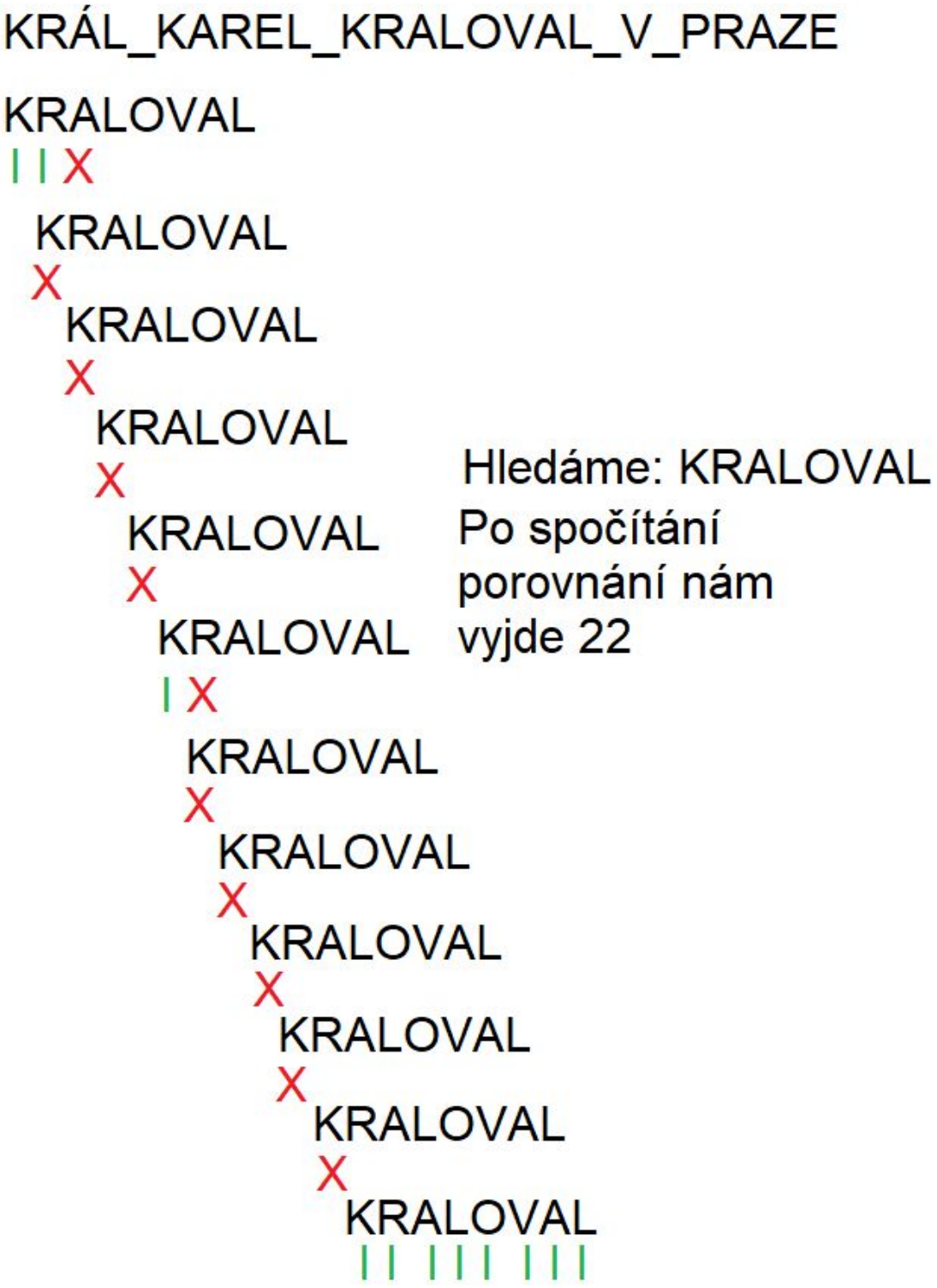
Určení, kolik operací porovnání se musí provést

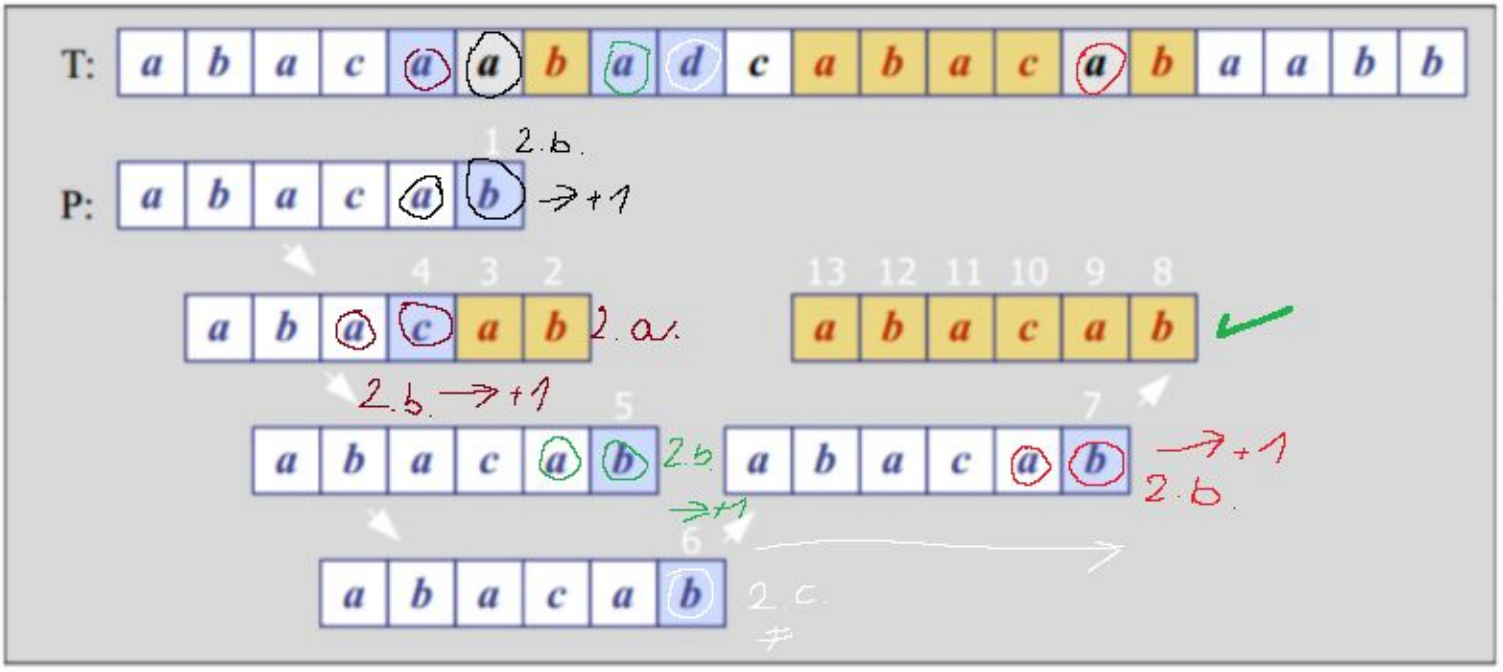
Jednoduchý algoritmus přímého vyhledávání brute-force

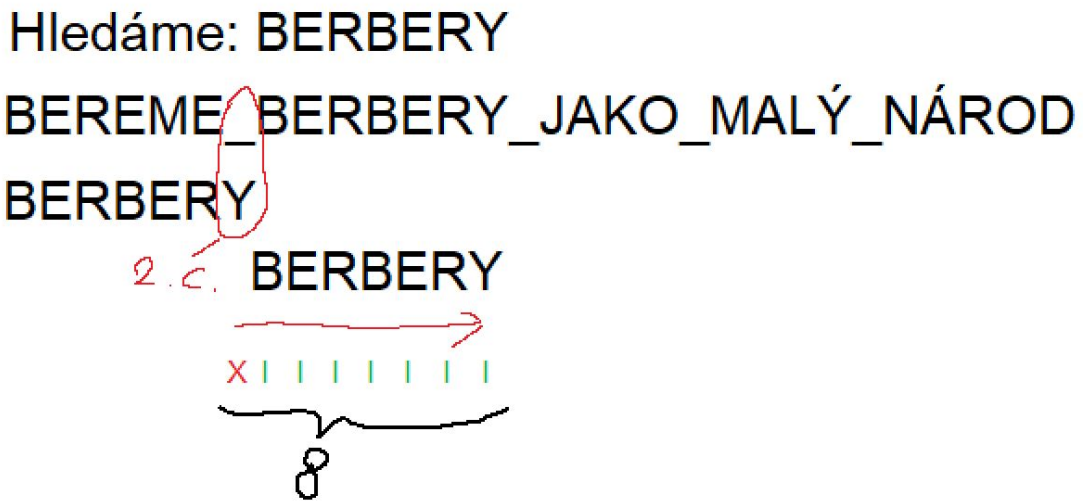
1. Porovnám písmeno s písmenem, pokud je stejné posunu se na další písmeno, opakuji dokud jsou písmena stejná
2. Jakmile narazím na písmeno rozdílné, tak se posunu o jedno doprava od počátečního testovaného písmena (musím počítat i porovnání, kde písmeno bylo rozdílné)
3. Pokud narazím rovnou na rozdílné písmeno, tak si jen počítám porovnání a jdu na další písmeno
4. ???
5. profit



Boyer - Mooreův algoritmus (doplnění funkce Last(x))

1. Začínáme od konce řetězce, který hledáme
2. Máme několik možností:
    a. Když se symboly rovnají, tak porovnáme ty před nimi
    b. Když se symboly nerovnají, ale symbol v řetězci ve kterém hledáme *(T)* je někde v hledaném řetězci *(P)*, tak hledaný řetězec posuneme tak, aby se rovnaly
    c. Když se symboly nerovnají a symbol v řetězci ve kterém hledáme se nenachází nikde v hledaném řetězci tak skočíme o celou délku hledaného řetězce





Hodnotou funkce **last(x)** je vlastně index posledního výskytu znaku x ve hledaném řetězci (-1 = v řetězci není)

BERBERY
0 1 2 3 4 5 6

X	_	A	B	D	E	J	K	L	M	O	R	Y	Á	Ý
Last(x)	-1	-1	3	-1	4	-1	-1	-1	-1	-1	5	6	-1	-1

Knuth - Morris - Prattův algoritmus (doplnění chybové funkce F(k) KMP)

