Vyhledávací stromy *Patricia Trie*

Stromy *trie* mají nevýhody:

- Vnitřní a listové uzly jsou rozdílné (ve vnitřních uzlech prvek uložen není, v listových ano), což komplikuje efektivní implementaci.
- Ve stromu se vyskytují za sebou i sekvence vnitřních uzlů, ve kterých není žádné větvení, což citelně zvyšuje počet uzlů ve stromu.

Stromy *Patricia trie* mají obdobně jako jiné binární vyhledávací stromy v každém uzlu uložen prvek. Počet uzlů je tím opět stejný jako počet uložených prvků. Další jejich vlastnosti jsou:

- V každém uzlu je uvedeno, podle kterého bitu je v tomto uzlu větvení (index bitu větvení).
- Každý uzel má dva odkazy na další uzly (odkaz může být i na sebe).
- Všechny uzly (nelistové i listové) mají stejnou strukturu:

  datový prvek + index bitu větvení + 2 odkazy na další uzly
- Kořenem stromu je specifický uzel, ve kterém je uložen nulový prvek a je v něm odkaz na první z uzlů, ve kterém jsou uloženy prvky. Kořen budeme označovat jako hlavu stromu. Nulový prvek (ve všech bitech má hodnotu 0) má v *patricia trie* specifickou úlohu. Do stromu proto nelze vložit prvek, který má stejnou hodnotu jako nulový prvek.

Vyhledání prvku

1. Počáteční krok

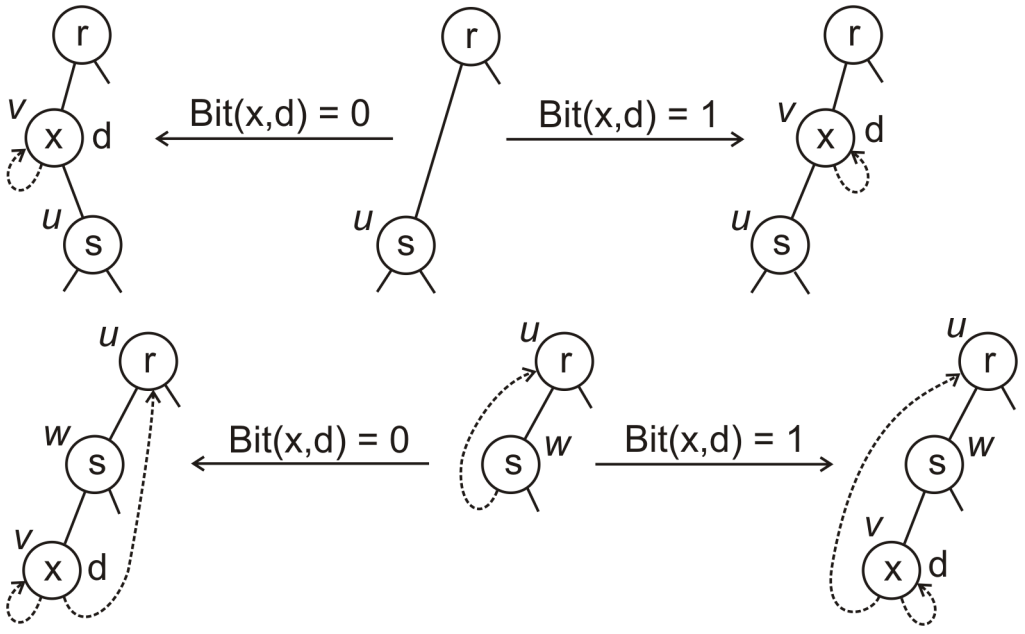
   Uzel, který je v daném okamžiku vyhledávání aktuální, budeme označovat *u*. Na začátku jím bude uzel, na který je levý odkaz v hlavě stromu.
   Hledaný prvek nechť je *x*.
   Aktuální index bitu větvení označíme *d*. Jeho hodnotu na začátku nastavíme na -1.
   Index bitu uložený v uzlu *u* udávající, podle kterého bitu je v uzlu *u* větvení, budeme označovat *u.bit*.
   Pro zjištění hodnoty bitu prvku *x* s indexem *d* budeme používat funkci *Bit*(*x*,*d*).
2. Průběžný krok
   - Je-li index bitu *u.bit* uložený v aktuálním uzlu *u* menší nebo roven indexu *d*, je to příznak, že jsme se dostali na konec vyhledávání. Srovnáme, zda prvek uložený v aktuálním uzlu *u* je roven hledanému prvku *x*. Pokud ano, prvek byl nalezen. Jinak hledání končí neúspěšně.

