S&M

Split and Merge Compression Algorithm

By

Abdullah Hashim

Compression: The Tree Structure

The Tree Structure

Data Tree (DT) links:

If a link has a value equal to -1, the link is a null link, leading to no node; the node does not exist.

Parent(i)

Parent link is equal to an integer (p) pointing towards the node v_p, i.e. the parent of v_i is the node v_p , ***Parent***(i) = p

Sibling link is equal to an integer (s) pointing towards the node v_s. i.e. the sibling of v_i is the node v_s, ***Sibling***(i) = s

Data(i)

node i

Sibling(i)

Data link is equal to a character 'α' It is a single character store containing the character 'α'. i.e. the data of node v_i is 'α'

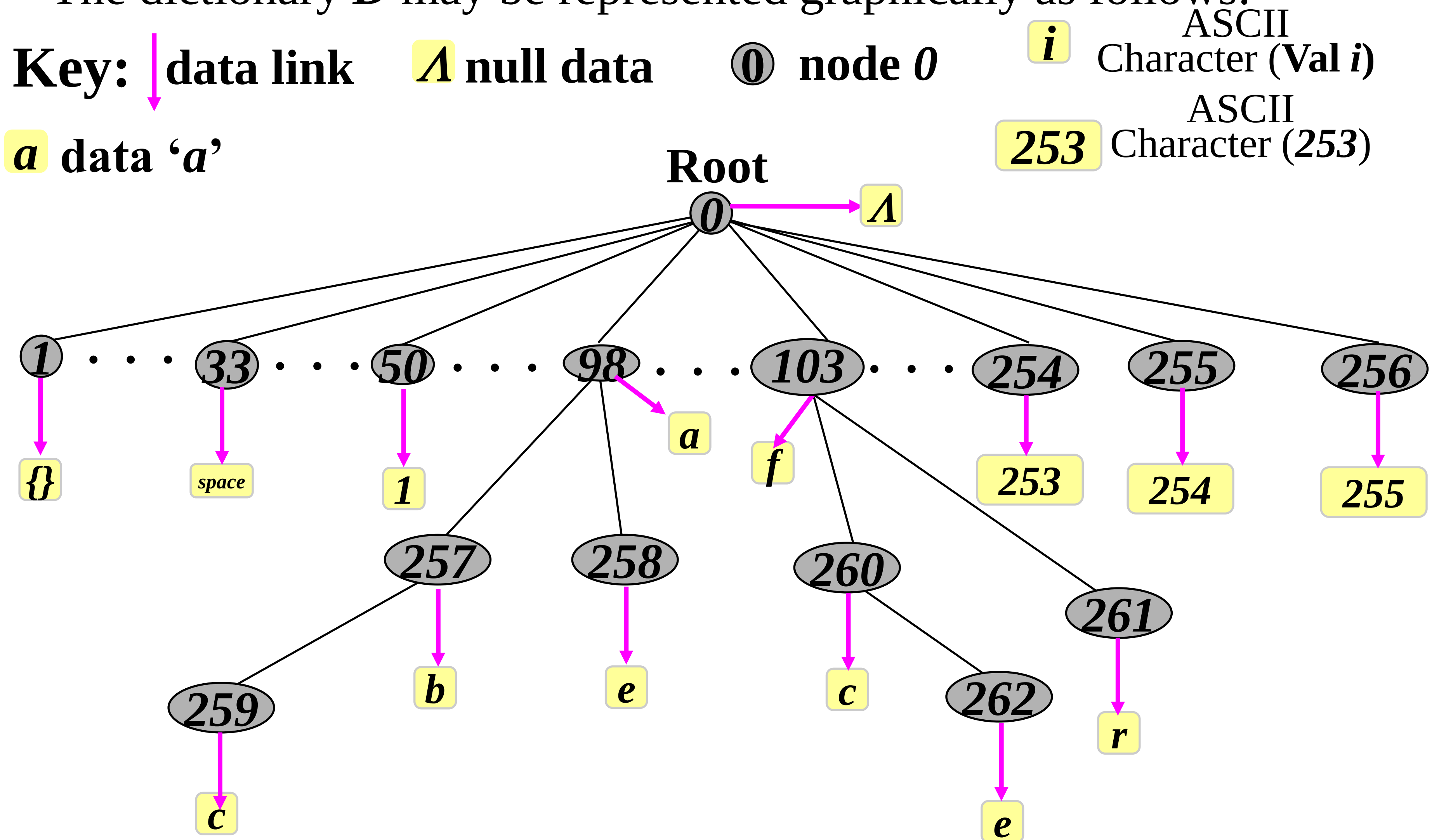
Child link is equal to an integer (c) pointing towards the node v_c. i.e. the child of v_i is the node v_c, ***Child***(i) = c