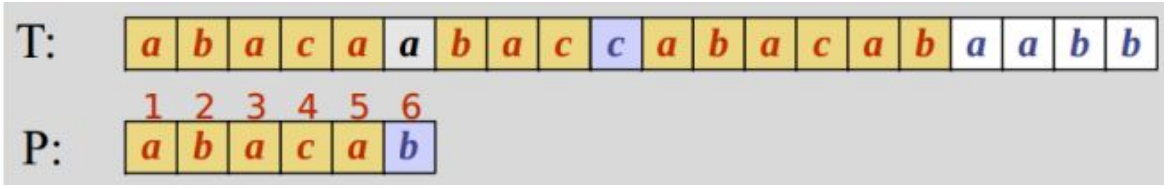
1. **Chybová funkce**
   a. Tabulka o velikosti řetězce (od 0)
   b. U 0 bude vždy hodnota 0
   c. Poté vypíšeme všechny možné prefixy a sufixy
   d. Když se nějaké rovnají, tak napíšeme do tabulky délku toho nejdelšího, když se nerovnají, tak 0
   e. Takto projdeme tabulku až do konce

	B	E	R	B	E	R	Y
k	0	1	2	3	4	5	6
F(k)	0	0	0	1	2	3	0

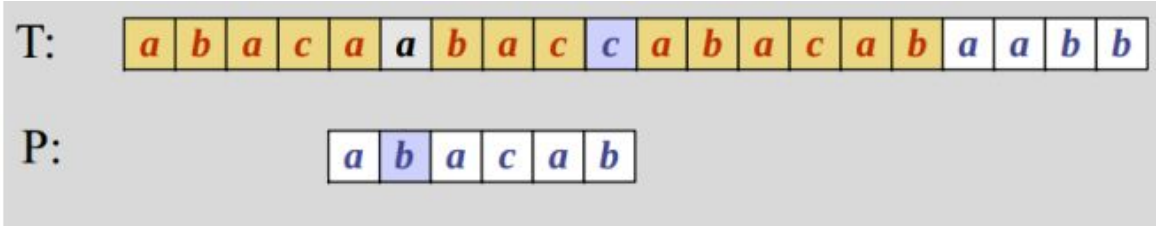
	1	2	3	4	5	6
Prefixy:	B	B BE	B BE BER	B BE BER BERB	B BE BER BERB BERBE	B BE BER BERB BERBE BERBER
Sufixy:	E	R ER	B RB ERB	E BE RBE ERBE	R ER BER RBER ERBER	Y RY ERY BERY RBERY ERBERY
	0	0	B	BE	BER	0

k	0	1	2	3	4	5
***P*[*k*]**	***a***	***b***	***a***	***c***	***a***	***b***
***F*(*k*)**	0	0	1	0	1	2

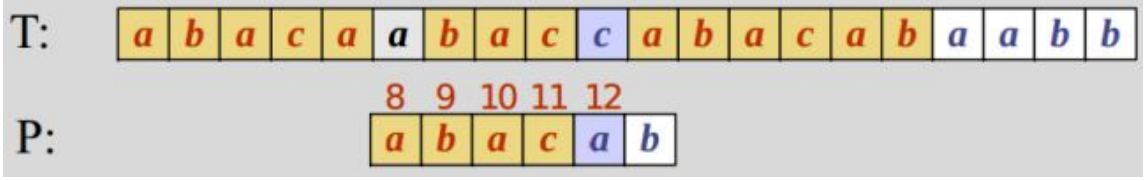
2. **Samotné vyhledávání**
   a. Začnu od začátku, pokud se první symbol rovná, tak pokračuju na další, pokud ne, posunu se o jedna
   b. Když po == symbolech narazím na symbol, který se nerovná, kouknu se v hledaném řetězci na symbol o jedna zpátky a kouknu se na jeho hodnotu v tabulce



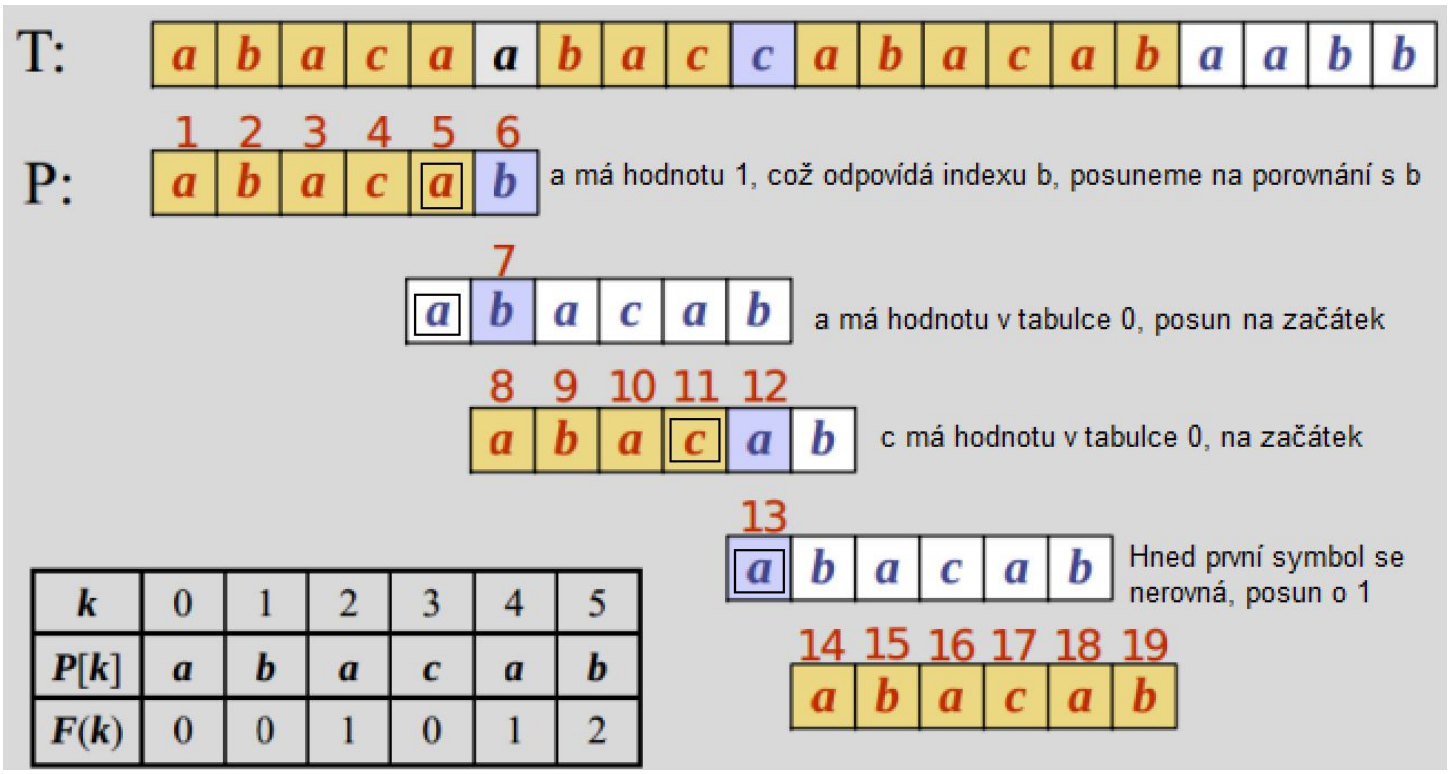
   V tomto případě by to bylo a, které podle tabulky má hodnotu 1
   c. Pokud má jinou hodnotu než 0, posunu řetězec tak, aby se porovnával symbol, který na tom indexu tabulky je



   d. Pokud má hodnotu 0, tak porovnávám zase od začátku, b na obrázku dva má hodnotu v tabulce 0, takže porovnáváme zase od začáku



Celé řešení



k	0	1	2	3	4	5
$P[k]$	a	b	a	c	a	b
$F(k)$	0	0	1	0	1	2

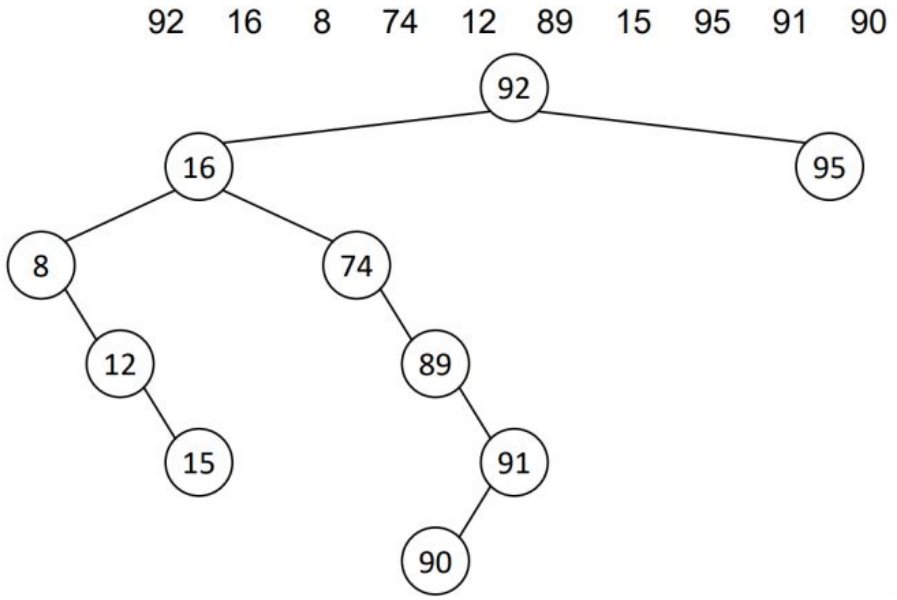
(Tady se nevyhledává odzadu, poznámka pro mě)

Reprezentace položek BVS (BST) stromem

každý uzel pouze levého a pravého potomka
levý potomek obsahuje pouze klíče menší než klíče uzle nad ním
pravý potomek obsahuje pouze klíče větší než klíče uzle nad ním