- Jestliže nejsme na konci vyhledávání, do indexu *d* uložíme hodnotu bitu větvení aktuálního uzlu *u.bit*. Zjistíme hodnotu *Bit*(*x*,*d*). Je-li tato hodnota 0, učiníme aktuálním uzlem levého následníka současného uzlu *u*, jinak jím bude jeho pravý následník. A vyhledávání opět pokračuje krokem 2.

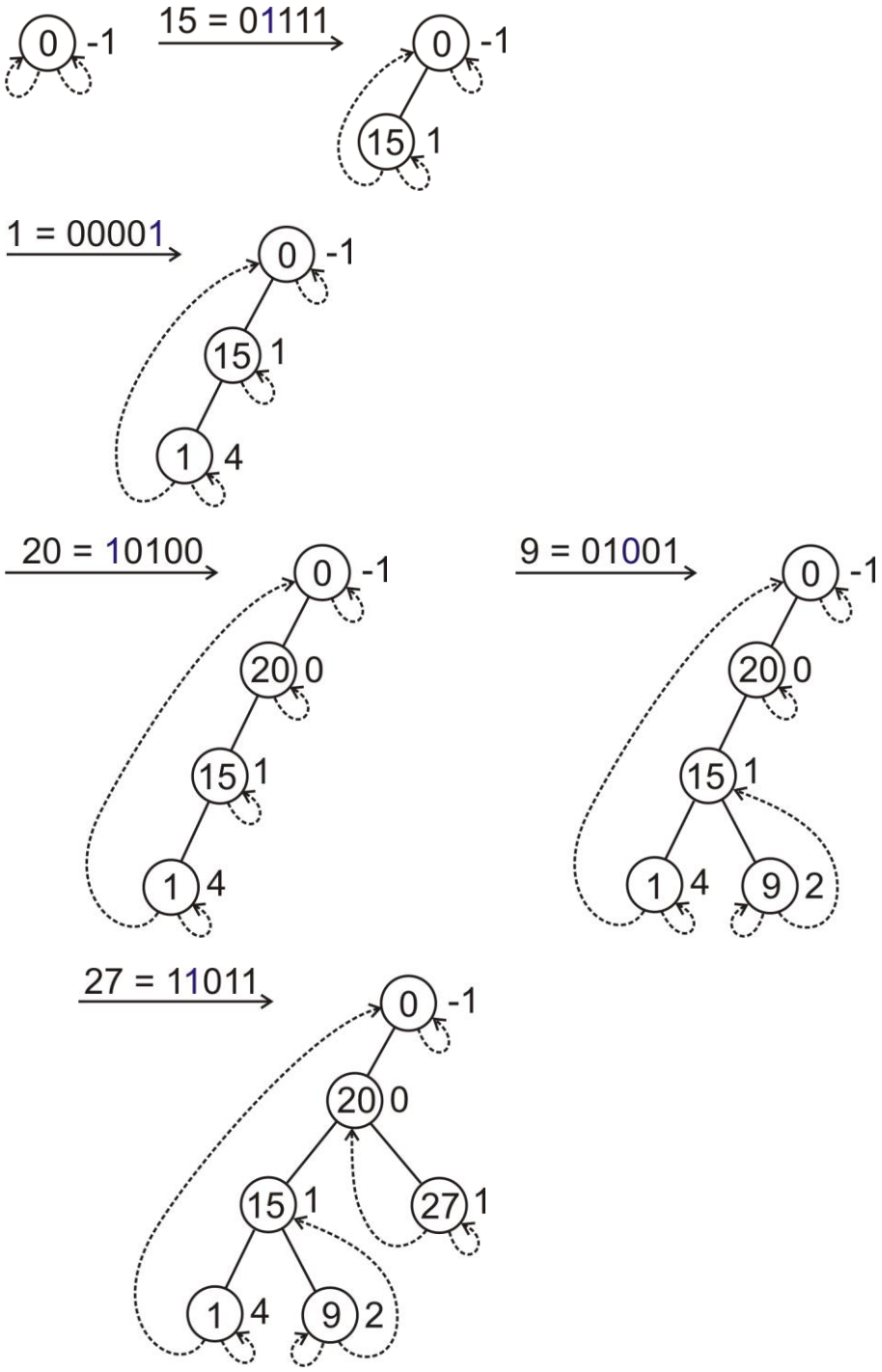
Přidání prvku

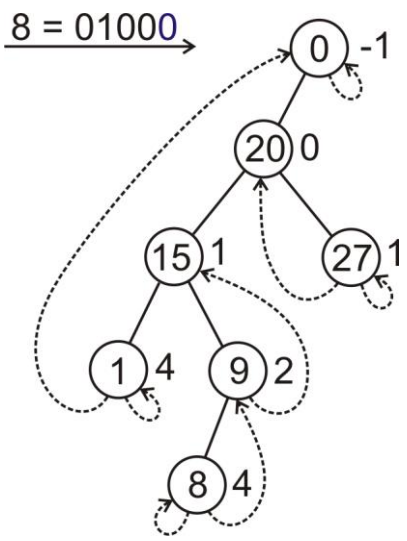
Přidávaný prvek *x* nejprve vyhledáme. Nechť vyhledání skončilo v uzlu, ve kterém je prvek *y*.

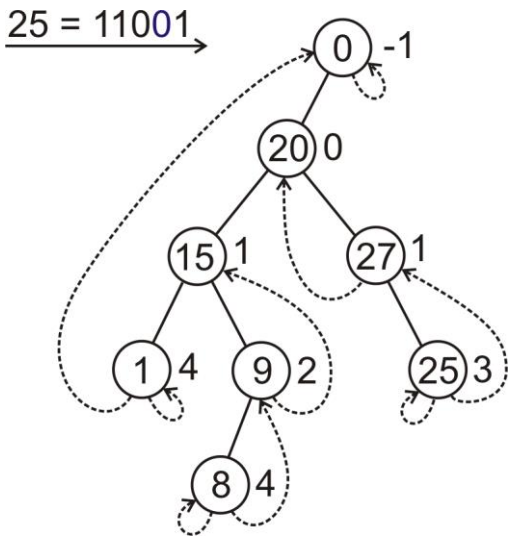
- Je-li prvek *y* obsažený v uzlu, ve kterém vyhledávání skončilo, roven hledanému prvku *x*, přidávání tím končí.
- Pokud nebyl prvek *x* ve stromu nalezen, začneme postupně srovnávat od začátku (zleva) jednotlivé bity prvku *x* s odpovídajícími bity prvku *y*, až najdeme první bit, ve kterém se tyto prvky liší. Označme index tohoto bitu *d*. Následně začneme opět procházet strom od začátku po stejné cestě, po které jsme vyhledávali přidávaný prvek *x*, tak dlouho, dokud:
  - nenajdeme uzel *u*, jehož index bitu *u.bit* je větší než index *d*. Pak před tento uzel vložíme do stromu nový uzel *v*, do kterého dáme přidávaný prvek *x* a index bitu *v.bit* tohoto uzlu nastavíme na *d*.
  - anebo se dostaneme do uzlu *u*, ve kterém vyhledávání skončilo. Pak nový uzel *v* s prvkem *x* vložíme pod uzel *w*, z kterého jsme k uzlu *u* dostali (vložíme ho do hrany vedoucí z uzlu *w* nahoru do uzlu *u*) a údaj indexu bitu *v.bit* nového uzlu nastavíme na *d*.
- Při vkládání nového uzlu *v* bude jeho levý odkaz na tento uzel (na sebe), jestliže *Bit*(*x*,*d*) má hodnotu 0. Je-li hodnota tohoto bitu rovna 1, je naopak jeho pravý odkaz na tento uzel (na sebe).

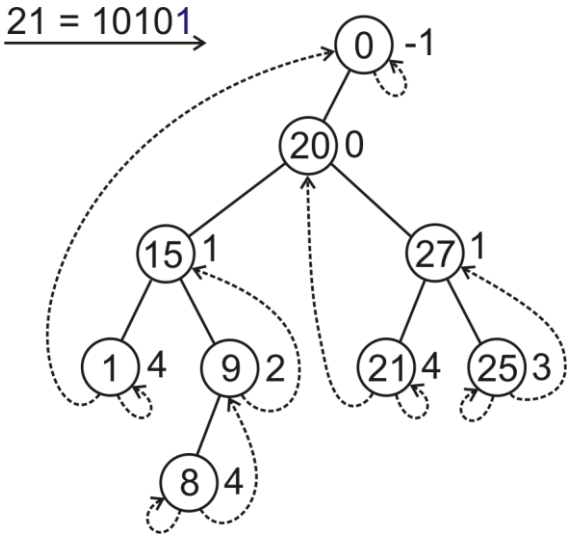


Příklad. Do stromu budeme ukládat pětibitová čísla.