Child(i)

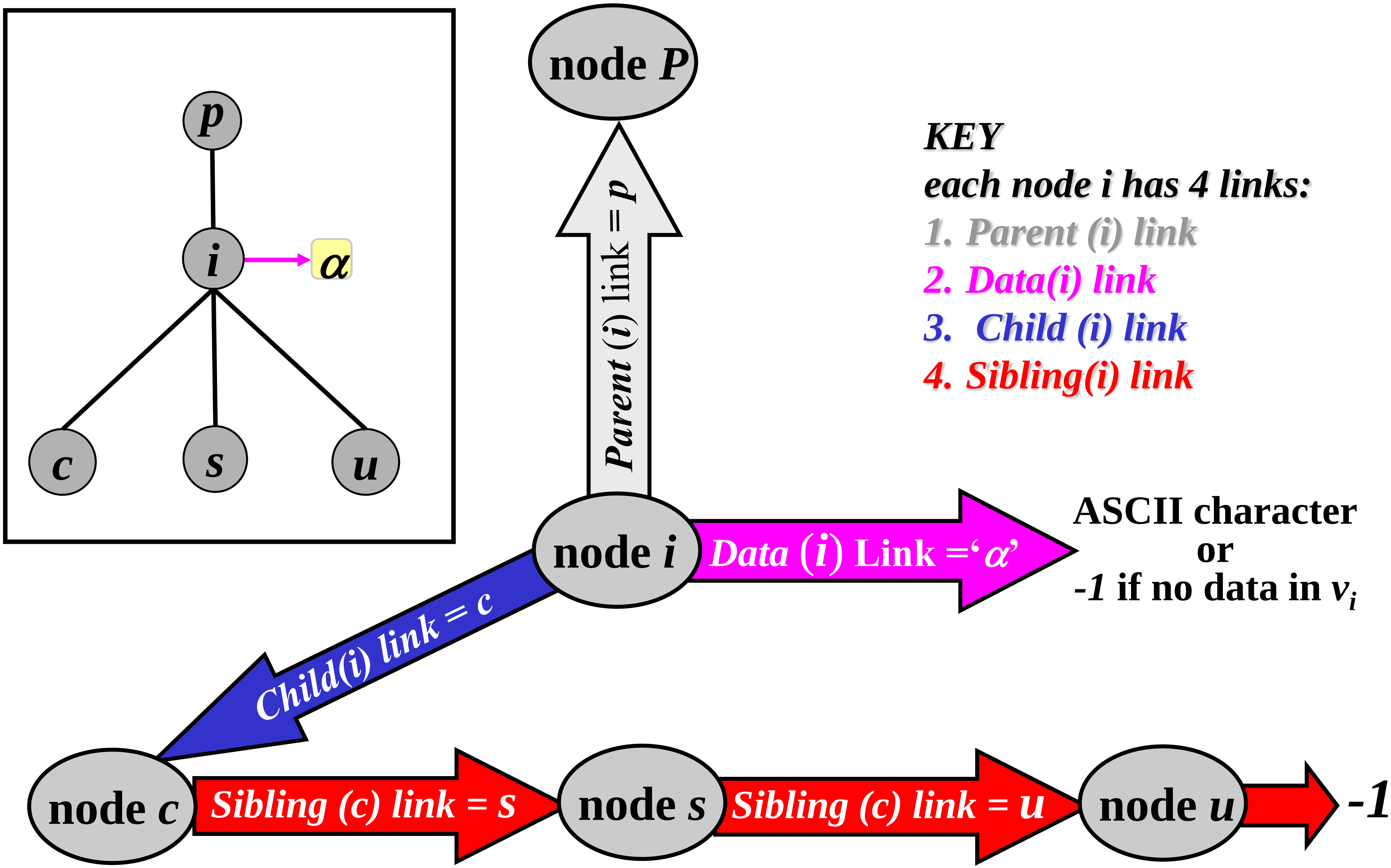
The Tree Structure

Let ***D*** = {*0, 1, 2,…, 253, 254, 255, ab, ae, abc, fc, fr, fce*}

The dictionary ***D*** may be represented graphically as follows:



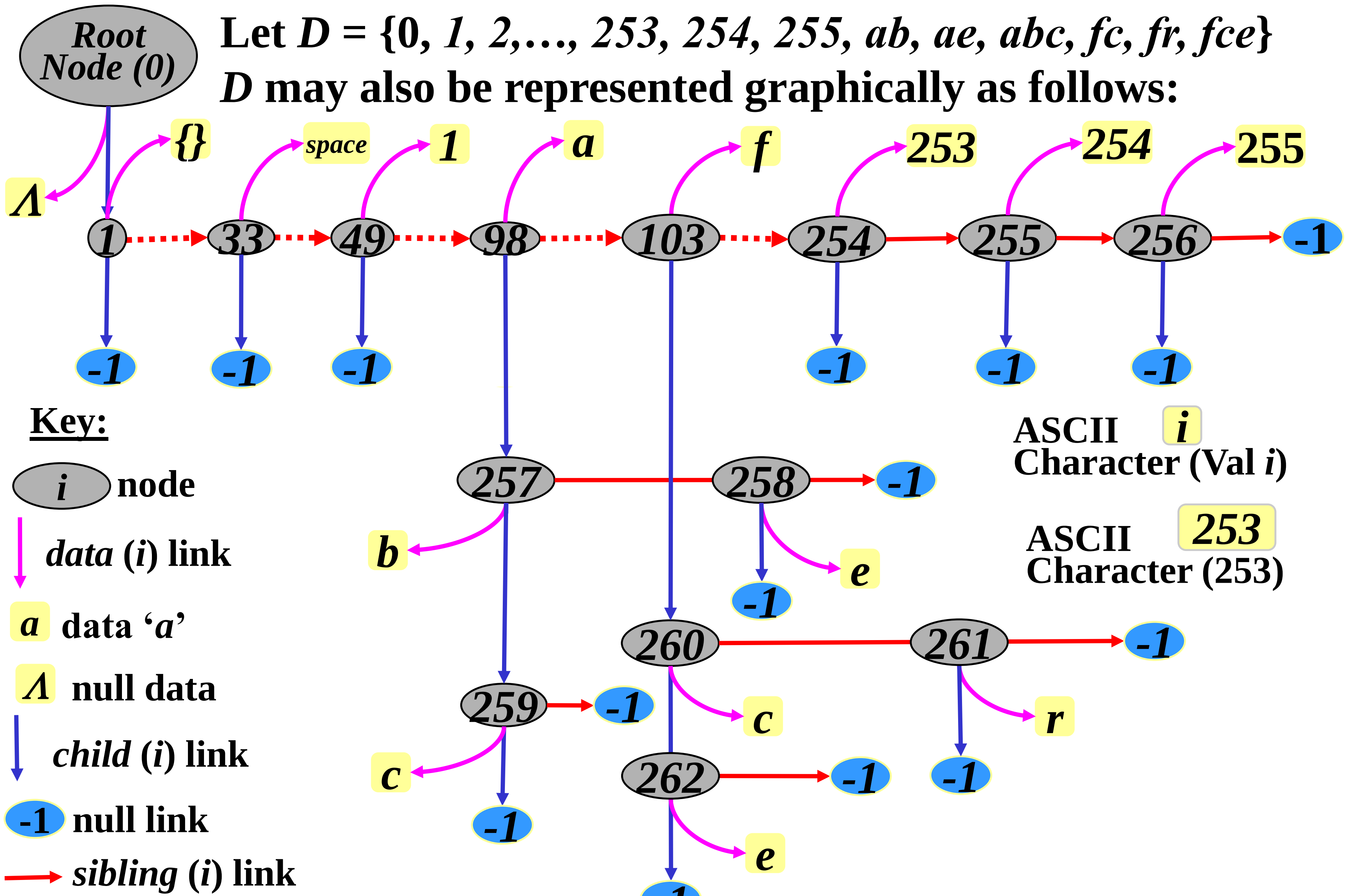
The Tree Structure



Tree Structure

Let *D* = {0, *1*, *2*,…, *253*, *254*, *255*, *ab*, *ae*, *abc*, *fc*, *fr*, *fce*}

D may also be represented graphically as follows:



The Tree Structure

ED**:** *Extended Dictionary*: ***ED*** is a dictionary containing:

1. All single ASCII characters, $\boldsymbol{A} = \{\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, \ldots, \boldsymbol{\alpha_N}\}$.
2. All multiple character words, $\boldsymbol{W} = \{\mathbf{w}_{\boldsymbol{1}}, \mathbf{w}_{\boldsymbol{2}}, \ldots\ \mathbf{w}_{\boldsymbol{M}}\}$.
3. All command "control" words, $\boldsymbol{C} = \{\boldsymbol{\delta_1}, \boldsymbol{\delta_2}, \ldots, \boldsymbol{\delta_C}\}$, usually $|\boldsymbol{C}|$ is in the range of two to three characters.
4. All the null words, $\boldsymbol{\{\}} = \{\boldsymbol{\Lambda_1}, \boldsymbol{\Lambda_2}, \ldots, \boldsymbol{\Lambda_i}, \ldots, \boldsymbol{\Lambda_B}\}$.

