory délky 9 (každý bit reprezentuje jeden ze stavů, *i*-tý bit zleva bude odpovídat stavu *i*). Počáteční konfigurace je popsána vektorem <1,0,0,0,0,0,0,0,0>. Tabulka přechodů udává pro každý stav *s* a symbol *d* množinu stavů, do kterých existuje přechod z *s* po přečtení *d*. Například, pro stav 1 máme v tabulce pro symboly *a*,*b*,*c* postupně vektory <1,0,0,1,0,0,0,0,0>, <1,1,0,0,0,0,0,0,0>, <1,0,0,1,0,0,0,0,0>. Kromě toho, vektor F=<0,0,0,0,0,0,0,0,1> reprezentuje množinu přijímacích stavů. Simulace probíhá tak, že vezmeme aktuální konfiguraci (množinu aktivních stavů) a vypočítáme konfiguraci následnou pro čtený symbol *d*. Procházíme bity současné konfigurace. Kdykoliv je *i*-tý bit roven 1, vezmeme z tabulky vektor přechodů pro stav *i* a symbol *d*. Na všechny takovéto vektory aplikujeme bitovou operaci OR. Tím obdržíme následnou konfiguraci.

Příklad: V počáteční konfiguraci čteme symbol *a*. Aktivní je pouze stav 1. Z tabulky vezmeme vektor pro přechody po čtení *a*, tedy <1,0,0,1,0,0,0,0,0>, což bude druhá konfigurace. Dále čteme symbol *c*. Máme nyní dva aktivní stavy, 1 a 4. Těm odpovídají vektory přechodů <1,0,0,1,0,0,0,0,0> a <0,0,0,0,0,0,1,0,0>. Jejich OR je vektor <1,0,0,1,0,0,1,0,0> který je následnou konfigurací, atd.

Navíc, pro každou novou konfiguraci testujeme, zda neobsahuje přijímací stav. To provedeme pomocí bitové operace AND aplikované na vektor konfigurace a vektor přijímacích stavů F (pokud je výsledek nenulový, konfigurace obsahuje přijímací stav).