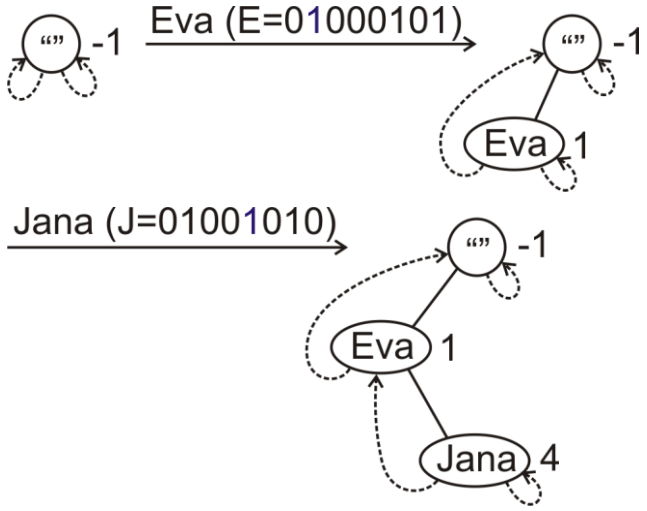

8 = 01000
0 -1
20 0
15 1
27 1
1 4
9 2
8 4

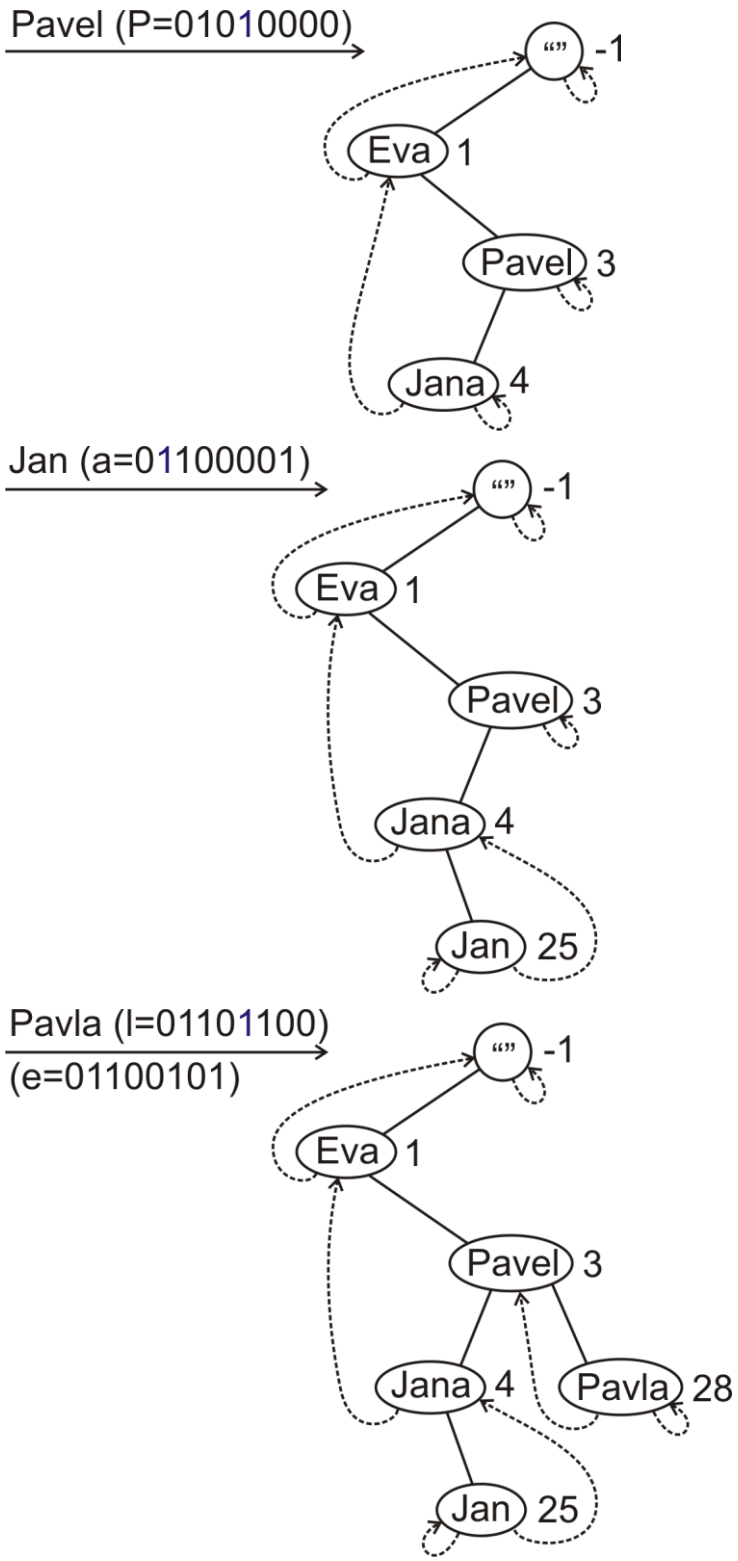

25 = 11001
0 -1
20 0
15 1
27 1
1 4
9 2
25 3
8 4


21 = 10101
0 -1
20 0
15 1
27 1
1 4
9 2
21 4
25 3
8 4

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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	End 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	Substitution	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	□

Příklad. Do stromu budeme ukládat jména (řetězce). Binární reprezentaci písmen jmen stanovíme dle ASCII tabulky. Pokud potřebujeme další znaky za koncem jména (řetězce), jejich binární hodnota bude rovna nule.





Pseudokód vyhledání:

Funkce *Search* vrací prvek, který je v uzlu, ve kterém skončilo vyhledávání. Ten je po návratu z funkce následně zapotřebí srovnat s hledaným prvkem.

```
Search(T, x)
  u ← T.head.left
  if u = T.head
    return nullItem
```

```
  d ← -1
  while u.bit > d
    d ← u.bit
    if Bit(x,d) = 0
      u ← u.left
    else