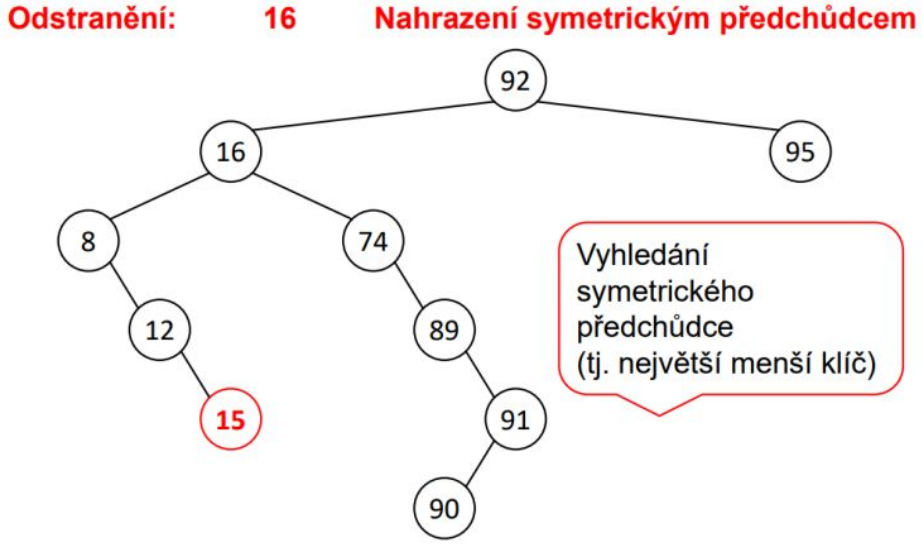
Vložení prvku X

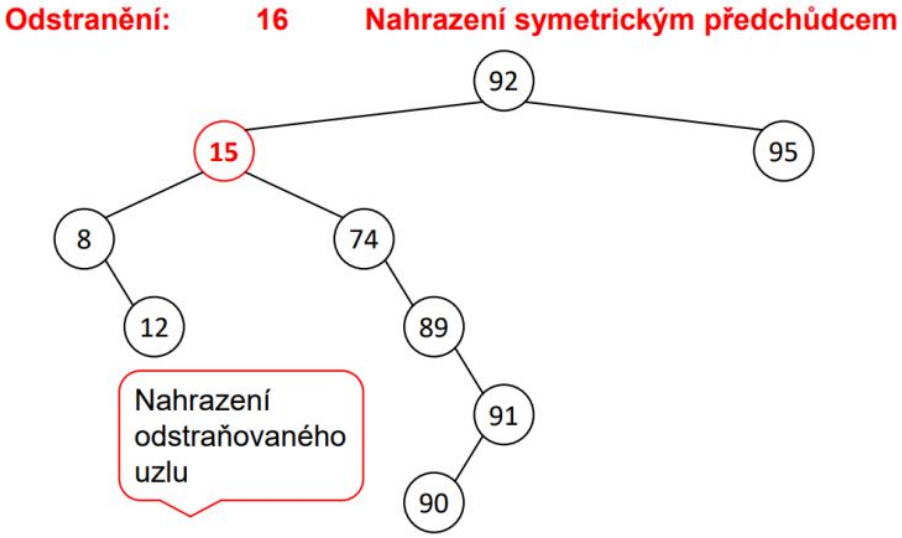
Pokud aktuální uzel odpovídá X, tak už X ve stromě je - konec
Pokud je aktuální uzel větší než X, tak X pokračuje do levého potomka
Pokud je aktuální uzel menší než X, tak X pokračuje do levého potomka



Odebrání prvku X

Lze nahradit buď největším menším klíčem nebo nejmenším větším klíčem
Ve vizualizacích většinou nahrazujou největším menším klíčem (pokud lze)





Reprezentace položek AVL stromem

- výškově vyvážený binární vyhledávací strom
- pro všechny uzly platí, že výška levého a pravého podstromu se liší maximálně o 1
- vyvážení stromu se provádí po každé operaci insert/delete
- každý uzel stromu má proměnnou reprezentující stupeň vyvážení
    - 0 oba podstromy stejně vysoké
    - 1 pravý podstrom o 1 vyšší
    - 2 pravý podstrom o 2 vyšší -> potřeba vyvážit
    - -1 levý podstrom o 1 vyšší
    - -2 levý podstrom o 2 vyšší -> potřeba vyvážit

Reprezentace položek B stromem (konkrétně řádu m = 5)

- m=počet ukazatelů (každý uzel krom kořene obsahuje 2 až m-1 klíče)
- Nový prvek se vždy vkládá do listové stránky, ve stránce se klíče řadí podle velikosti.
- Pokud dojde k přeplnění listové stránky, stránka se rozdělí na dvě a prostřední klíč se přesune do nadřazené stránky (pokud nadřazená stránka neexistuje, tak se vytvoří)
- Pokud dojde k přeplnění nadřazené stránky předchozí postup se opakuje dokud nedojde k zařazení nebo k vytvoření nového kořene

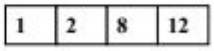
- B-strom řádu *m* je strom, kde každý uzel má maximálně m následníků a ve kterém platí:
    1. Počet klíčů v každém vnitřním uzlu, je o jednu menší, než je počet následníků (synů)
    2. Všechny listy jsou na stejné úrovni (mají stejnou hloubku)
    3. Všechny uzly kromě kořene mají nejméně (*m*/2) následníků ((*m*/2)-1 klíčů)
    4. Kořen je buďto list, nebo má od 2 do *m* následníků
    5. Žádný uzel neobsahuje více než *m* následníků (m-1 klíčů)

Vkládání

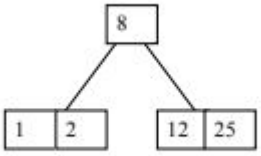
klíče: 1; 12; 8; 2; 25; 6; 14; 28; 17; 7; 52; 16; 48; 68; 3; 26; 29; 53; 55; 45

1. přidáme prvek do uzlu

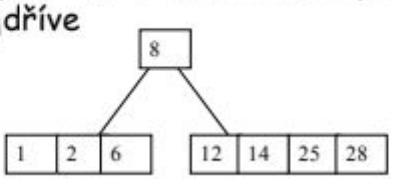
- První 4 klíče :



2. když už by uzel obsahoval m klíčů tak prostřední prvek dáme o patro vejš, stávající patro se rozpůlí na obrázku přidání klíče 25

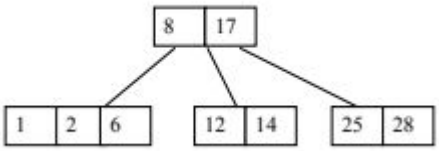


Další položky 6, 14, 28 budou vloženy do listů (listy se obsazují nejdříve

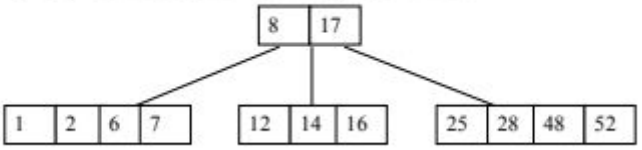


přidání 17
lze vidět jak:

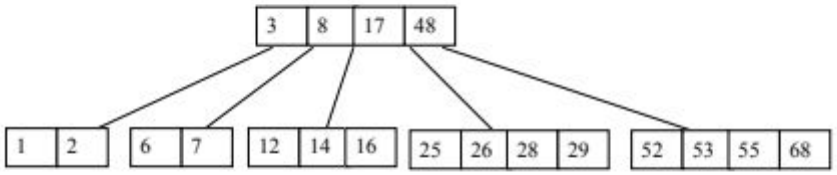
- 1 2 6 jsou před 8
- 12 14 jsou mezi 8 a 17
- 25 28 jsou za 17



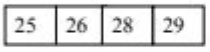
7, 52, 16, 48 se opět přidají do listů



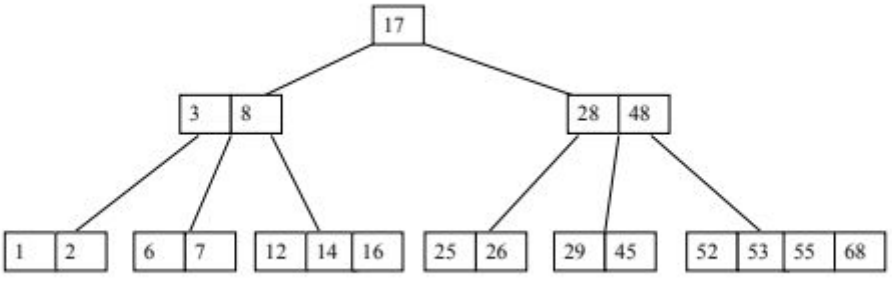
Vložení 68 opět způsobí přeplnění stránky vpravo, klíč 48 se přesune do kořene, 3 přeplní levou stránku a po rozdělení přechází do kořene; 26, 29, 53, 55 jsou vloženy do listů



45 přeplní stránku

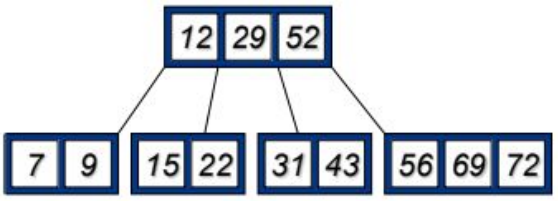


a klíč 28 se přesune do kořene, kde způsobí přeplnění a rozdělení kořenové stránky

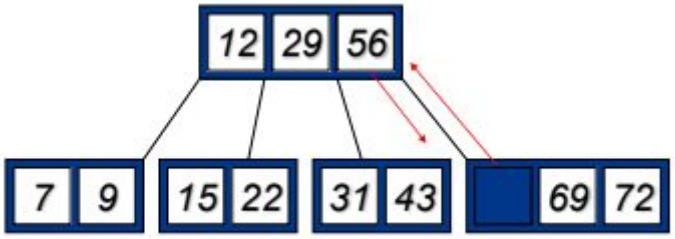


Mazání

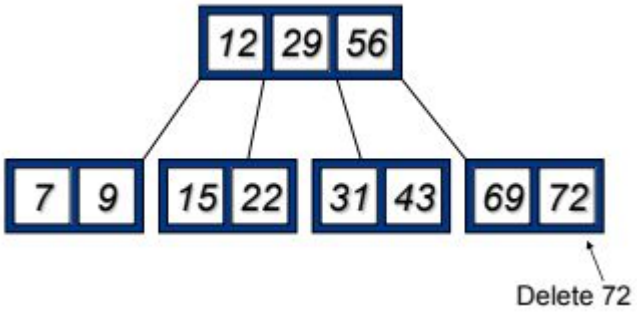
1) Když máme ve stránce dostatečné množtví klíču tak prostě hodnotu smažem