Therefore: total words in ***ED*** is $|\boldsymbol{ED}|_{\boldsymbol{max}} = \boldsymbol{N} + \boldsymbol{M} + \boldsymbol{C} + \boldsymbol{B}$

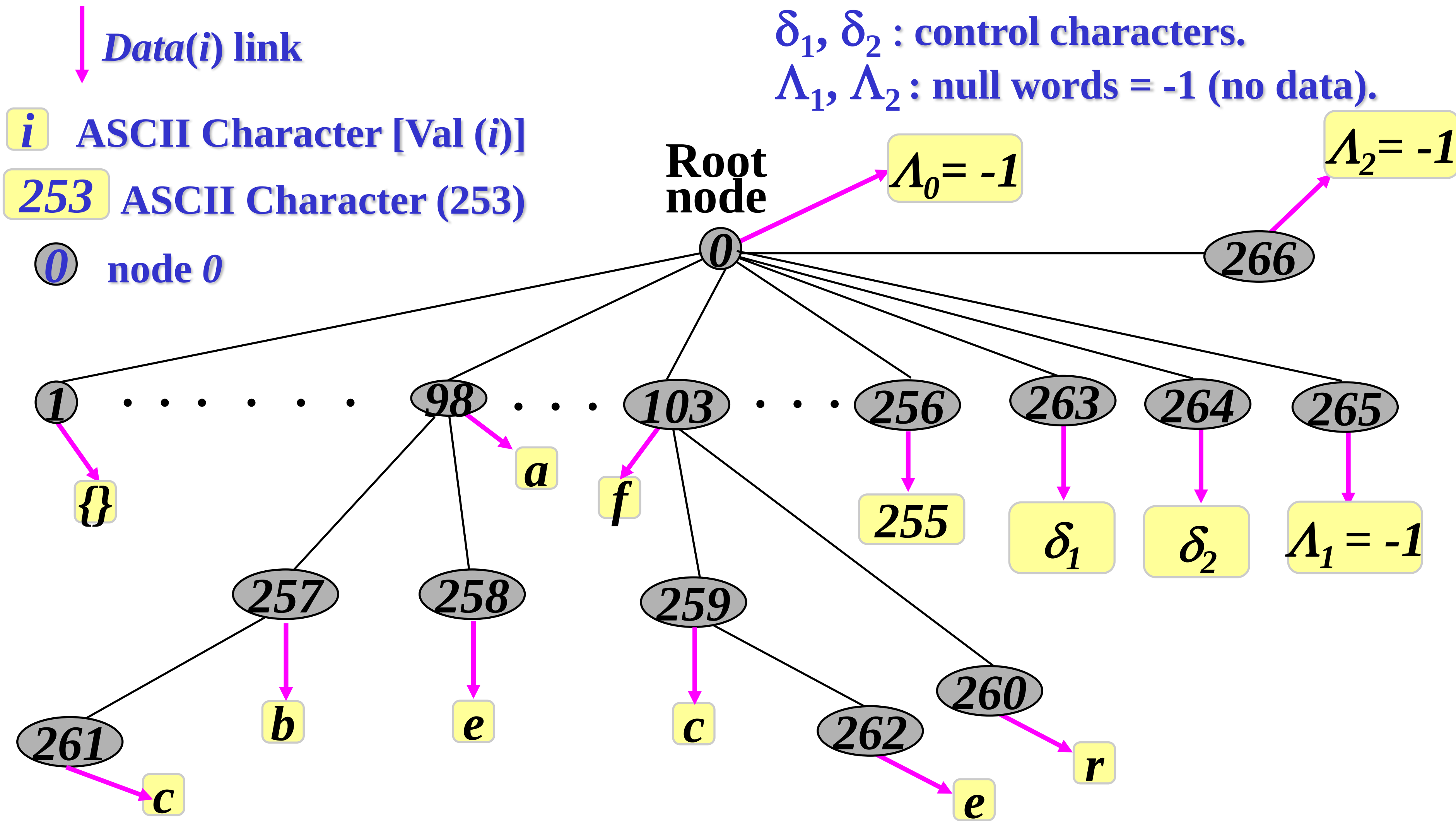
The Tree Structure

For example let the extended dictionary be given by the set *ED*:

$\{0, 1, 2, \ldots, 253, 254, 255, \delta_1, \delta_2, ab, ae, abc, fc, fr, fce, \Lambda_1, \Lambda_2\}$

δ_1, δ_2 : control characters.

Λ_1, Λ_2 : null words = -1 (no data).



