객체지향프로그래밍과 자료구조

Ch 12. Text Processing, trie, Prefix Matching

정보통신공학과
교수 김 영 탁
(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ 텍스트(문자열) 처리 응용 프로그램
  - 예측구문 (predictive text)
  - Longest prefix matching
- ◆ trie 자료구조
- ◆ trie 자료구조의 C++ 구현
  - class TrieNode
  - class Trie_String
  - insert()
  - search()
  - deleteKeyStr()
  - eraseTrie()

텍스트 처리와 trie 자료구조

텍스트 처리 응용 분야

텍스트 처리 응용 분야	설명
예측 구문 (predictive text)제시	휴대 단말장치에서 신속한 문장 입력 완성을 할 수 있도록 초기 단계에서 입력된 문자를 기반으로 예측되는 단어 또는 문장을 제시하며, 제시된 단어/문장을 쉽게 선택함으로써 전체 문장을 신속하게 완성할 수 있게 함.
Longest Prefix Matching	인터넷에서 전송되는 패킷의 목적지 주소의 prefix (주소의 앞부분)와 가장 많이 일치하는 항목을 찾아 다음 패킷 교환기를 결정한 후, 패킷을 전송할 수 있게 함으로써, 가장 효율적인 패킷 전송이 가능하도록 함.
DNA 염기 서열 분석	염기서열 (nucleic acid sequence)을 A (adenine), C (cytosine), G (guanine), T (thymine)의 4 개 문자가 조합되는 문자열로 표시하며, 이를 기반으로 다양한 분석 알고리즘을 수행
무손실 텍스트 압축	문서를 저장하는 공간을 줄이고, 그 문서를 전송하기 위한 통신 시간을 단축시킬 수 있도록 문장을 압축. 문서에 포함된 각 문자들의 발생 빈도를 감안하여 문자 별로 서로 다른 길이의 표기 방법을 사용함으로써 압축.

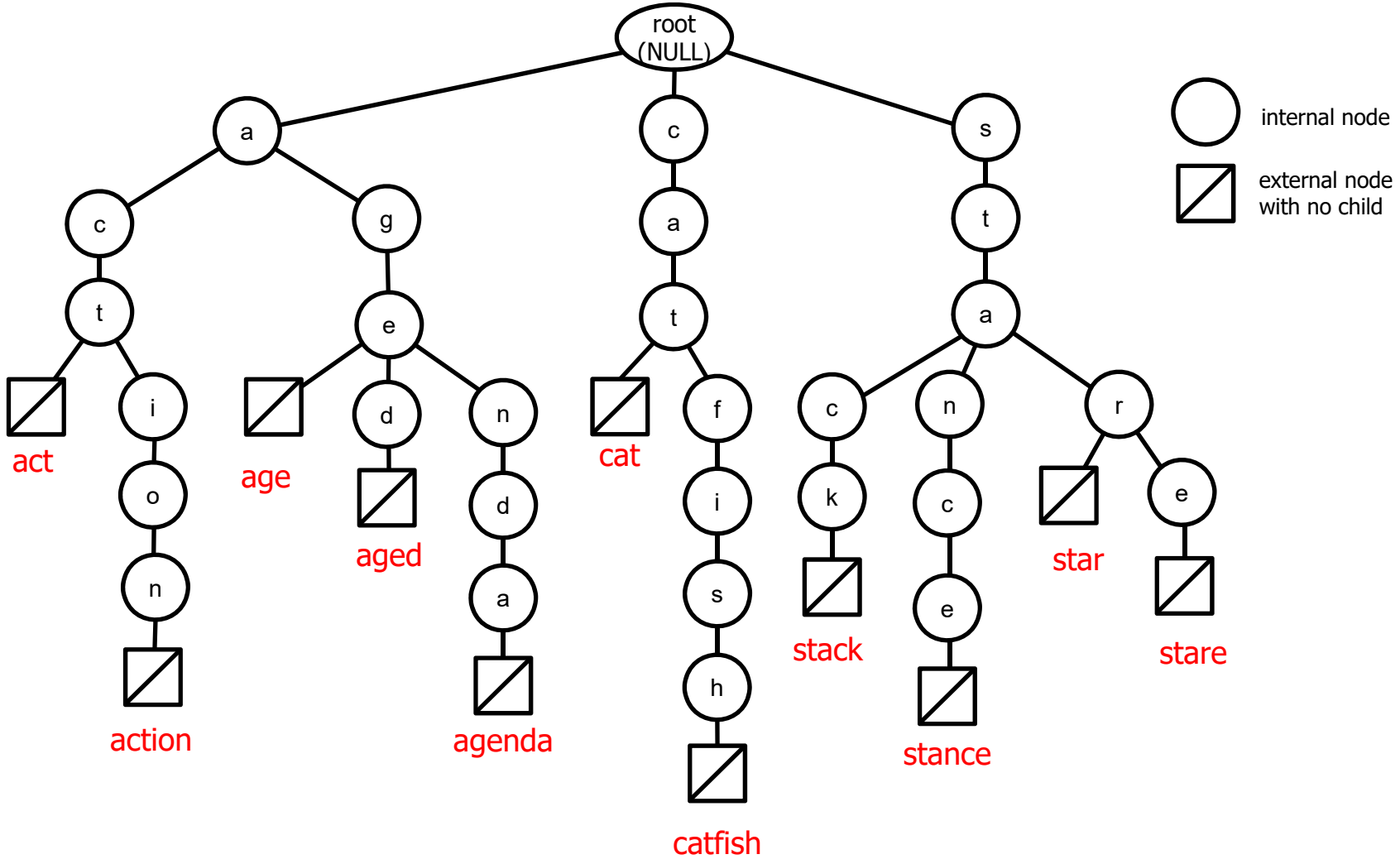
trie 자료구조란?

◆ trie

- trie는 retrieval 단어로 부터 유래하였으며, 탐색 트리로 사용되는 자료구조이다.
- 특히, 동적으로 변화하는 데이터 집합에서 탐색을 효율적으로 수행할 수 있는 자료구조이며, 주로 탐색 key가 텍스트(문자열, string)로 주어질 때 사용된다.
- 이진 탐색 트리 (binary search tree)와는 달리, 각 trie node는 key 값 (문자열) 중의 일부만을 가지며, trie 구조에서 root 노드로 부터 그 trie node에 도달 할 때 까지 경유한 모든 trie node의 key 값들이 결합되어 최종 key 값을 구성한다.
- root trie node는 null 문자 (즉, empty string)을 가진다. 따라서, 어떤 trie node의 자식 노드들 및 그 후손 노드 들은 모두 그 trie node까지 경로에서 구성된 key 문자열을 동일한 공통적인 접두어 (prefix)로 가지게 된다.
- trie는 동일한 접두어 (prefix) 또는 radix (어근)을 가지는 다양한 단어들을 keyword로 사용하는 응용 분야에 효과적으로 사용될 수 있다. (예: 예측 구문 (predictive text) 제시, 인터넷 패킷 경로 선정 (longest prefix matching))

trie 자료구조란? (2)

◆ trie 구성 예



trie 자료구조의 특성

◆ trie와 hash table의 비교

- hash table에서는 서로 다른 key 문자열에 대하여 동일한 hash 값이 생성되는 "충돌"이 발생할 수 있으나, trie에서는 key 값이 서로 다를 경우 충돌이 발생하지 않는다.
- hash table에서는 key 값의 순서에 관계없는 hash value가 생성되어 저장되나, trie에서는 key 값의 순서에 따라 정렬시켜 저장할 수 있다.
- trie에서는 key string의 각 문자 (character) 마다 trie node를 구성하여야 하므로, hash table 보다 메모리를 더 많이 사용할 수 있다. 이 단점을 해결하는 compressed trie 구조가 있다.

◆ trie 자료 구조 기반의 검색 성능

- key string의 최대 길이 (문자 개수) 만큼 비교: O(len(keyStr))

trie 자료구조의 주요 응용 분야

◆ 예측 구문 (predictive text)

- 스마트 기기의 입력에서 적은 수의 타이핑으로도 예측되는 구문 (predictive text)을 안내하여 신속하게 구문을 완성할 수 있게 함
- 사전 (dictionary)을 활용한 단어 검색에서 자동 완성 (auto complete) 기능으로 예상되는 단어를 열거하여 주고, 이 단어 들 중에서 고르게 함

◆ Longest prefix matching

- 인터넷 패킷 교환기 (router)의 packet forwarding table 구성에서 목적지 주소 (destination address)와 가장 많이 일치하는 항목을 선정하여 전달 할 수 있도록 라우팅 테이블을 구성