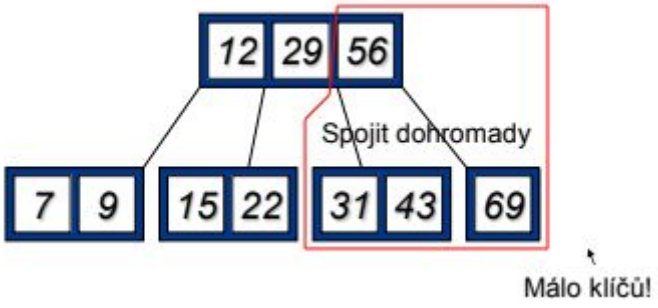


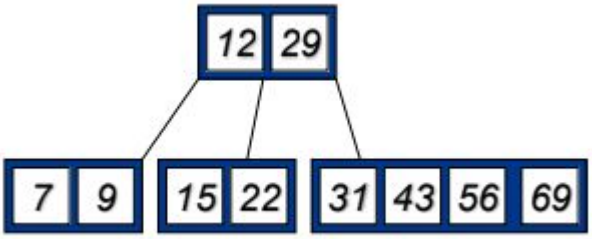
2) Když nerušíme list, ale mamé dostatek klíčů tak neřešíme a jen doplníme z listů (delete 52, nahradíme jí 56)



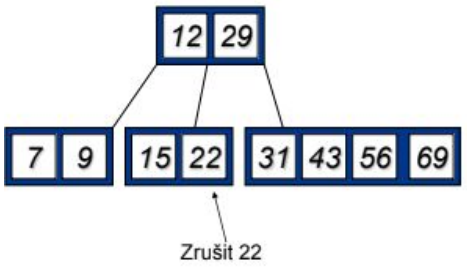
3) Když máme minimální počet klíčů a sousední stránky taky min. klíčů tak se dolní stránka spojí tady delete 72, samotná 69 je málo

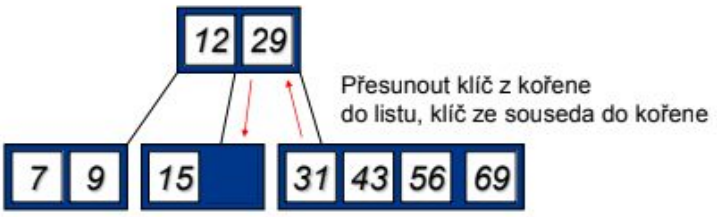


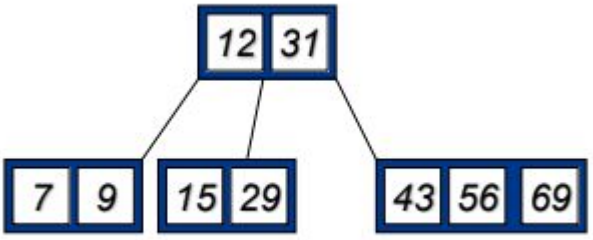




4) Když máme ve stránce minimální počet klíčů, ale sousední stránky obsahujou dostatek klíčů, tak přesunem klíč z kořene do listu a z vedlejšího souseda do kořene


12 29
7 9
15 22
31 43 56 69
Zrušit 22


12 29
7 9
15
31 43 56 69
Přesunout klíč z kořene
do listu, klíč ze souseda do kořene


12 31
7 9
15 29
43 56 69

Nejdelší společná sekvence znaků (LCS) (tabulka)

1) V tabulce u X a Y 0

X = DACDA a Y = ABCA

			Y_j				
	j	0	1	2	3	4
	i		{ }	A	B	C	A
X_i	0	{ }	0	0	0	0	0
	1	D	0				
	2	A	0				
	3	C	0				
	4	D	0				
	5	A	0				

2) Prochází se postupně porovnávání vždy x se všema y
3) Pokud se x[i] != y[j] tak se napíše větší ze dvou hodnot na pozicích x-1 a y-1 a šipka na tu větší hodnotu, když jsou shodné tak se preferuje nahoru

			Y_j				
	j	0	1	2	3	4
	i		{ }	A	B	C	A
X_i	0	{ }	0	0	0	0	0
	1	D	0	0↑			
	2	A	0				
	3	C	0				
	4	D	0				
	5	A	0				

4) Pokud se x[i] == y[j] tak se napíše (hodnota na pozici o jedna menší na obou osách) + 1 a šipka směrem tam

			Y_j				
	j	0	1	2	3	4
	i		{ }	A	B	C	A
X_i	0	{ }	0	0	0	0	0
	1	D	0	0↑	0↑	0↑	0↑
	2	A	0	1↖			
	3	C	0				
	4	D	0				
	5	A	0				

5) Takhle furt dokola

			Y_j				
	j	0	1	2	3	4
	i		{}	A	B	C	A
X_i	0	{}	0	0	0	0	0
	1	D	0	0↑	0↑	0↑	0↑
	2	A	0	1↖	1←	1←	1↖
	3	C	0	1↑	1↑	2↖	2←
	4	D	0	1↑	1↑	2↑	2↑
	5	A	0	1↖	1↑	2↑	3↖

Pak se jde po šipkách vodspoda, tam kde se potkají písmena na osách je stejná část řetězce

SCS

Podobný jako LCS
Když se x[i]==y[j] tak stejný jak LCS
akorát když se nerovnají, tak se bere min(c[i, j − 1] + 1, c[i − 1, j] + 1) (o jedno vlevo +1 a o jedno nahoru +1)

- **Najděte nejkratší společný nadřetězec pro řetězce:**

  X = ABCD a Y = BDAC

			Y_j				
	j	0	1	2	3	4
	i		{}	B	D	A	C
X_i	0	{}	0	1←	2←	3	4
	1	A	1	2↑	3↑	3↖	4←
	2	B	2	2↖	3←	4↑	5↑
	3	C	3	3↑	4↑	5↑	5↖
	4	D	4	4↑	4↖	5←	6↑

$SCS(X, Y)$ = BDABCD

X: BDABCD
Y: BDABCD

Vzdálenost mezi řetězci když všechny chybové transformace mají váhu 1

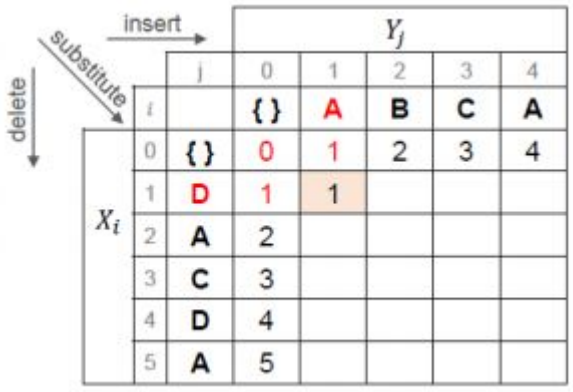
- **Stanovte edit distance pro řetězce:**

  X = DACDA a Y = ABCA

insert, substitute, delete

			Y_j				
	j	0	1	2	3	4
	i		{}	A	B	C	A
X_i	0	{}	0	1	2	3	4
	1	D	1				
	2	A	2				
	3	C	3				
	4	D	4				
	5	A	5				

Zase porovnání x se všema y
Pokud se nerovná tak se bere min(d[i-1, j-1]+1, d[i-1, j]+1, d[i, j-1]+1)



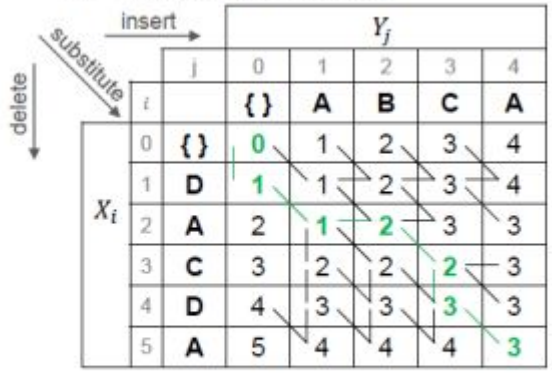
i	j	0	1	2	3	4
		{ }	A	B	C	A
0	{ }	0	1	2	3	4
1	D	1	1			
2	A	2				
3	C	3				
4	D	4				
5	A	5				

$$x[1] \neq y[1]$$
$$\Rightarrow d[1,1] = \min(1,2,2)$$

Pokud se rovná tak se bere min(d[i-1, j-1], d[i-1, j]+1, d[i, j-1]+1)

Je třeba zaznamenávat všechny pozice kde minimum nastalo

X = DACDA a Y = ABCA



i	j	0	1	2	3	4
		{ }	A	B	C	A
0	{ }	0	1	2	3	4
1	D	1	1	2	3	4
2	A	2	1	2	3	3
3	C	3	2	2	2	3
4	D	4	3	3	3	3
5	A	5	4	4	4	3

Tranformace (zelená cesta):
- delete(D) → –
- substitute(A, A) → A
- insert(B) → B
- substitute(C, C) → C
- delete(D) → –
- substitute(A, A) → A

->Změny byly 3, pokud zaměňujeme stejná písmena-NEPOČÍTÁ SE TO jako změna.