      u ← u.right
  return u.item
```

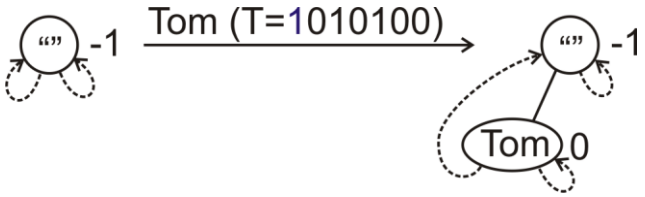
Pseudokód přidání:

```
NewNode(x, d)
  u ← new Node
  u.item ← x
  u.bit ← d
  return u

Insert(T, x)
  if T.head.left = T.head
    d ← 0
    while Bit(x,d) = 0
      d ← d+1
    v ← NewNode(x,d)
    v.left ← T.head
    v.right ← v
    T.head.left ← v
    return true
  y ← Search(T,x)
  if x = y
    return false
  d ← 0
  while Bit(x,d) = Bit(y,d)
    d ← d+1
  T.head.left ← InsertN(T.head.left,x,d,T.head)
```

```
InsertN(u, x, d, p)          // p je předchozí uzel před uzlem u
  if u.bit>d or u.bit≤p.bit
    v ← NewNode(x,d)
    if Bit(x,d) = 0
      v.left ← v
      v.right ← u
    else
      v.left ← u
      v.right ← v
    return v
  if Bit(x,u.bit) = 0
    u.left ← InsertN(u.left,x,d,u)
  else
    u.right ← InsertN(u.right,x,d,u)
  return u
```

Příklad. Do stromu budeme ukládat jména, která mají tři písmena a jsou bez diakritiky. Jejich binární reprezentaci stanovíme dle ASCII tabulky, použijeme jen 7 nižších bitů z kódování znaku.