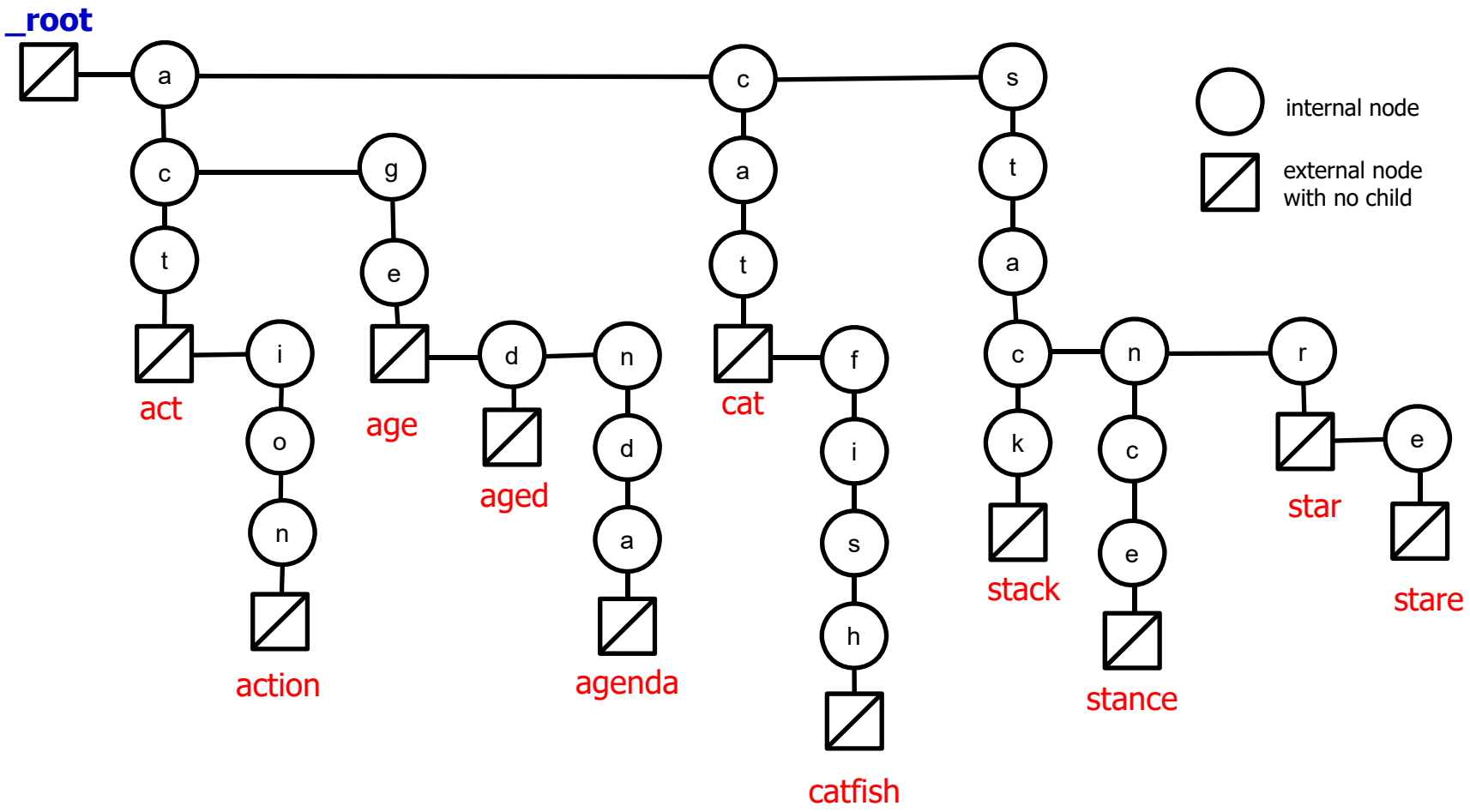
trie 자료구조의 구현

◆ trie 자료 구조의 구현에서의 고려사항

- 이진 탐색 트리와 달리 하나의 트리 노드에 접속되는 자식 노드의 수가 3개 이상 포함될 수 있음
- 동일한 substring을 가지는 다수의 key string이 존재할 수 있으며, 어떤 key string의 prefix가 다른 key string이 될 수 도 있음
  - 예) key string "age", "aged", "agenda"
- root node로 부터 trie tree 탐색에서 longest matching이 가능하여 predictive text가 제공 될 수 있도록 구성하여야 함

trie 자료구조의 구현

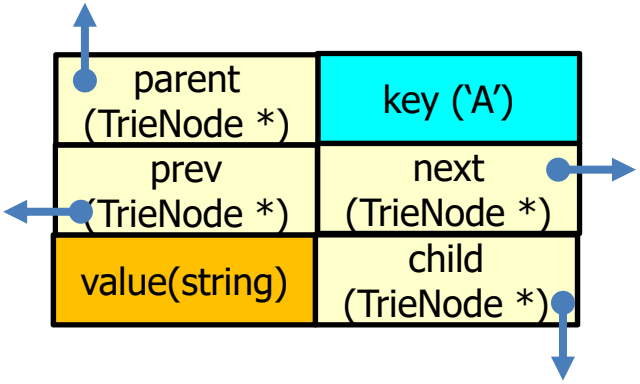
◆ trie 구현 예



trie의 C++ 프로그램 모듈 구현 (1) – Class TrieNode

◆ Class TrieNode

- private data members
  - char key; // key character of this trie_node
  - E value; // value assigned to the key string that ends at this trie_node
  - 4 pointers to previous, next, parent, and child trie_nodes



class TrieNode

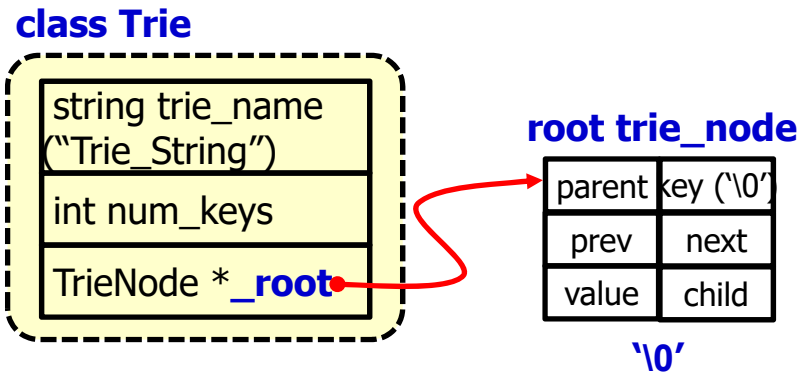
```
/* TrieNode.h  (1) */

#ifndef TRIE_NODE_H
#define TRIE_NODE_H
#include <iostream>
#include <string>
#include <list>
#define VALUE_INTERNAL_NODE ""
using namespace std;

typedef list<string> STL_list;

template <typename E>
class TrieNode
{
public:
    TrieNode() {} // default constructor
    TrieNode(char k, E v) : key(k), value(v)
        { prev = next = parent = child = NULL; }
    void setKey(char k) { key = k; }
    void setValue(E v) { value = v; }
    void setNext(TrieNode<E> *nxt) { next = nxt; }
    void setPrev(TrieNode<E> *pv) { prev = pv; }
    void setParent(TrieNode<E> *pr) { parent = pr; }
    void setChild(TrieNode<E> *chld) { child = chld; }
```

```
/* TrieNode.h  (2) */

    char getKey() { return key; }
    E getValue() { return value; }
    TrieNode<E> *getPrev() { return prev; }
    TrieNode<E> *getNext() { return next; }
    TrieNode<E> *getParent() { return parent; }
    TrieNode<E> *getChild() { return child; }
    void _fprint(ostream& fout,
        TrieNode<E> *pTN, int indent);
private:
    char key;
    E value;
    TrieNode<E> *prev;
    TrieNode<E> *next;
    TrieNode<E> *parent;
    TrieNode<E> *child;
};
```

Class Trie

Class Trie

- data members
  - TrieNode<E> *_root;
  - int num_keys;
  - string trie_name;
- member functions
  - Trie(string name); // constructor
  - void insert(char *keyWord, E value);
  - void insertExternalTN(TrieNode<E> *pTN, char *keyWord, E value);
  - TrieNode<E> *findKeyWord (char *keyWord);
  - void findPrefixMatch(char * prefix, STL_list& predictWords);
  - void deleteKeyWord(char *keyWord);
  - void eraseTrie();
  - void fprintTrie(ostream& fout);
  - TrieNode<E> *_find(char *keyWord, SearchMode sm=FULL_MATCH);
  - void _traverse(TrieNode<E> *pTN, STL_list& list_keywords);