Znázornění grafu G maticí sousednosti

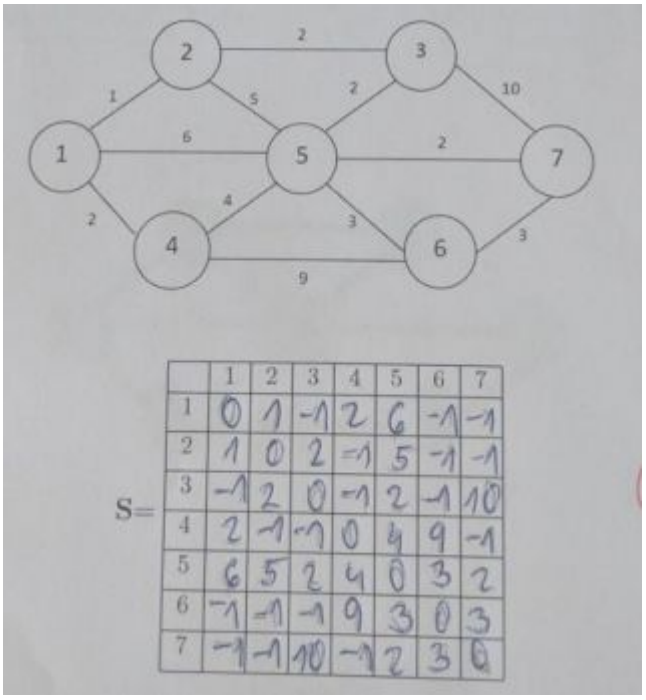
Řádky - odkud

Sloupce - kam

Hodnota 0 pokud je to stejnej vrchol (na diagonale)

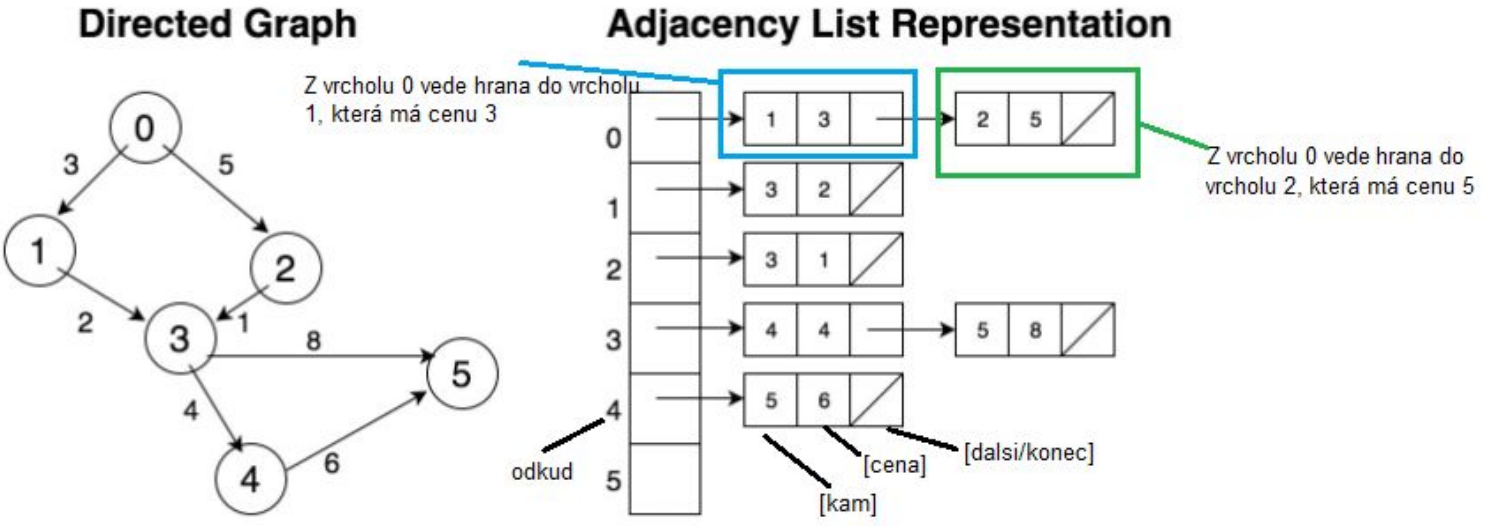
Hodnota -1 pokud žádná cesta nevede

Hodnota ceny pokud vede cesta z vrcholu do vrcholu



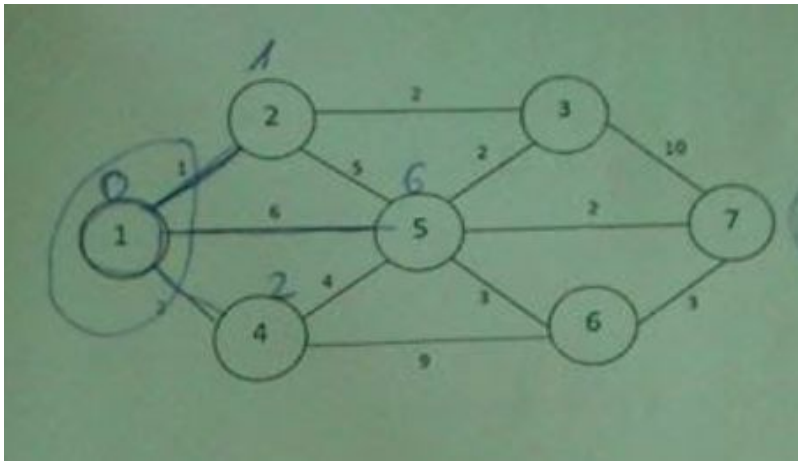
	1	2	3	4	5	6	7
1	0	1	-1	2	6	-1	-1
2	1	0	2	-1	5	-1	-1
3	-1	2	0	-1	2	-1	10
4	2	-1	-1	0	4	9	-1
5	6	5	2	4	0	3	2
6	-1	-1	-1	9	3	0	3
7	-1	-1	10	-1	2	3	0

Znázornění grafu G seznamem sousednosti

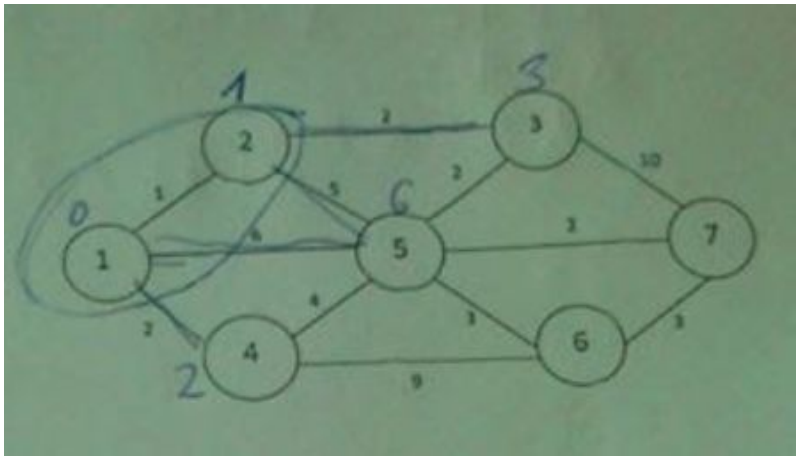


Djikstrův algoritmus

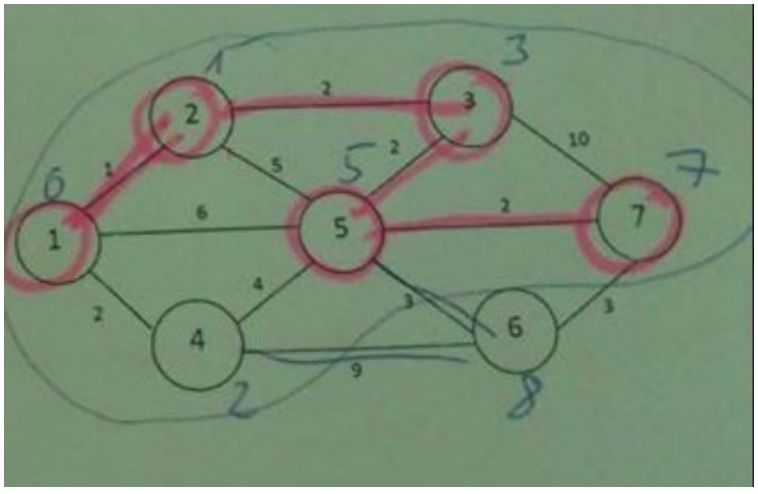
1) V počátečním vrcholu označim všechny vrcholy do kterých je cesta její cenou a přidám počáteční vrchol do mraku



2) Kouknu, kterej další vrchol má nejmenší cenu a přejdu do něj
3) Projdu všechny vrcholy, do kterých je cesta a ještě nejsou v mraku, aktualizuju cenu cest
4) Přidám vrchol do mraku



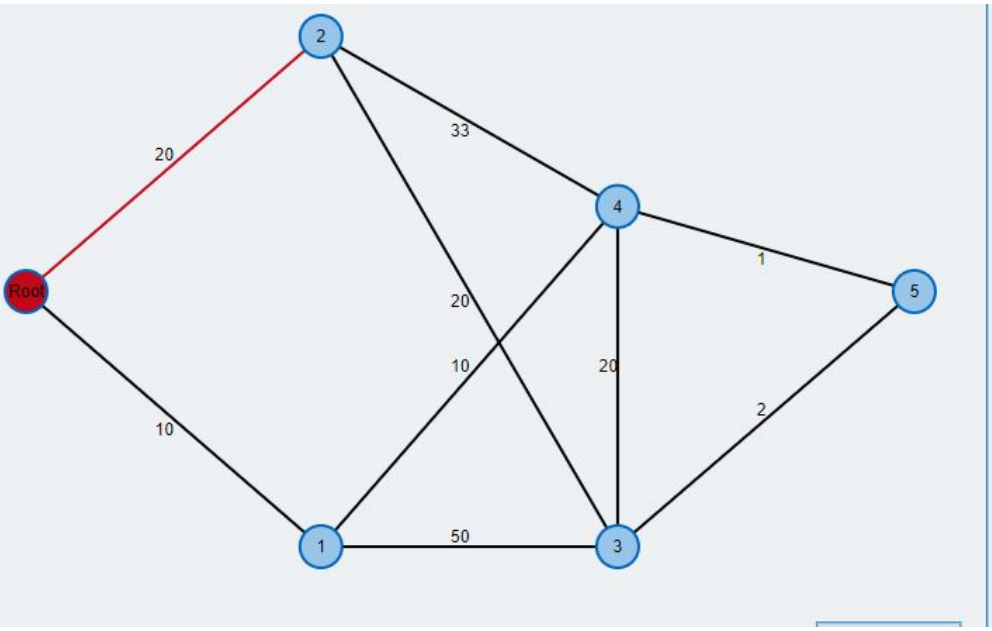
5) Back to 2) LOL



U tabulky vždy v řádku napíšu cenu procházenejch vrcholů a označim ten, kterej bude další procházenej
V další řádce pak projdu z toho vrcholu ceny cest a ty který se nezměnily jen opíšu a zas vyberu další nejmenší

Prim-Jarnikův algoritmus (minimální kostra grafu)

1) Vezmu první vrchol (root) a přidám do mraku
2) Vezmu souseda, když ještě nebyl zpracovanej, přidám do fronty podle ceny cesty(da se preskakovat)
3) Pokud je už ve frontě ale nová cesta je kratší, změním jeho pozici ve frontě
4) Dělám 2), dokud nějací sousedi jsou
5) Vezmu z fronty vrchol s nejmenší cenou cesty a přidám ho do mraku
6) Back to 2)



Algorithm status

prev | next | fast forward

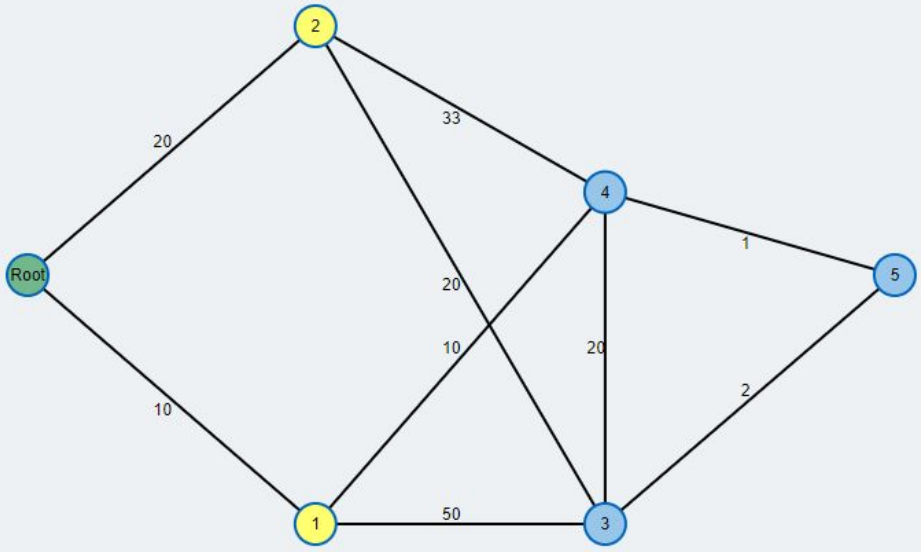
Explanation | Pseudocode

Considering the neighbour

Each neighbouring node is processd for possible modifications.