The Tree Structure

OD *Ordered Dictionary*. The ordered dictionary is a list of words. The words are ordered according to some parameter or function; the leftmost word ($\boldsymbol{w_1}$) has the highest rank while the rightmost word ($\boldsymbol{w_M}$) has the lowest rank.

$\boldsymbol{OD} = < \boldsymbol{w_1, w_2, \ldots, w_j}, \ldots, \boldsymbol{w_{(M-1)}, w_M} >$

ODT *Ordered Dictionary Tree*. A graphical presentation of ***OD***. Let $\boldsymbol{ODT} = < v_1, v_2, \ldots, v_j, \ldots, v_{(M-1)}, v_M >$, where v_j is a node in ***ODT*** corresponding to the word $\boldsymbol{w_j}$ in the ***OD***.

To represent ***ODT*** graphically, two extra links will be needed. The first is left node link of node ***i***, the second is right node link of node ***i***. The links of node ***i*** leading to named nodes are:

1. ***Parent***(***i***) link; 2. ***Child***(***i***) link; 3. ***Sibling***(***i***) link;

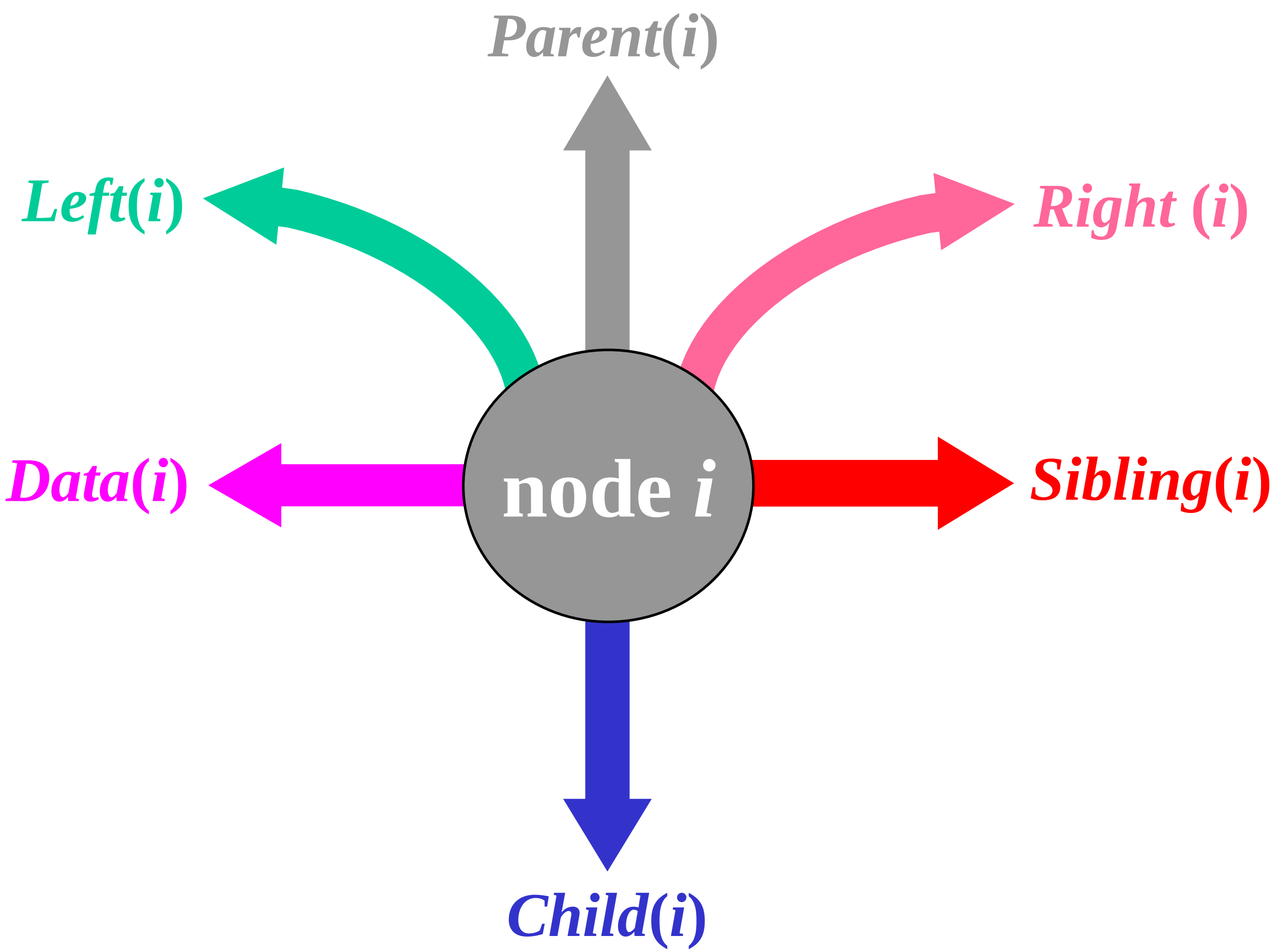
4. ***Data***(***i***) link; 5. ***Left*** (***i***) link and 6. ***Right(i)*** link.