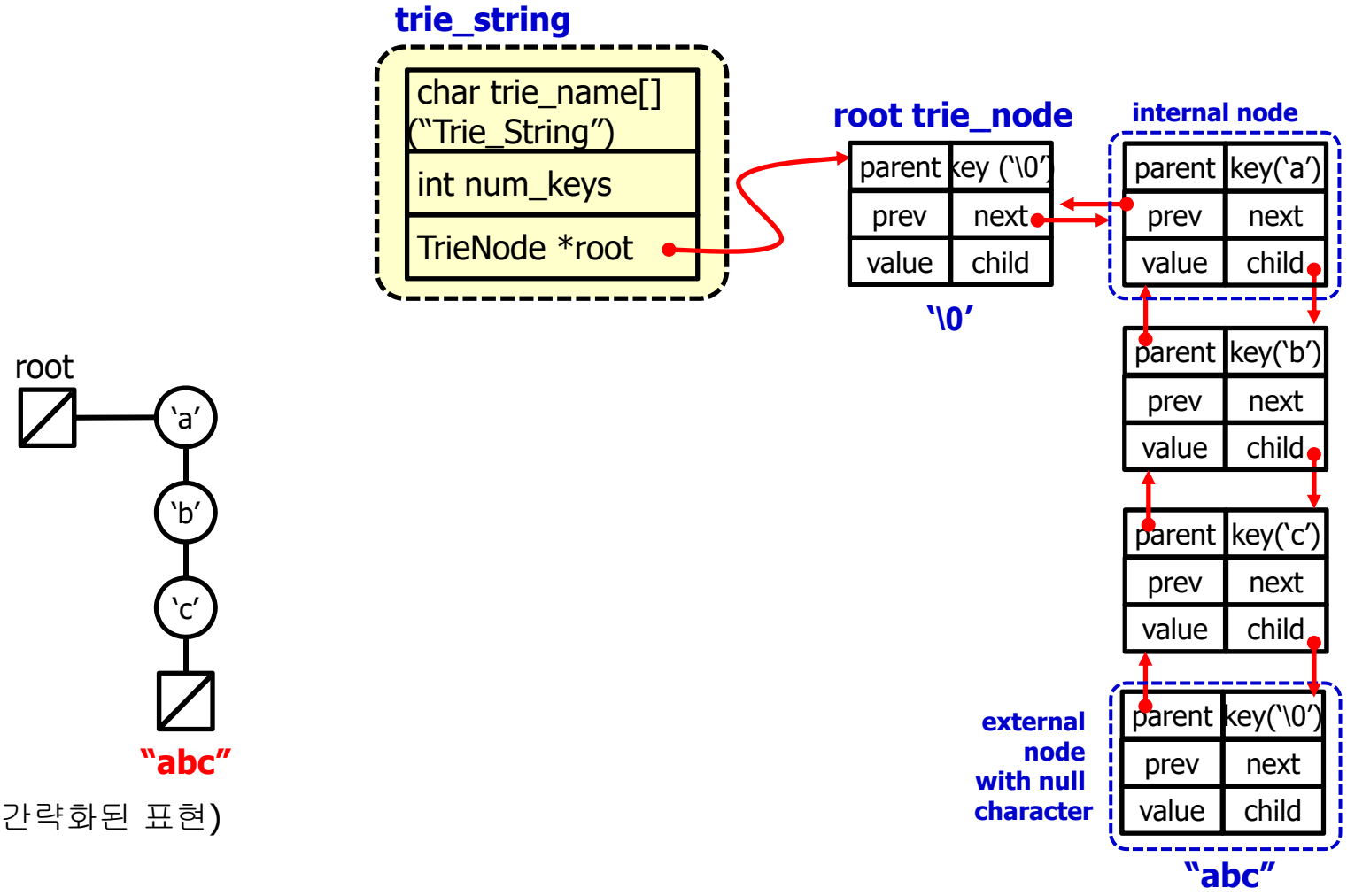


```
/* Trie.h  (1) */
#ifndef Trie_H
#define Trie_H
#include <iostream>
#include <string>
#include "TrieNode.h"
#define MAX_STR_LEN 50

//#define DEBUG
using namespace std;

typedef list<string> STL_list;
typedef list<string>::iterator List_String_Iter;
enum SearchMode {FULL_MATCH, PREFIX_MATCH};
template <typename E>
class Trie
{
public:
    Trie(string name); // constructor
    void insert(const char *keyWord, E value);
    void insertExternalTN(TrieNode<E> *pTN, const
        char *keyWord, E value);
    TrieNode<E> *findKeyWord(const char *keyWord);
    void findPrefixMatch(const char * prefix,
        STL_list& predictWords);
    void deleteKeyWord(const char *keyWord);
    void eraseTrie();
    void fprintTrie(ostream& fout);
```

```
/* Trie.h  (2) */
protected:
    TrieNode<E> *_find(const char *keyWord,
        SearchMode sm=FULL_MATCH);
    void _traverse(TrieNode<E> *pTN, STL_list&
        list_keywords);
private:
    TrieNode<E> *_root;  // _root trie node
    int num_keys;
    string trie_name;
};

template<typename E>
Trie<E>::Trie(string name)
{
    trie_name = name;
    _root = new TrieNode<E>('\0', "");
    _root->setKey('\0');
    _root->setPrev(NULL);
    _root->setNext(NULL);
    _root->setParent(NULL);
    _root->setChild(NULL);
    num_keys = 0;
}
```

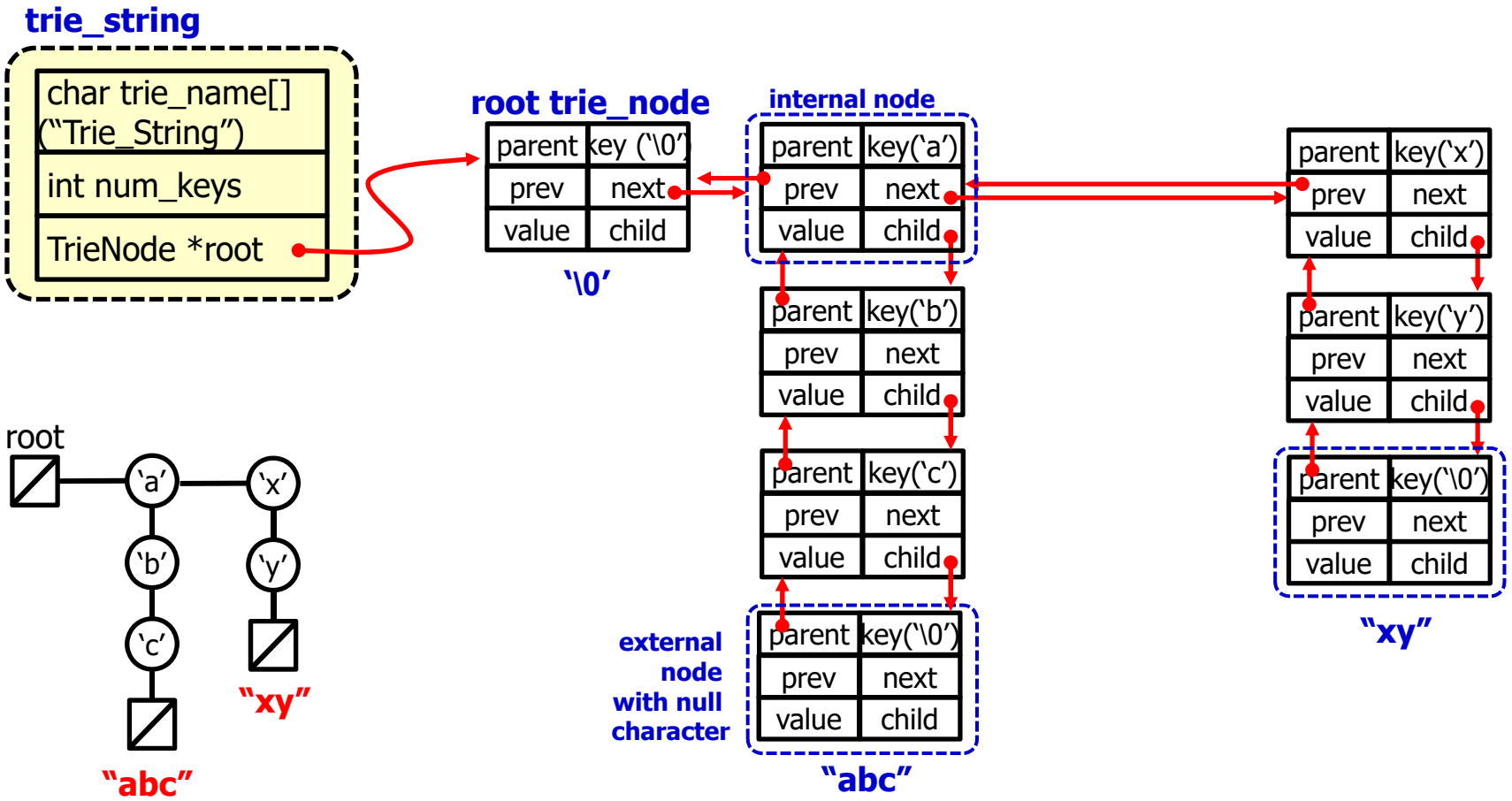
trie 자료구조에서의 연산 (1)
- insert()