There are two cases in which the node is not further processed. First is the case when the node has been added to the MST before. Second is the case when the node is in the queue, but the previous edge connnecting it to the MST has a lower weight

Queue:



SVG Download

+ Legend

Algorithm status

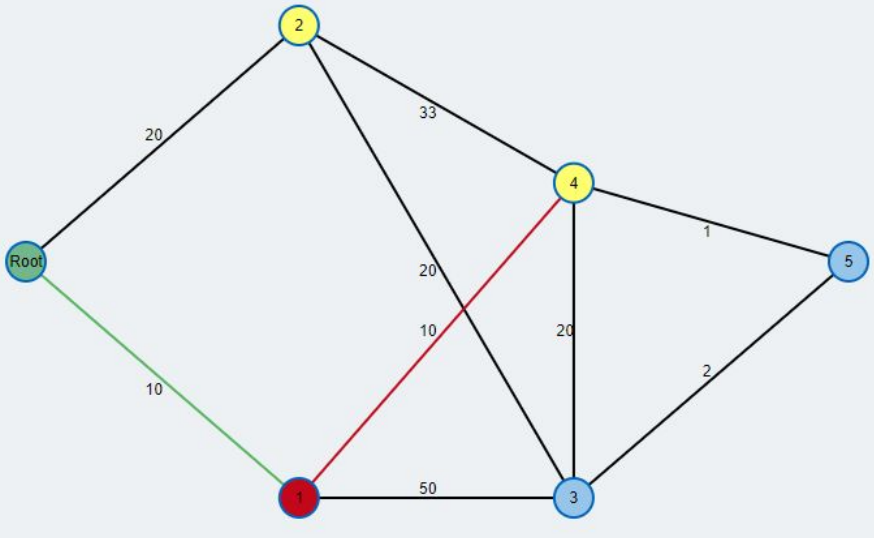
prev | next | fast forward

Explanation | Pseudocode

All outgoing edges of the node are processed

Queue:

1 | 2



SVG Download

+ Legend

Algorithm status

prev | next | fast forward

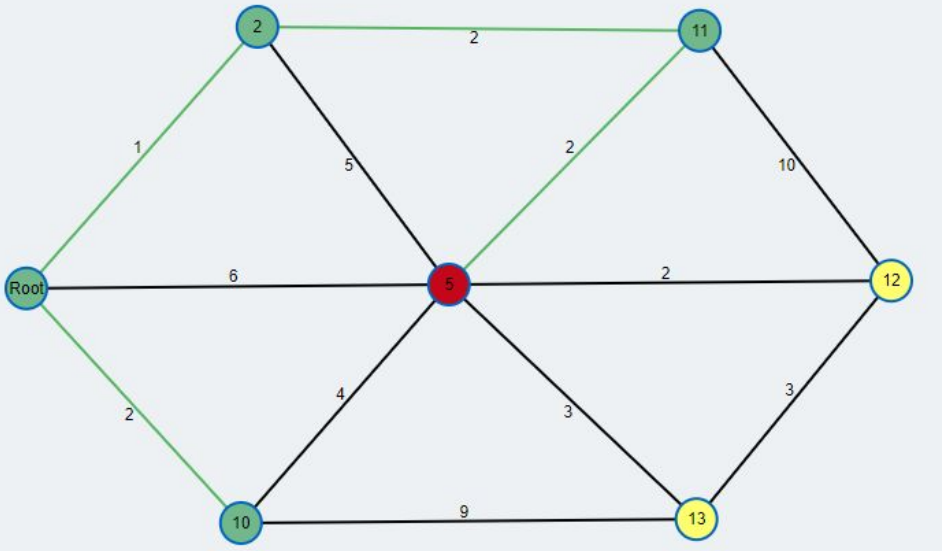
Explanation | Pseudocode

Not yet visited neighboring nodes

If the node is not already added to the MST, then it must be processed later. In this case, the node is added to the queue.

Queue:

4 | 2



SVG Download

+ Legend

Algorithm status

prev | next | fast forward

Explanation | Pseudocode

Process next node

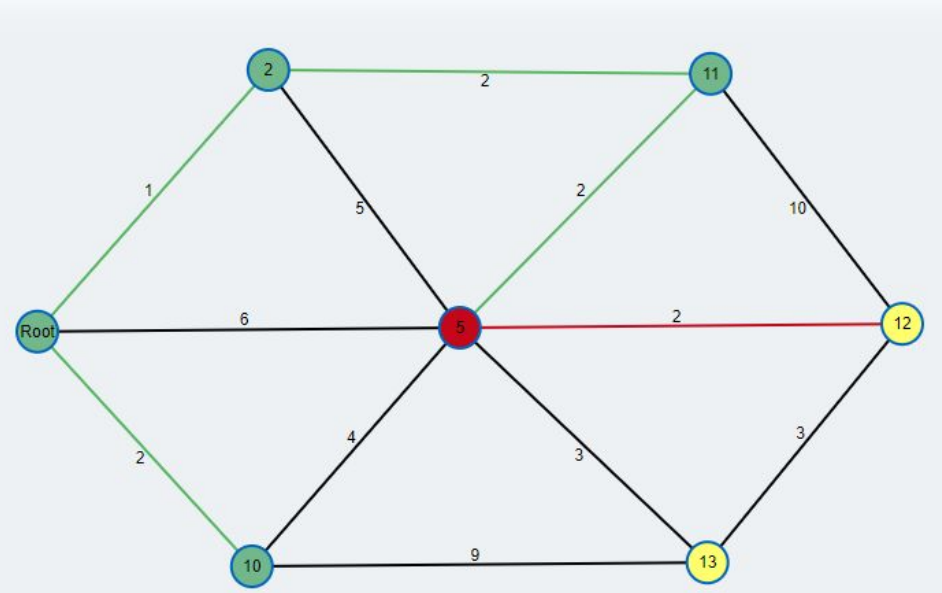
The node with the smallest distance value is removed from the queue, and the edge connecting it to the MST is added to the resulting set. The only exception is the root node which has a parent value set to itself.

As this node is being further processed, it is colored red.

The edges added to the tree are shown in green.

Queue:

13 | 12



SVG Download

+ Legend

Algorithm status

prev | next | fast forward

Explanation | Pseudocode

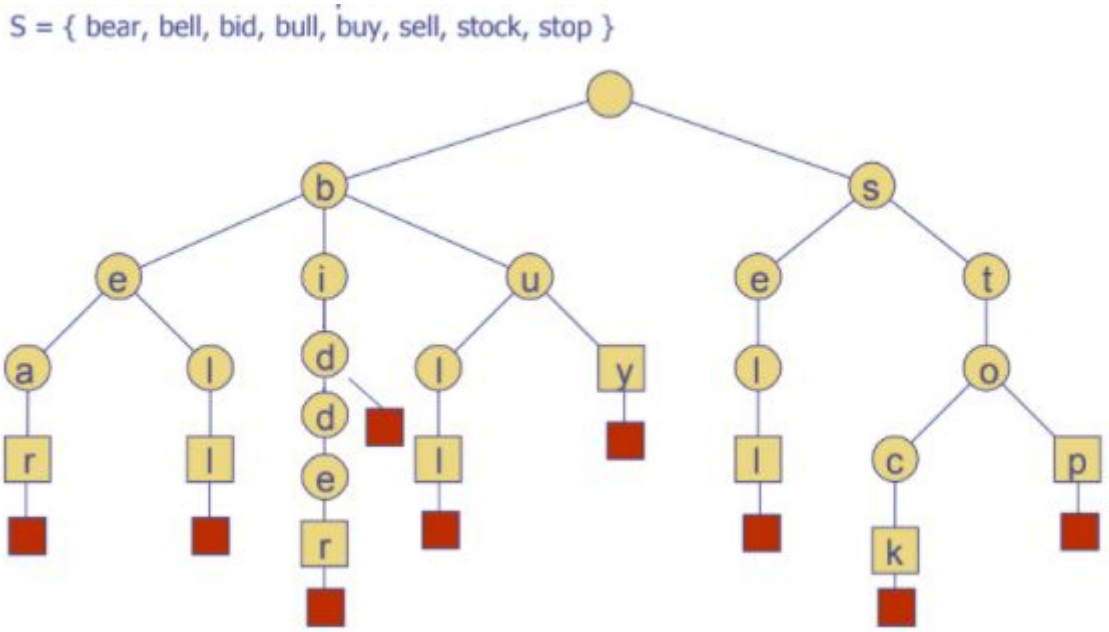
Node already in the queue

If a node has already been added to the queue for further processing, then it is possible that the new edge is shorter than the previous one. In this case, the priority of the neighbour is set to the new value.