The Tree Structure

***ODT* links**:

With left and right links

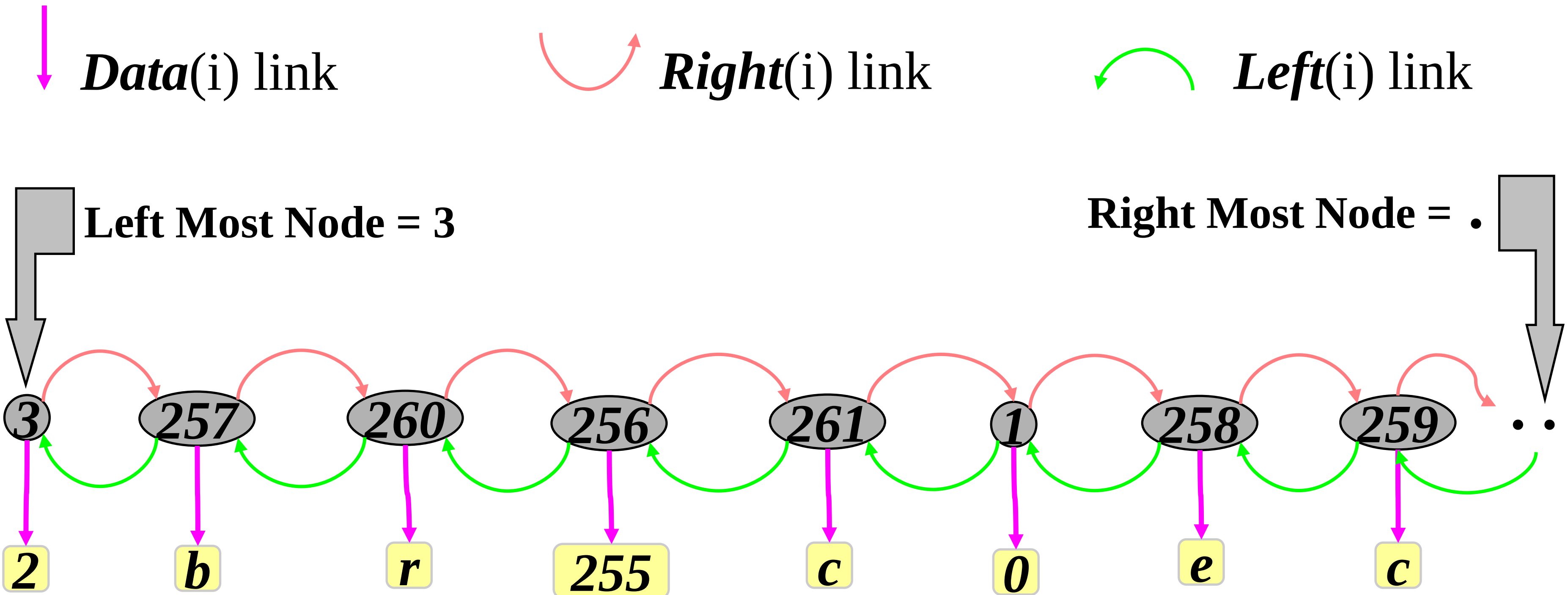
If a link has a value equal to -1, the link is a null link.