◆ insert ("abc") insert operation



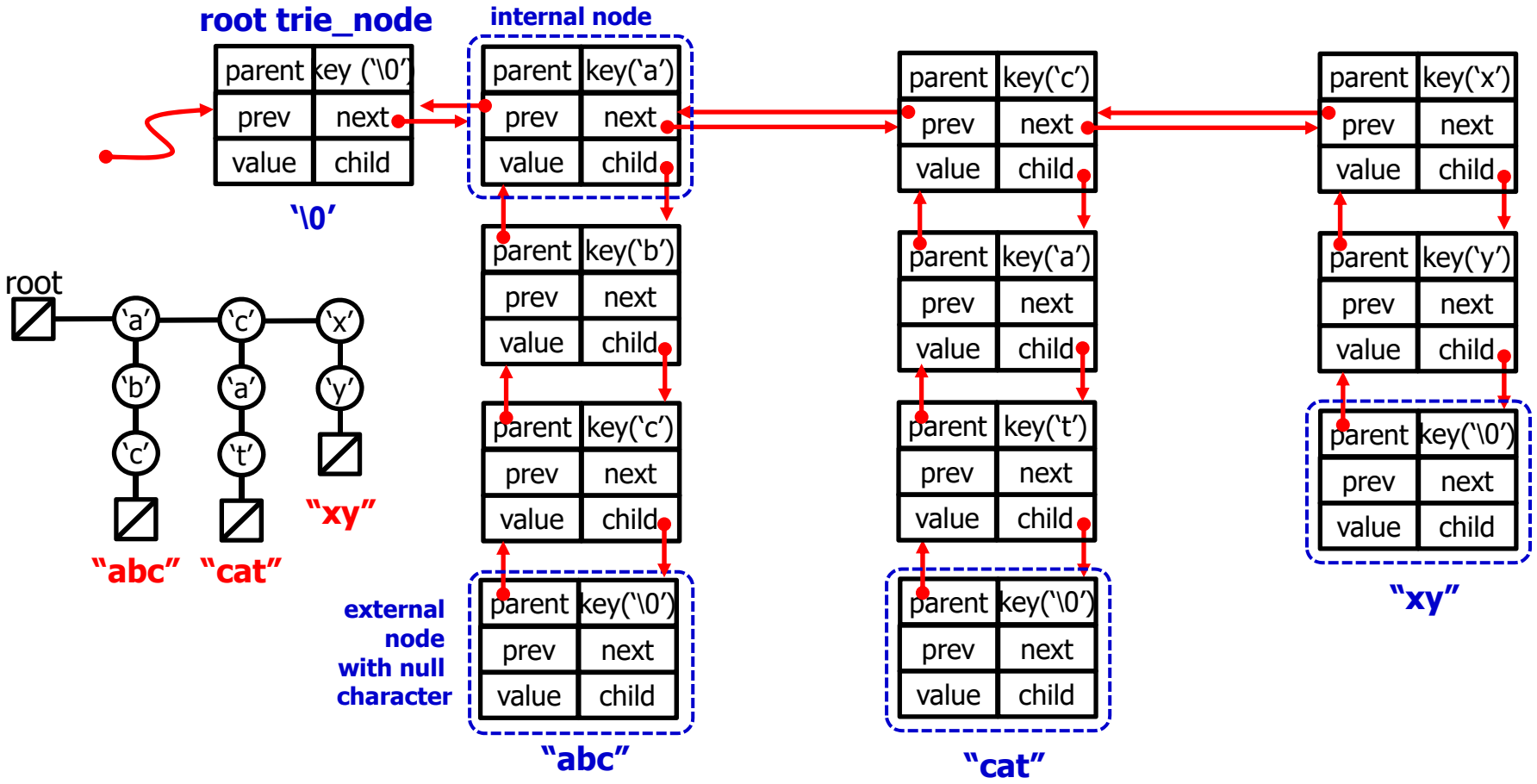
insert operation

- insert ("xy") while "abc" is already inserted before



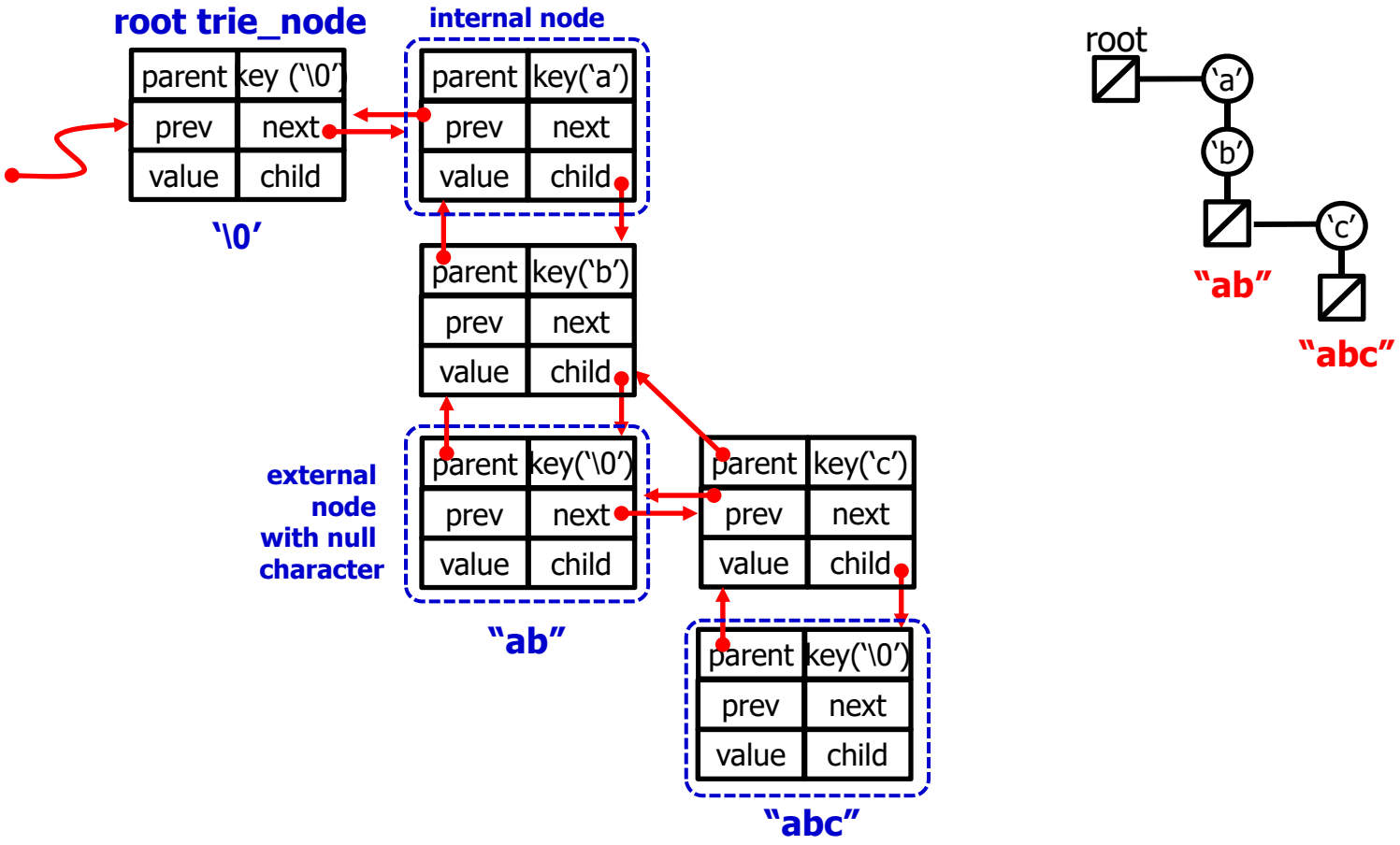
insert operation

- insert ("cat") while "abc" and "xy" are already inserted before



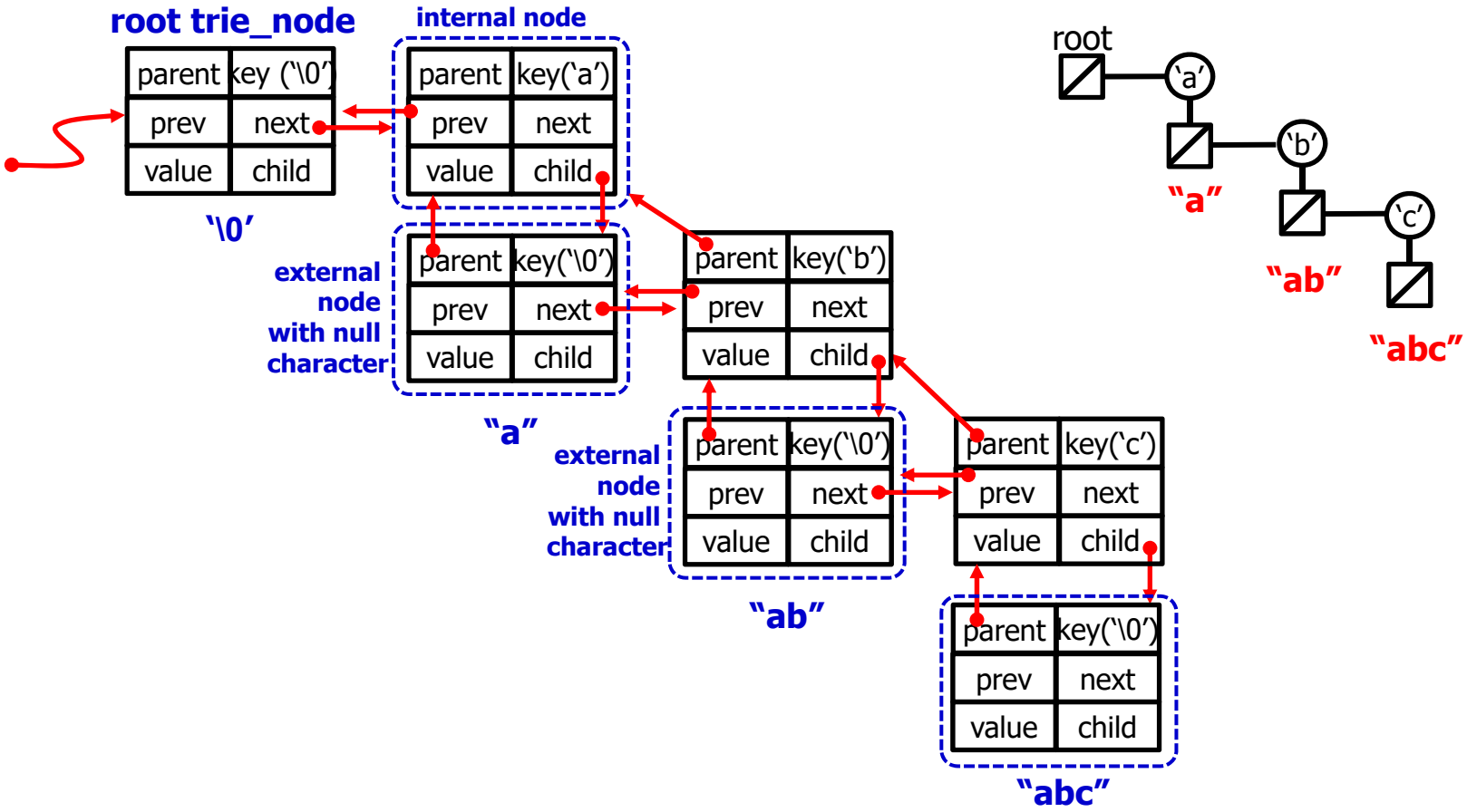
insert operation

- insert ("ab") while "abc" was already inserted before



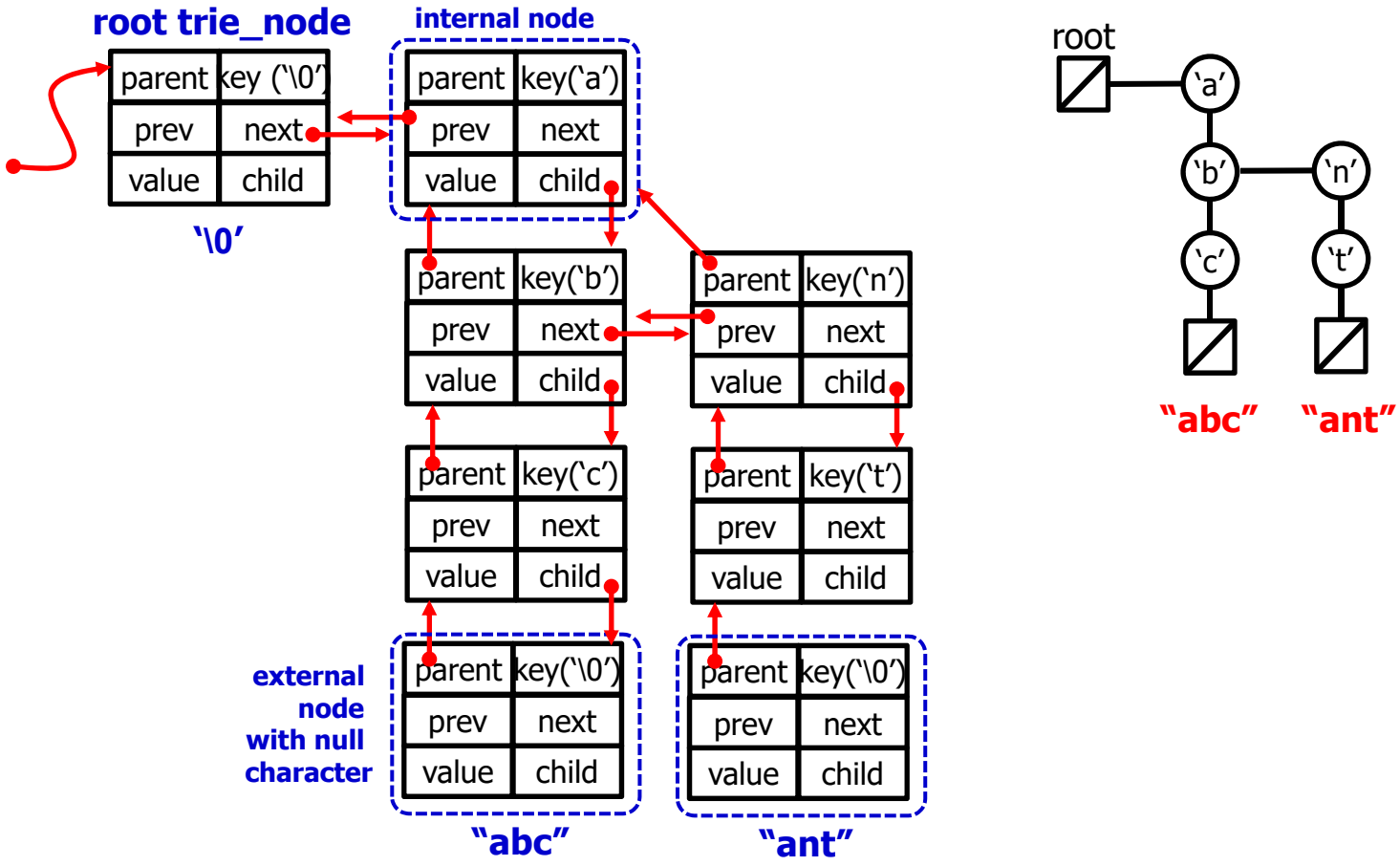
insert operation

- insert ("a") while "abc" and "ab" were already inserted before



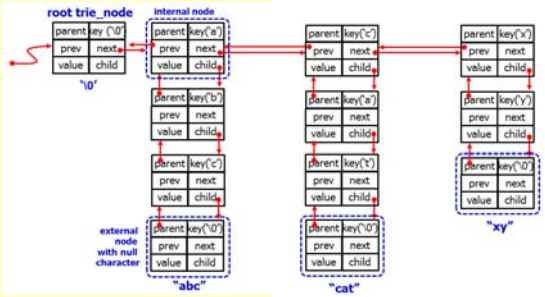
insert operation

- insert "ant" while "abc" is already existing

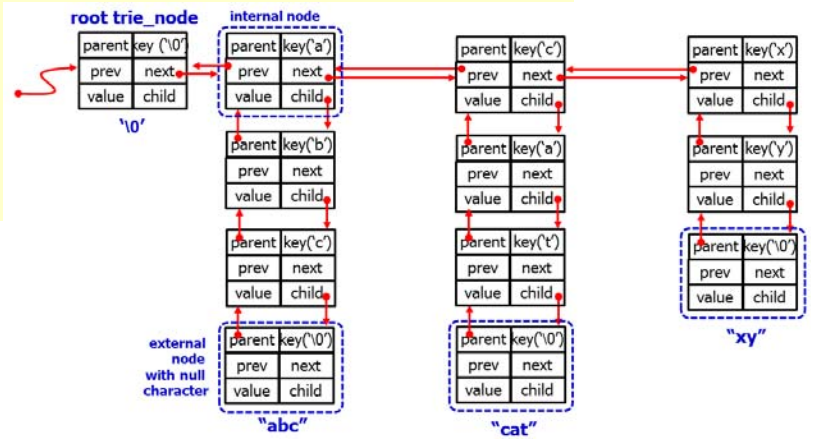


```
/* Trie.h  (3) */

template<typename E>
void Trie<E>::insertExternalTN(TrieNode<E> *pTN, const char *keyWord, E value)
{
    TrieNode<E> *pTN_New = NULL;

    pTN_New = new TrieNode<E>('\0', value);
    pTN->setChild(pTN_New);
    (pTN->getChild())->setParent(pTN);
    pTN_New->setValue(value);
    //cout << "key (" << keyWord << ") is inserted \n";
}

template<typename E>
void Trie<E>::insert(const char *keyWord, E value)
{
    TrieNode<E> *pTN = NULL, *pTN_New = NULL;
    char keyWording[MAX_STR_LEN];
    char *keyPtr = keyWord;

    if (keyWord == NULL)
       return;

    /* Firstly, check any possible duplicated key insertion  */
    if (_find(keyWord, FULL_MATCH) != NULL)
    {
       cout << "The given key is already existing; just return !!" << endl;
       return;
    }
```

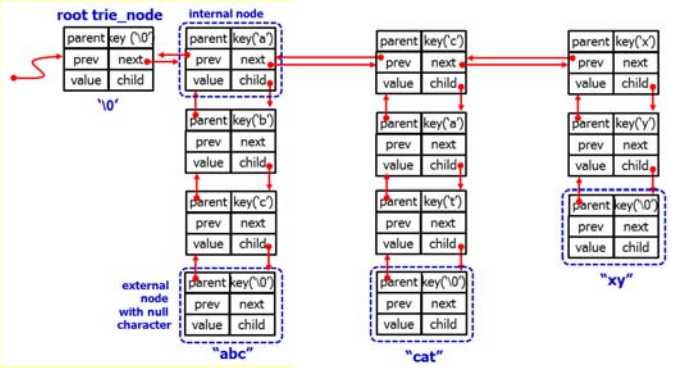
```
/* Trie.h  (4) */

    pTN = this->_root;
    while ((pTN != NULL) && (*keyPtr != '\0'))
    {
        if ((pTN->getKey() < *keyPtr) && (pTN->getNext() == NULL) && (*keyPtr != '\0'))
            break;
        while ((pTN->getKey() < *keyPtr) && (pTN->getNext() != NULL))