Queue:

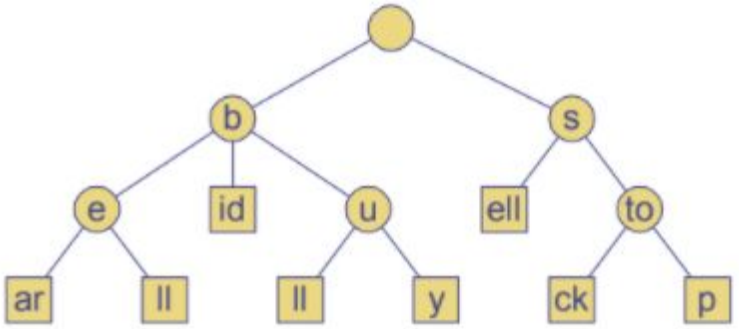
12 | 13

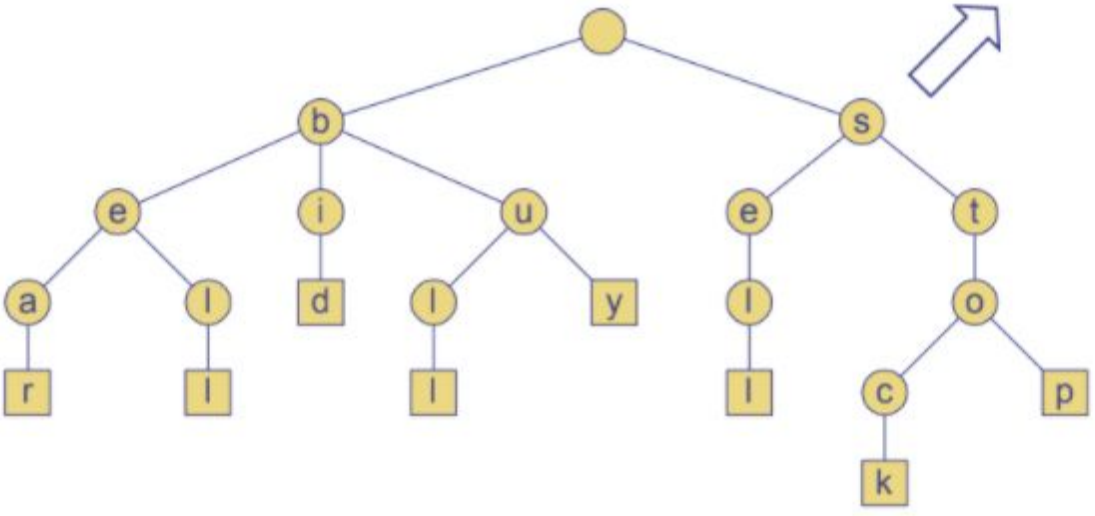
Struktura Trie



Komprimovaná struktura Trie

- Komprimovaná trie má vnitřní uzly stupně nejméně 2
- Získáva se ze standardní trie, kompresí řetězců tzv. „redundantních" uzlů tj. uzlů, které mají pouze jednoho následníka



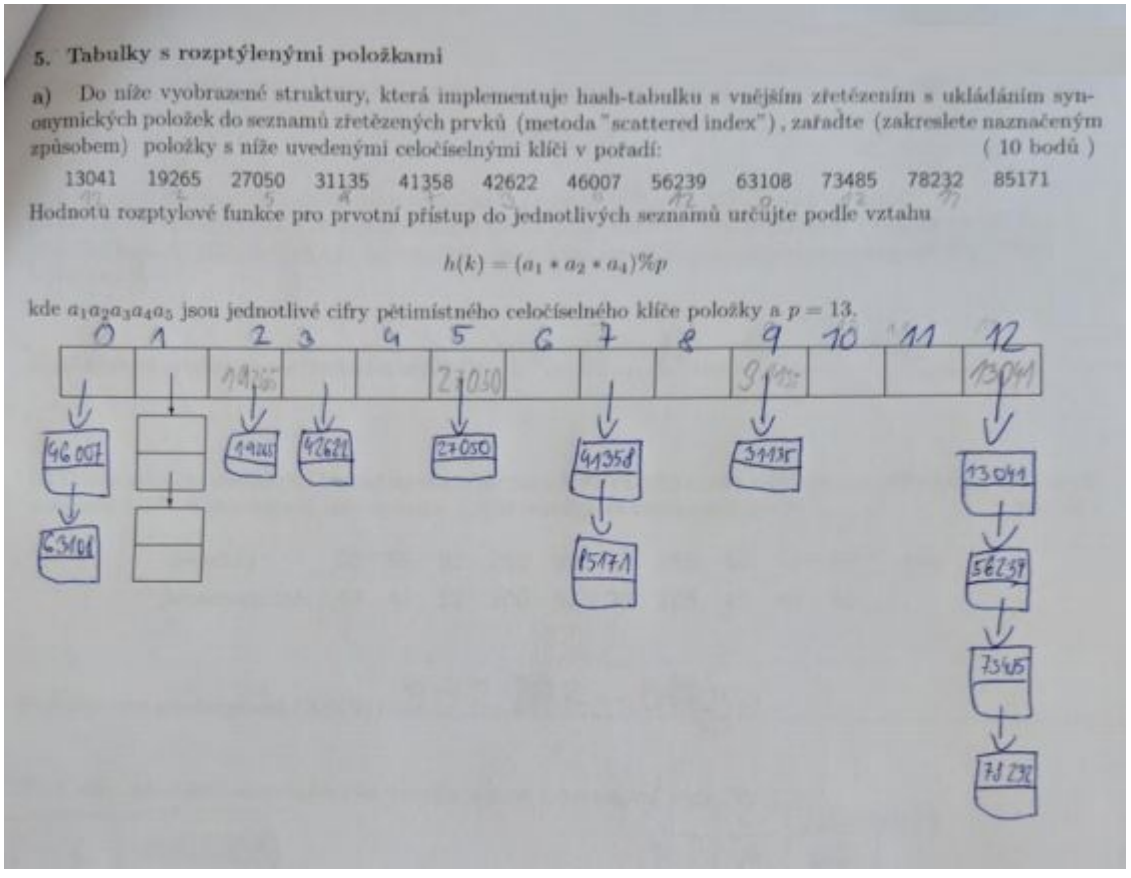


Suffixová struktura Trie

V uzlech je dvojice indexů j,k
j je počáteční index podřetězce uloženého v uzlu
k je koncový index podřetězce uloženého v uzlu

Doplnění hodnot do hash tabulky se zadanou rozptylovou funkcí

1) Stačí jen doplnit hodnoty do rozptylový funkce a vyjde index


5. **Tabulky s rozptýlenými položkami**

a) Do níže vyobrazené struktury, která implementuje hash-tabulku s vnějším zřetězením s ukládáním synonymických položek do seznamů zřetězených prvků (metoda "scattered index"), zařaďte (zakreslete naznačeným způsobem) položky s níže uvedenými celočíselnými klíči v pořadí: (10 bodů)

13041 19265 27050 31135 41358 42622 46007 56239 63108 73485 78232 85171

Hodnotu rozptylové funkce pro prvotní přístup do jednotlivých seznamů určujte podle vztahu

$$h(k) = (a_1 * a_2 * a_4)\%p$$

kde $a_1a_2a_3a_4a_5$ jsou jednotlivé cifry pětimístného celočíselného klíče položky a $p = 13$.

Střední algoritmická složitost vyhledávání položek

netušim ale je to jen za dva body

Huffmanův kód

1) Určíme počet jednotlivých znaků

- Zadaný řetězec: **HRADEC_KRALOVE** (znak _ značí mezeru)

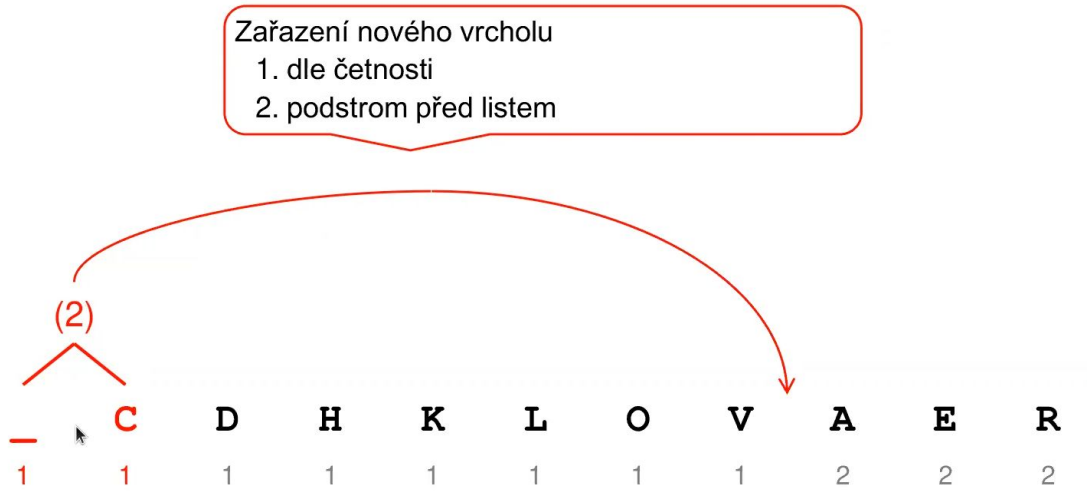
znak	četnost
H	1
R	2
A	2
D	1
E	2
C	1

znak	četnost
_	1
K	1
L	1
O	1
V	1

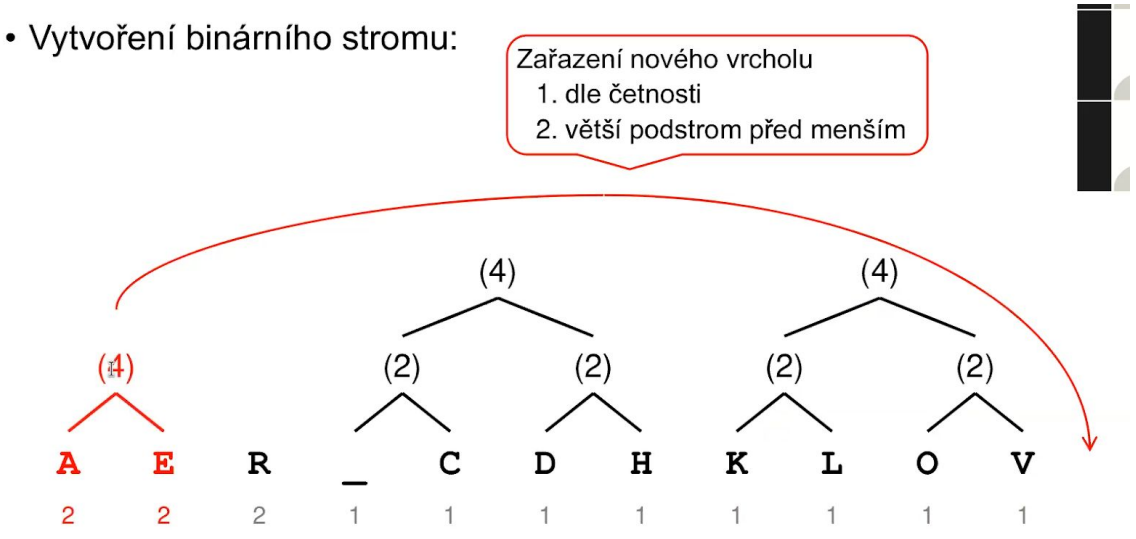
2) Seřadíme posloupnost znaků (primárně podle četnosti, pak větší podstrom před menším a nakonec podle abecedy)