The Tree Structure

Graphical Representation of *ODT*

Let ***OD*** = <***3 , 257, 260, 256, 261, 1, 258, 259, ……***>

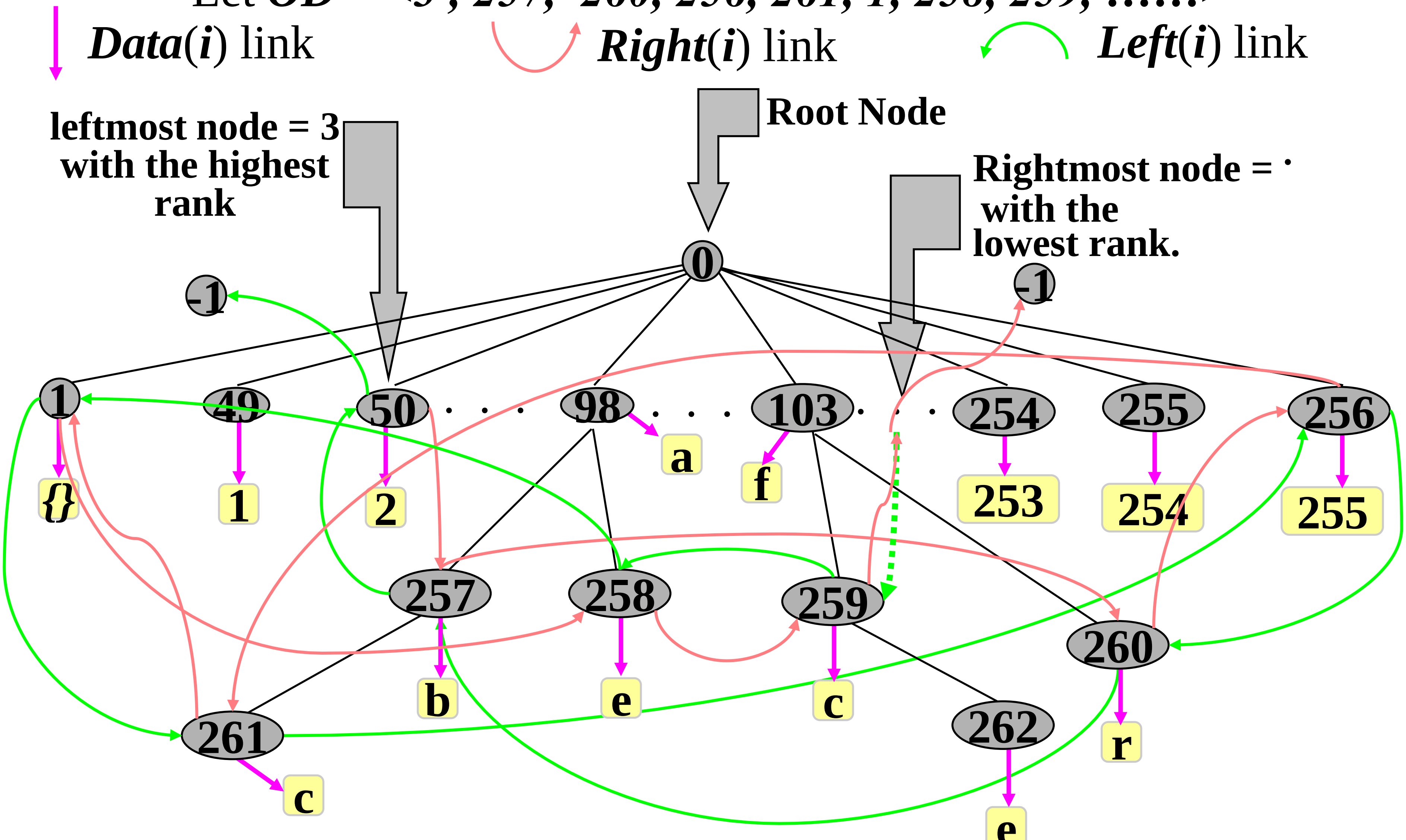


Node **3** has a higher rank than node **257**, node **257** has a higher rank than node **260, 260** has higher rank than **256**, ……etc.

The Tree Structure

Graphical Representation of *ODT*

Let ***OD*** = <*3 , 257, 260, 256, 261, 1, 258, 259, ……*>



Lempel & Ziv algorithm

Let string s_n be an input string from a file of characters in alphabet A over language Θ, at interval n. Match S_n to the longest word w_n in the dictionary given by:
$s_n = w_n = \langle e_1 e_2 \ldots e_i \ldots e_p \rangle$. If e_{p+1} is the unmatched character resulting from the matching process, then the new word added to the dictionary at interval n is $w_{new} = \langle e_1 e_2 \ldots e_p e_{p+1} \rangle$.