            pTN = pTN->getNext();
        while ((pTN != NULL) && (pTN->getKey() == *keyPtr) && (*keyPtr != '\0'))
        {
            pTN = pTN->getChild();
            keyPtr++;
        }
        if ((pTN->getKey() > *keyPtr) && (*keyPtr != '\0'))
            break;
    } // end while for positioning

    /* Secondly, the given key string is a sub-string of an existing key */
    /* e.g.) trying to insert "abc" while "abcde" is already exisiting. */
    if ((pTN->getKey() != '\0') && (*keyPtr == '\0'))
    {
        /* there was a longer key string already !! */
        /* break the longer key string, and connected to the separated key strings */
        pTN_New = new TrieNode<E>('\0', value);
        pTN_New->setParent(pTN->getParent());
        (pTN->getParent())->setChild(pTN_New);
        pTN_New->setNext(pTN);
        pTN->setPrev(pTN_New);
        //cout << "key (" << keyWord << ") is inserted" << endl;
        this->num_keys++;
        return;
    }
```



```
/* Trie.h  (5) */

    else if ((pTN->getKey() < *keyPtr) && (pTN->getNext() == NULL) && (*keyPtr != '\0'))
    {
        /* at this level, a new substring is inserted as the last nodes */
        pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
        pTN_New->setParent(pTN->getParent());
        pTN_New->setPrev(pTN);
        pTN->setNext(pTN_New);
        pTN = pTN_New;
        keyPtr++;
        while (*keyPtr != '\0')
        {
            pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
            pTN->setChild(pTN_New);
            (pTN->getChild())->setParent(pTN);
            pTN = pTN->getChild();
            keyPtr++;
        }
        if (*keyPtr == '\0')
        {
            insertExternalTN(pTN, keyWord, value);
            this->num_keys++;
            return;
        }
    }
```