x		*u*	*u.bit*
Iva	1001001 _ _	Tom	0
		Tom	0

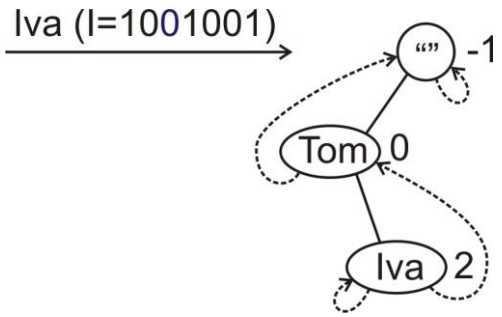
Iva	1001001 _ _	Tom	1010100 _ _

x		*d*	*u*	*u.bit*	*p*	*p.bit*	*u.bit>d* || *u.bit≤p.bit*
Iva	1001001 _ _	2	Tom	0	head	-1	false
			Tom	0	Tom	0	true

Iva	1001001 _ _	Iva.left	Iva.right

		Iva	Tom

```
head.left ← Tom
Tom.right ← Iva
```



---

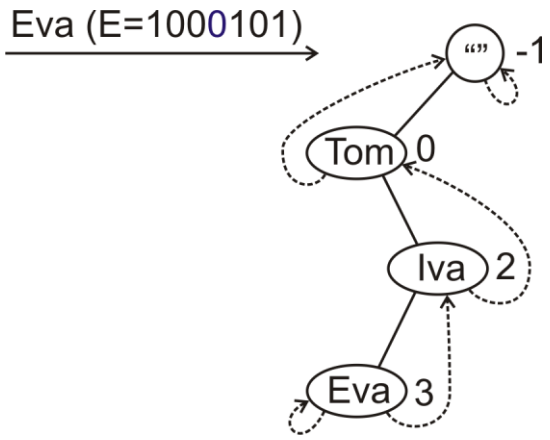
x		*u*	*u.bit*
Eva	1000101 _ _	Tom	0
	1000101 _ _	Iva	2
		Iva	0

Eva	1000101 _ _	Tom	1001001 _ _

x		*d*	*u*	*u.bit*	*p*	*p.bit*	*u.bit*>*d* || *u.bit*≤*p.bit*
Eva	1000101 _ _	3	Tom	0	head	-1	false
	1000101 _ _	3	Iva	2	Tom	0	false
			Iva	2	Iva	2	true

Eva	1000101 _ _	Eva.left	Eva.right
		Eva	Iva

```
Iva.left ← Eva
Tom.right ← Iva
head.left ← Tom
```



---

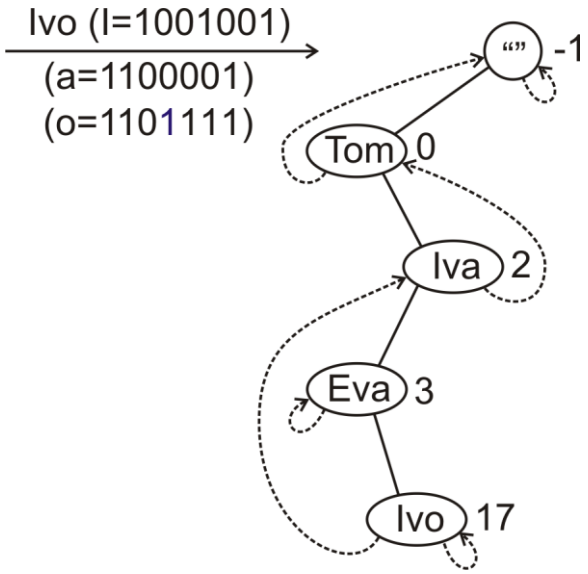
x		*u*	*u.bit*
Ivo	1001001 _ _	Tom	0
	1001001 _ _	Iva	2
	1001001 _ _	Eva	3
		Iva	2

Ivo	_ _ 1101111	Iva	_ _ 1100001

x		*d*	*u*	*u.bit*	*p*	*p.bit*	*u.bit*>*d* || *u.bit*≤*p.bit*
Ivo	1001001 _ _	17	Tom	0	head	-1	false
	1001001 _ _	17	Iva	2	Tom	0	false
	1001001 _ _	17	Eva	3	Iva	2	false
			Iva	2	Eva	3	true

Ivo	_ _ 1101111	Ivo.left	Ivo.right
		Iva	Ivo

```
head.left ← Tom
Tom.right ← Iva
Iva.left ← Eva
Eva.right ← Ivo
```



Tom	1010100 1101111 1101101	54 6F 6D
Iva	1001001 1110110 1100001	49 76 61
Eva	1000101 1110110 1100001	45 76 61
Ivo	1001001 1110110 1101111	49 76 6F