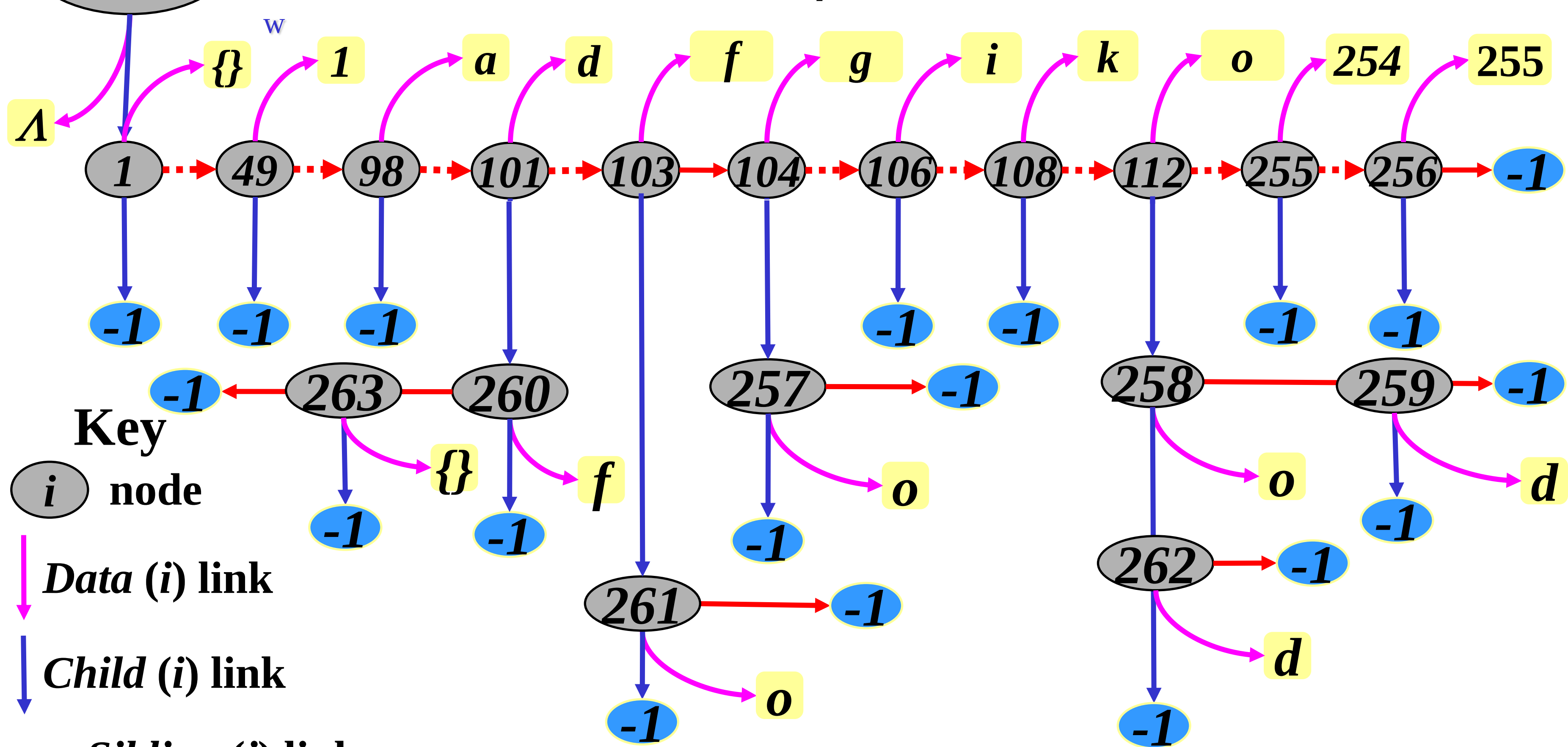
1. At interval n, an input string of source symbols is matched with the longest string in the dictionary (w_n).
2. The matched word w_n is coded by a binary codeword to form the compressed word [$w_{word}(w_n)$].
3. The input string $s_n = w_n$ is appended to the unmatched character resulting from the matching process in 2 above to form the new word (w_{new}).
4. The new word w_{new} is added to the dictionary.
5. The process is repeated starting from step one until EOF.

Lempel & Ziv algorithm

A practical simulation for input string <goodfood{}>

s_n	{}	g~~o~~	o~~o~~	o~~d~~	d~~f~~	f~~o~~	oo~~d~~	d~~{}~~	EOF
n	0	1	2	3	4	5	6	7	8
w_{new}	{}	go	oo	od	df	fo	ood	d{}	

$$R = \frac{72}{72} = 1$$



s_8 =‘{}’=1 = 0 0 0 0 0 0 0 0

w_{word8} =‘{}’=1 = 0 0 0 0 0 0 0 0 1

Lempel & Ziv algorithm

Lempel and Ziv in their original paper (On the Complexity of Finite sequences. IEEE Trans. IT-22.1 Jan 1976, pp75-81) have shown that if an initial dictionary of single character words of an alphabet is allowed to grow indefinitely by adding a new word at every successive interval in time and the dictionary words are coded by equal length codewords, asymptotically the compression ratio of such a coding scheme will tend to infinity as the dictionary size tends to infinity. Many practical implementations of this basic theory have been reported in the literature of different approaches and complexity. However, all suffer from the same basic inherent feature that once a practical limit is imposed on the dictionary size and its wordlength, the compression ratio will be degraded rapidly. In fact all practical implementations of the dictionary result in an average codeword length even higher than second order entropy. The low compression ratios are due to the practical implementations of the scheme which fill the dictionary mainly with two-character words of low frequency of occurrence instead of long words with high frequency.