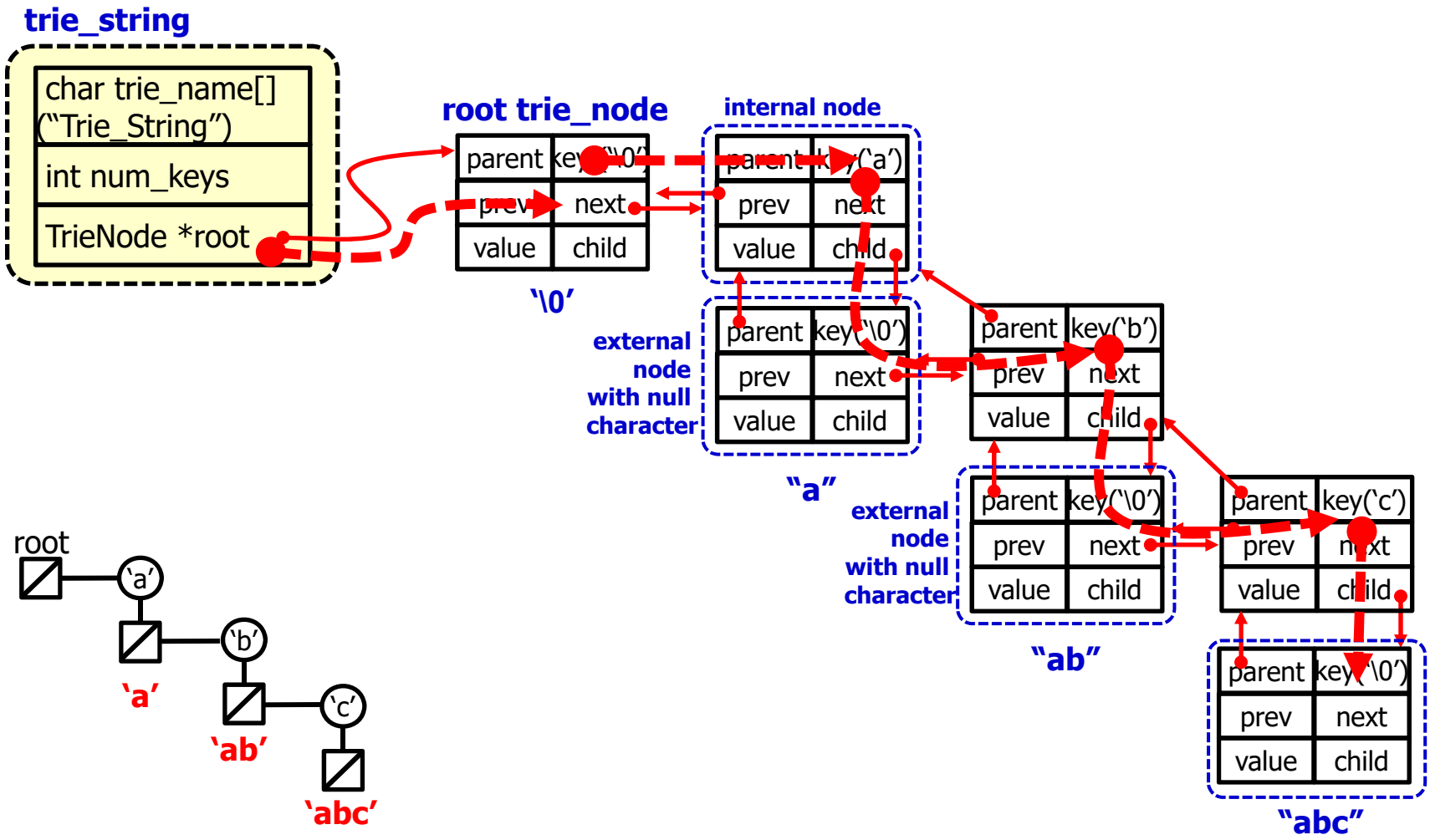
```
/* Trie.h  (6) */

    else if ((pTN->getKey() > *keyPtr) && (*keyPtr != '\0'))
    {
        /* insert between two existing trie nodes */
        pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
        pTN_New->setNext(pTN);
        pTN_New->setParent(pTN->getParent());
        if (pTN->getPrev() == NULL)
        {   /* this pTN_new becomes the new first in this level */
             if (pTN->getParent() != NULL)
                  (pTN->getParent())->setChild(pTN_New);
        } else {
             (pTN->getPrev())->setNext(pTN_New);
        }
        pTN_New->setPrev(pTN->getPrev());
        pTN->setPrev(pTN_New);

        pTN = pTN_New;
        keyPtr++;
        while (*keyPtr != '\0')
        {
             pTN_New = new TrieNode<E>(*keyPtr, VALUE_INTERNAL_NODE);
             pTN->setChild(pTN_New);
             (pTN->getChild())->setParent(pTN);
             pTN = pTN->getChild();
             keyPtr++;
        }
        if (*keyPtr == '\0')
        {
             insertExternalTN(pTN, keyWord, value);
             this->num_keys++;
             return;
        }
    }
}
```



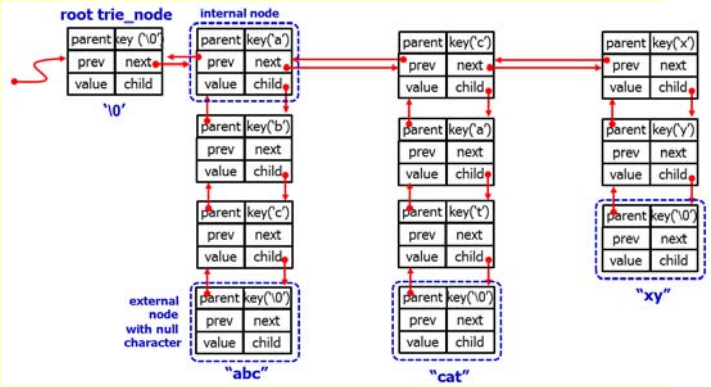
trie 자료구조에서의 연산 (2)
- findKeyWord(), findPrefixMatch()

findKeyWord() in trie – Full Match

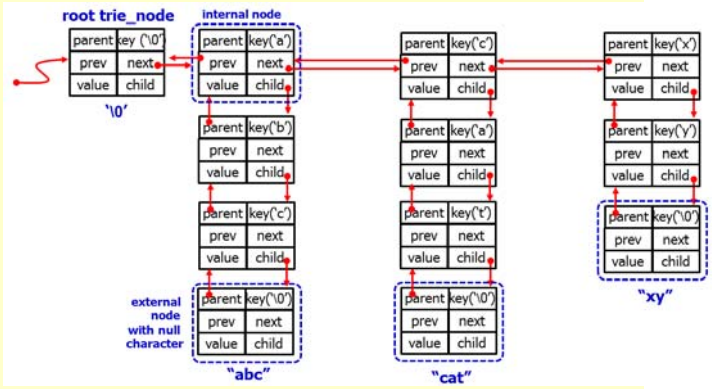
◆ findKeyWord ("abc")



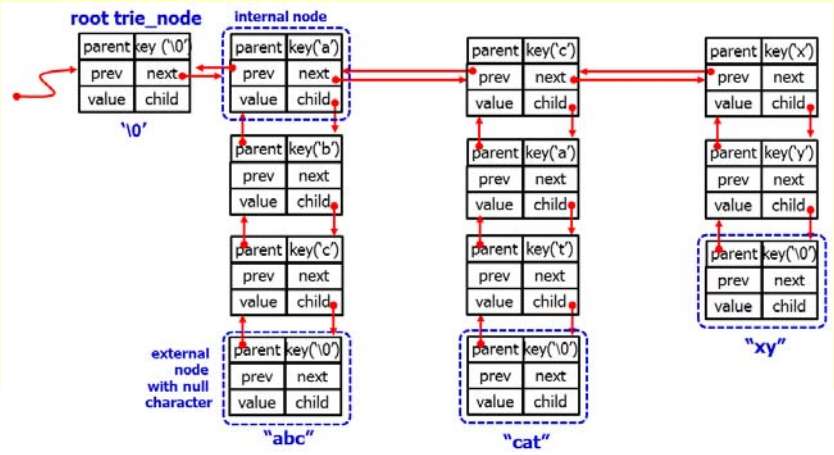
```
/* Trie.h  (7) */

template<typename E>
TrieNode<E> *Trie<E>::findKeyWord(const char *keyWord)
{
        TrieNode<E> *pTN = NULL;

        pTN = _find(keyWord, FULL_MATCH);
        return pTN;
}

template<typename E>
TrieNode<E> * Trie<E>::_find(const char * keyStr, SearchMode sm = FULL_MATCH)
{
      const char *keyPtr;
      TrieNode<E> *pTN = NULL;
      TrieNode<E> *found = NULL;

      if (keyStr == NULL)
           return NULL;

      keyPtr = keyStr;
      pTN = this->_root;
      while ((pTN != NULL) && (*keyPtr != '\0'))
      {
            while ((pTN != NULL) && (pTN->getKey() < *keyPtr))
            {
                  if (pTN->getNext() == NULL)
                        return NULL;
                  pTN = pTN->getNext();
            }
```



```
/* Trie.h  (8) */

        if ((pTN != NULL) && (pTN->getKey() > *keyPtr))
        {
            // key not found
            return NULL;
        }
        else if ((pTN == NULL) && (*keyPtr != '\0'))
        {
            // key not found
            return NULL;
        }
        else if ((pTN->getKey() == *keyPtr) && (*keyPtr != '\0'))
        {
            pTN = pTN->getChild();
            keyPtr++;
            if (*keyPtr == '\0')
            {
                /* key or prefix found */
                if (sm == FULL_MATCH)
                {
                    if (pTN->getKey() == '\0')
                    {
                        /* found the key string as a full-match */
                        return pTN;
                    }
                    else // (pTN->getKey() != '\0')
                    {
                        /* found the key string as a substring of a longer existing string */
                        return NULL;
                    }
                }
```