_	C	D	H	K	L	O	V	A	E	R
1	1	1	1	1	1	1	1	2	2	2

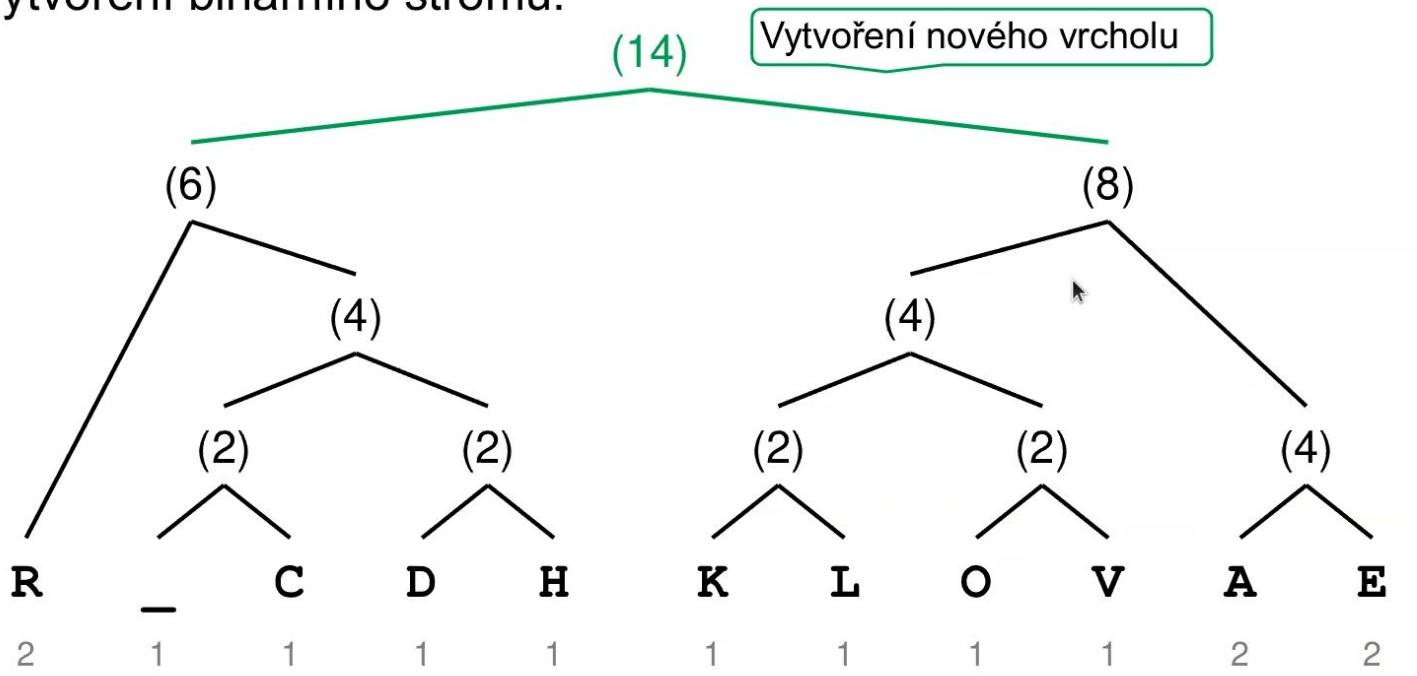
3) Potom zleva dvě vždy spojíme a seřadíme do posloupnosti znaků



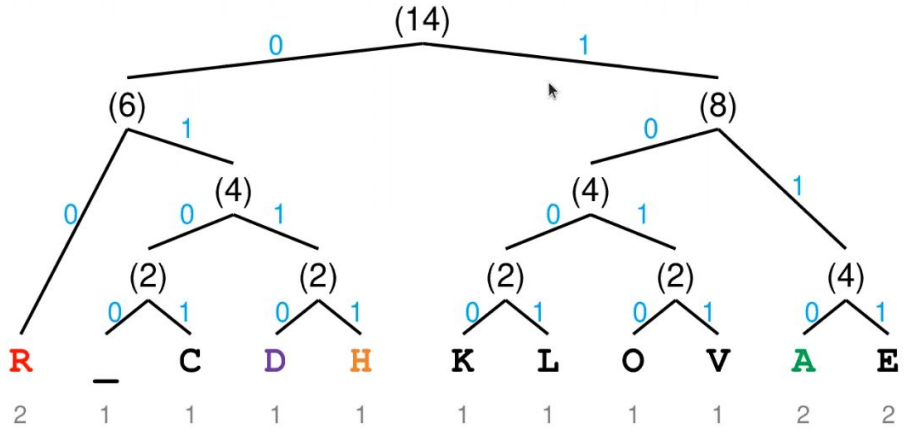
- Vytvoření binárního stromu:



- Vytvoření binárního stromu:

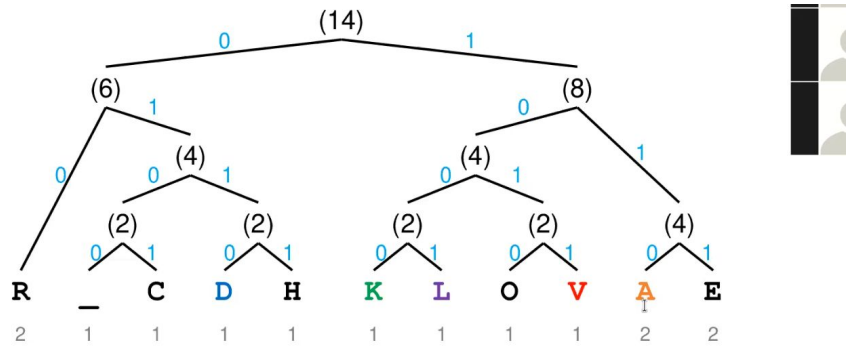


Kódování a zakodování pomocí vytvořenýho stromu provádíme tak, že vlevo je 0 a vpravo je 1, tak dojdem na každý písmeno



Zakódování slova **DRAHA**:

01100011000111110



Postupovat z kořene, dokud není nalezen list, pak návrat do kořene

Dekódovat sekvenci **10111000100111100110**:

VKLAD

LZW komprese dat (komprimace posloupnosti znaků)

```
S = načti znak ze vstupu
while(další znaky na vstupu){
      C = načti znak ze vstupu
      if((S+C) je v kódovací tabulce)
            S = S+C
      else{
            vypiš kód pro S
            přidej do tabulky (S+C)
            S = C
      }
}
vypiš kód pro S
```

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A		

Do S načti znak ze vstupu

ABRABABRA

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A	B	

Do C načti znak ze vstupu

ABRABABRA

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A	B	65 (A)
B		

S+C není v kódovací tabulce
⇒
vypiš S
(S+C) do tabulky
S = C

Kódovací tabul

kód	posloupn
0..255	jednotliv znaky
256	AB

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A	B	65 (A)
B	R	66 (B)
R		

S+C není v kódovací tabulce
⇒
vypiš S
(S+C) do tabulky
S = C

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky
256	AB
257	BR

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A	B	65 (A)
B	R	66 (B)
R	A	82 (R)
A	B	
AB		

S+C je v kódovací tabulce
⇒
S = S+C

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky
256	AB
257	BR
258	RA

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A	B	65 (A)
B	R	66 (B)
R	A	82 (R)
A	B	
AB	A	256 (AB)
A		

S+C není v kódovací tabulce
⇒
vypiš S
(S+C) do tabulky
S = C

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky
256	AB
257	BR
258	RA
259	ABA

ABRABABRA

Postup kódování

S (prefix)	C (sufix)	Výstup (kód)
A	B	65 (A)
B	R	66 (B)
R	A	82 (R)
A	B	
AB	A	256 (AB)
A	B	
AB	R	256 (AB)
R	A	
RA		258 (RA)

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky
256	AB
257	BR
258	RA
259	ABA
260	ABR

Výsledný výstup:

65 66 82 256 256 258

Dekomprese informace, která byla zkomprimována LZW

```
OLD_CODE = načti kód ze vstupu
vypiš posloupnost zakódovanou kódem OLD_CODE
while(další kódy na vstupu){
      NEW_CODE = načti kód ze vstupu
      if(NEW_CODE není v kódovací tabulce){
            S = posloupnost zakódovaná kódem OLD_CODE
            S = S+C
      }
      else
             S = posloupnost zakódovaná kódem NEW_CODE
      vypiš S
      C = první znak S
      přidej do tabulky (OLD_CODE+C)
      OLD_CODE = NEW_CODE
}
```

75 85 256 82 257 260

Postup dekódování

OLD_CODE	NEW_CODE	S	C	Výstup
75 (K)				K

Do OLD_CODE načti kód ze vstupu a zapiš na výstup

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky

Postup dekódování

OLD_CODE	NEW_CODE	S	C	Výstup
75 (K)				K
75 (K)	85 (U)	U	U	U
85 (U)				

NEW_CODE je v kódovací tabulce
⇒
S = posloupnost zakódovaná NEW_CODE

- vypiš S
- C = první znak S
- (OLD_CODE+C) do tabulky
- OLD_CODE = NEW_CODE

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky
256	KU

75 85 256 82 257 260

Postup dekódování

OLD_CODE	NEW_CODE	S	C	Výstup
75 (K)				K
75 (K)	85 (U)	U	U	U
85 (U)	256 (KU)	KU	K	KU
256 (KU)	82 (R)	R	R	R
82 (R)	257 (UK)	UK	U	UK
257 (UK)	260	UKU	U	UKU
260 (UKU)				

Kódovací tabulk

kód	posloupnc
0..255	jednotliv znaky
256	KU
257	UK
258	KUR
259	RU
260	UKU

Výsledný výstup:

KUKURUKUKU

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

256 je v hex 100

257 je 101 atd.