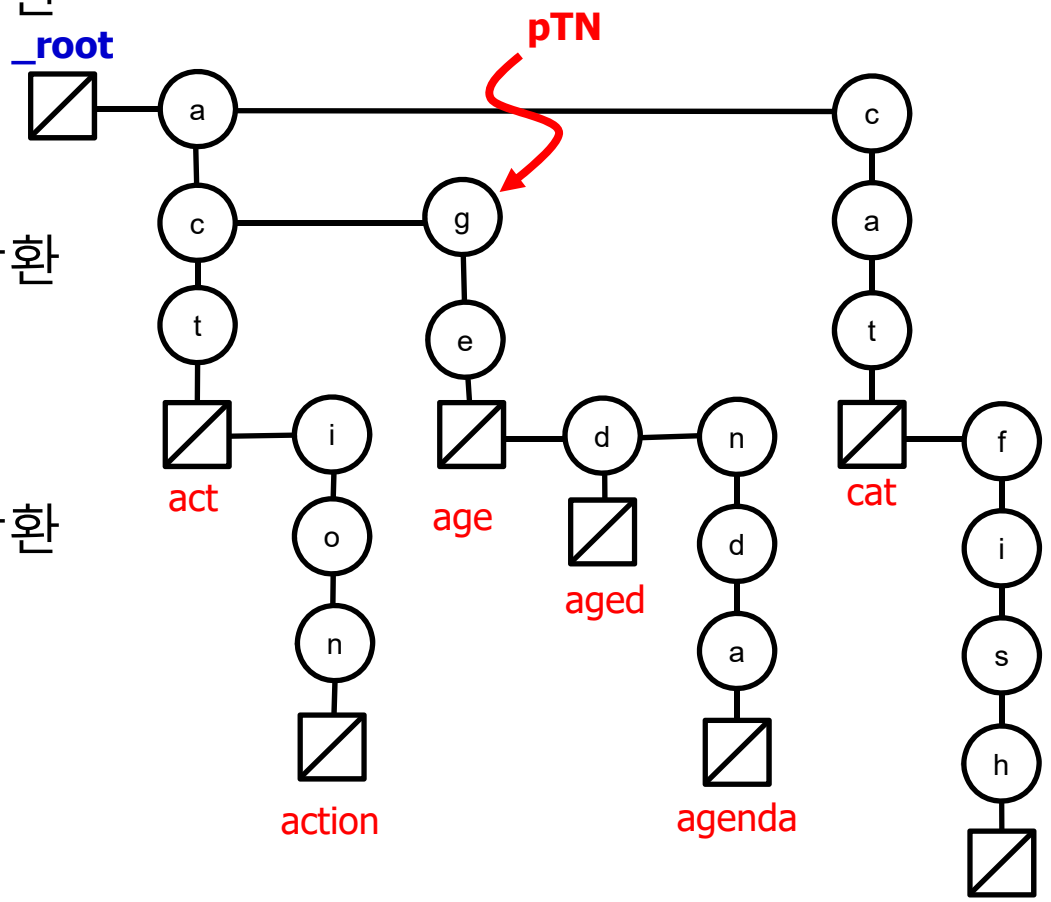
```
/* Trie.h  (9) */

                        else if (sm == PREFIX_MATCH)
                        {
                            /* found the key string as a full-match or as a substring of a longer existing
                            string */
                            return pTN;
                        }
                    }
                    else if ((pTN->getKey() == '\0') && (*keyPtr != '\0'))
                    {
                        if (pTN->getNext() != NULL)
                        {
                            pTN = pTN->getNext();
                            continue;
                        }
                        else
                            return NULL;
                    }
                    else
                    {
                        continue;
                    }

                }
            } // end while
}
```



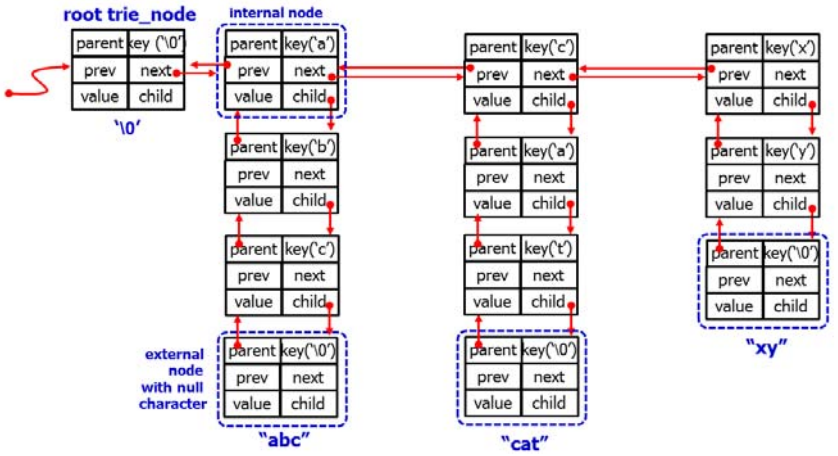
findPrefixMatch() in trie

◆ traverse(TrieNode<E> *pTN, STL_list& list_keywords)

- pTN이 가리키는 현재 위치의 prefix를 가지는 모든 단어들을 list_keywords에 담아 반환
- 만약 pTN이 "ac" prefix를 가리키는 경우, act, action을 list_keywords에 담아 반환
- 만약 pTN이 "ag" prefix를 가리키는 경우, age, aged, agenda를 list_keywords에 담아 반환



```
/* Trie.h  (9) */


template<typename E>
void Trie<E>::_traverse(TrieNode<E> *pTN, STL_list& list_keywords)
{
        if (pTN == NULL)
                return;
        if (pTN->getChild() == NULL)
        {
                list_keywords.push_back(pTN->getValue());
        }
        else
        {
                _traverse(pTN->getChild(), list_keywords);
        }

        if (pTN->getNext() != NULL)
        {
                _traverse(pTN->getNext(), list_keywords);
        }
}
```

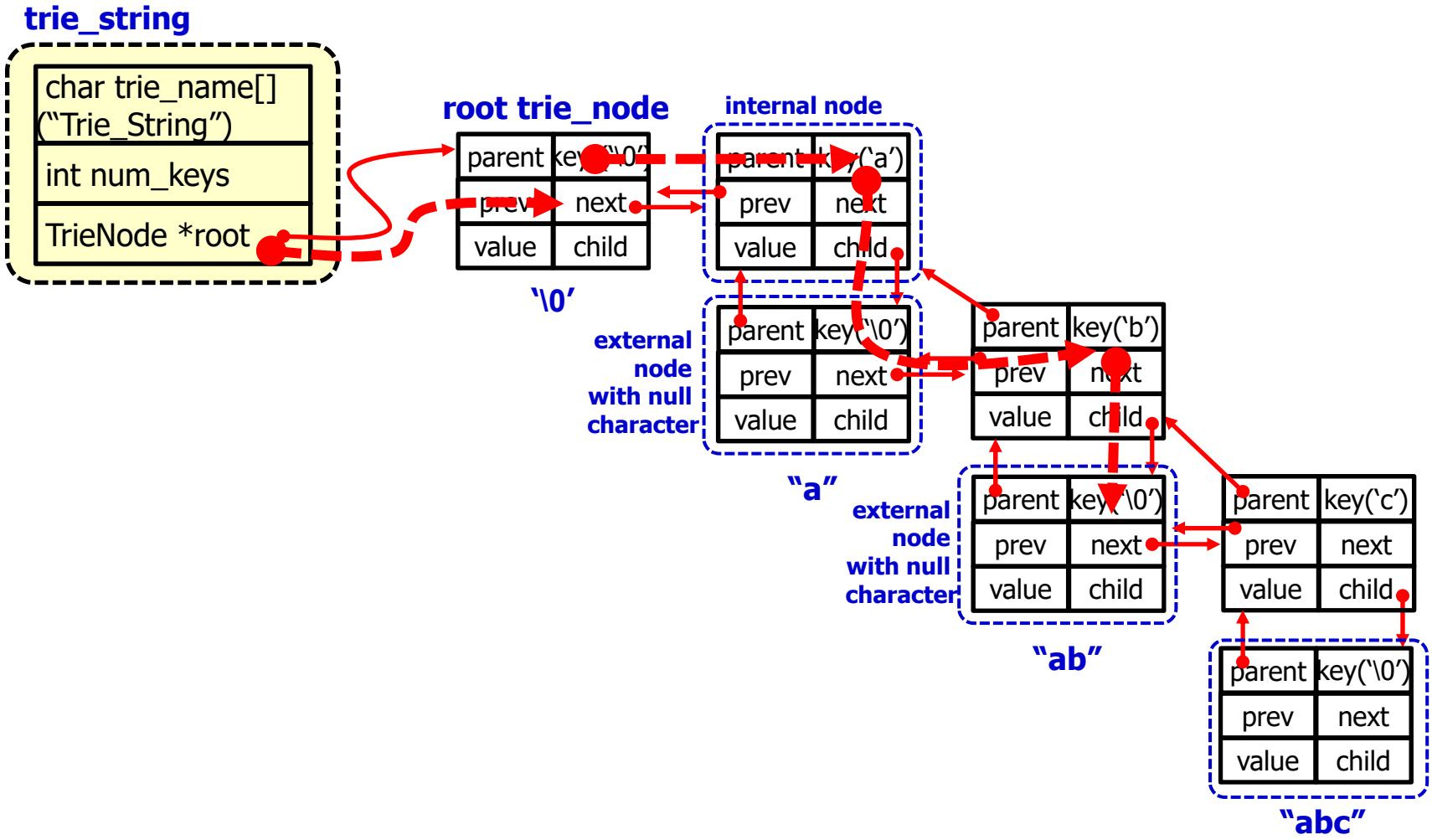


```
/* Trie.h  (9) */

template<typename E>
void Trie<E>::findPrefixMatch(const char * pPrefix, List_String& predictWords)
{
        TrieNode<E> *pTN = NULL;

        if (pPrefix == NULL)
                return;

        pTN = _find(pPrefix, PREFIX_MATCH);

        _traverse(pTN, predictWords);

        //printf("Error in TrieSearch (key: %s) !!\n", keyWord);

}
```

trie 자료구조에서의 연산 (3)
- deleteKeyWord(), eraseTrie()
- fprintTrie()

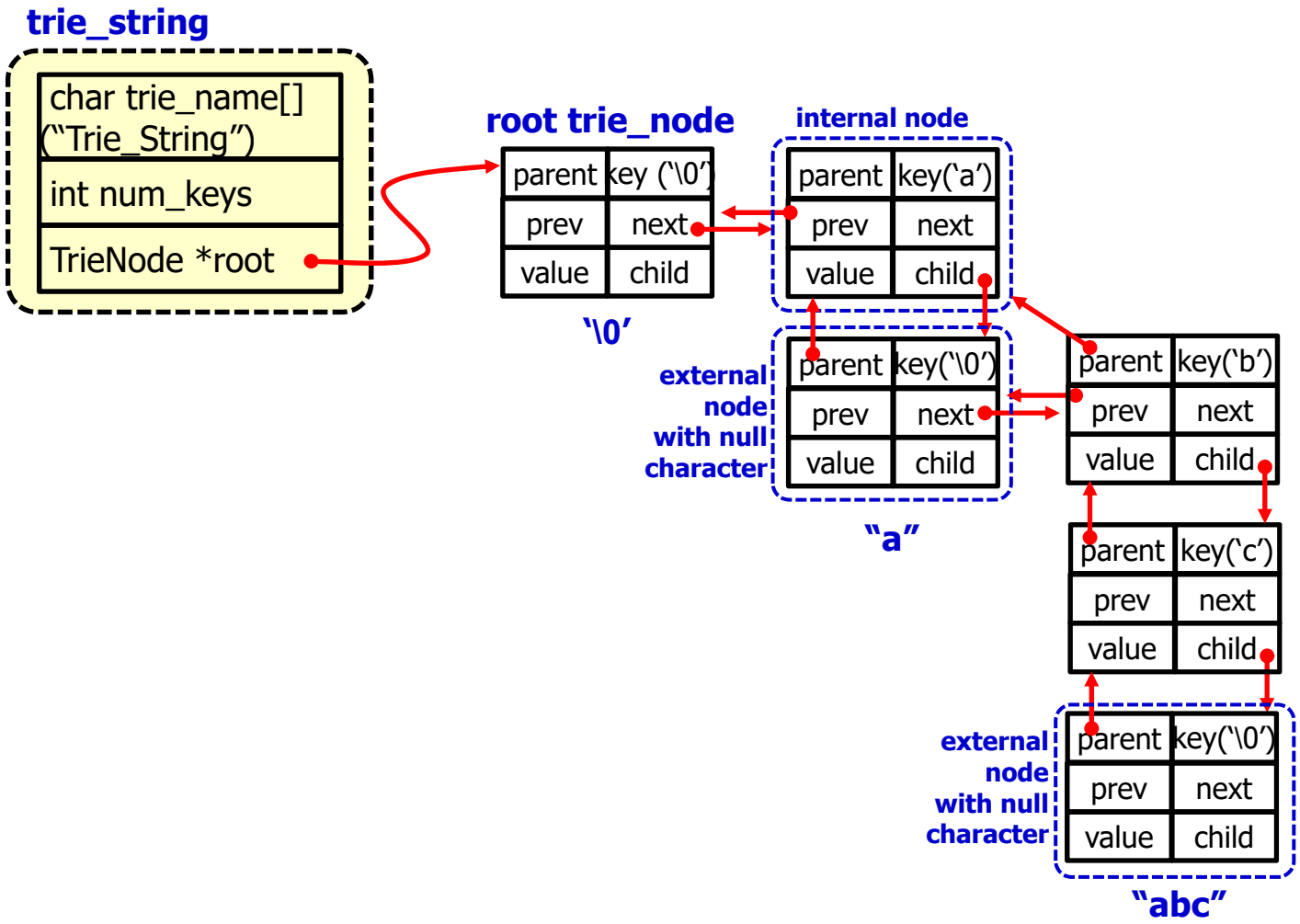
deleteKeyWord() in trie (1)

◆ findKeyWord ("ab") for deleteKeyWord("ab")

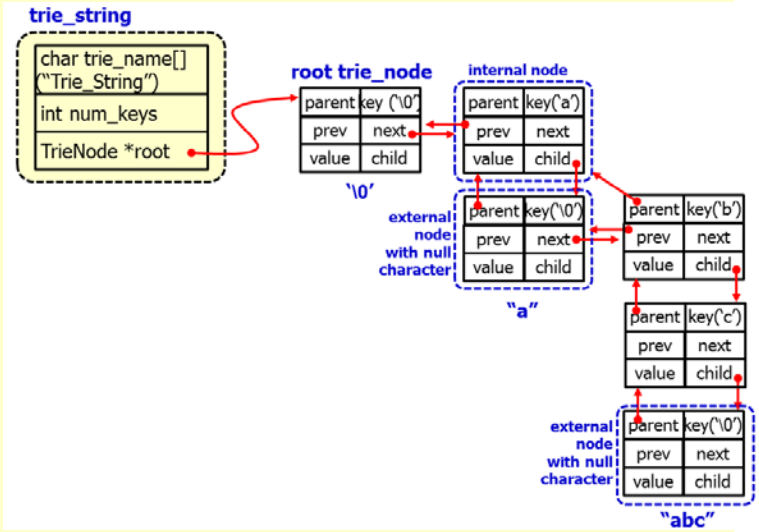


deleteKeyWord() in trie (2)

◆ delete the portion of "ab"

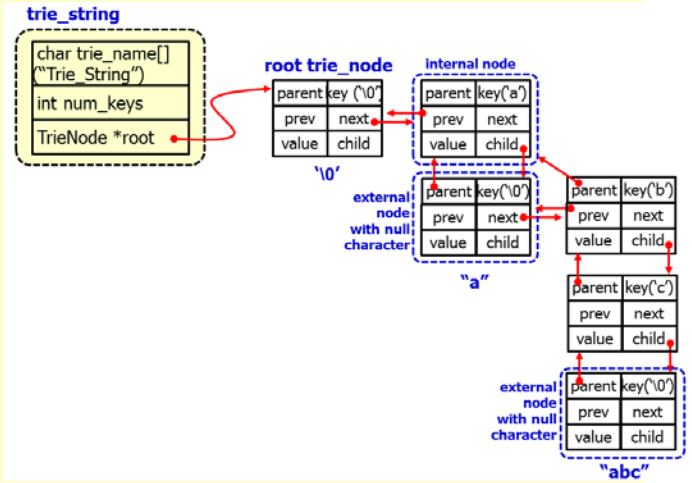


/* Trie.h (9) */




```
template<typename E>
void Trie<E>::deleteKeyWord(const char * keyWord)
{
	TrieNode<E> *pTN = NULL, *_root;
	TrieNode<E> *tmp = NULL;
	int trie_val;

	_root = this->_root;
	if (NULL == _root || NULL == keyWord)
		return;

	pTN = _find(keyWord, FULL_MATCH);

	if (pTN == NULL)
	{
		cout << "Key [" << keyWord << "] not found in trie" << endl;
		return;
	}

	while (1)
	{
		if (pTN == NULL)
			break;
		if (pTN->getPrev() && pTN->getNext())
		{
			tmp = pTN;
			(pTN->getNext())->setPrev(pTN->getPrev());
			(pTN->getPrev())->setNext(pTN->getNext());
			free(tmp);
			break;
		}
```

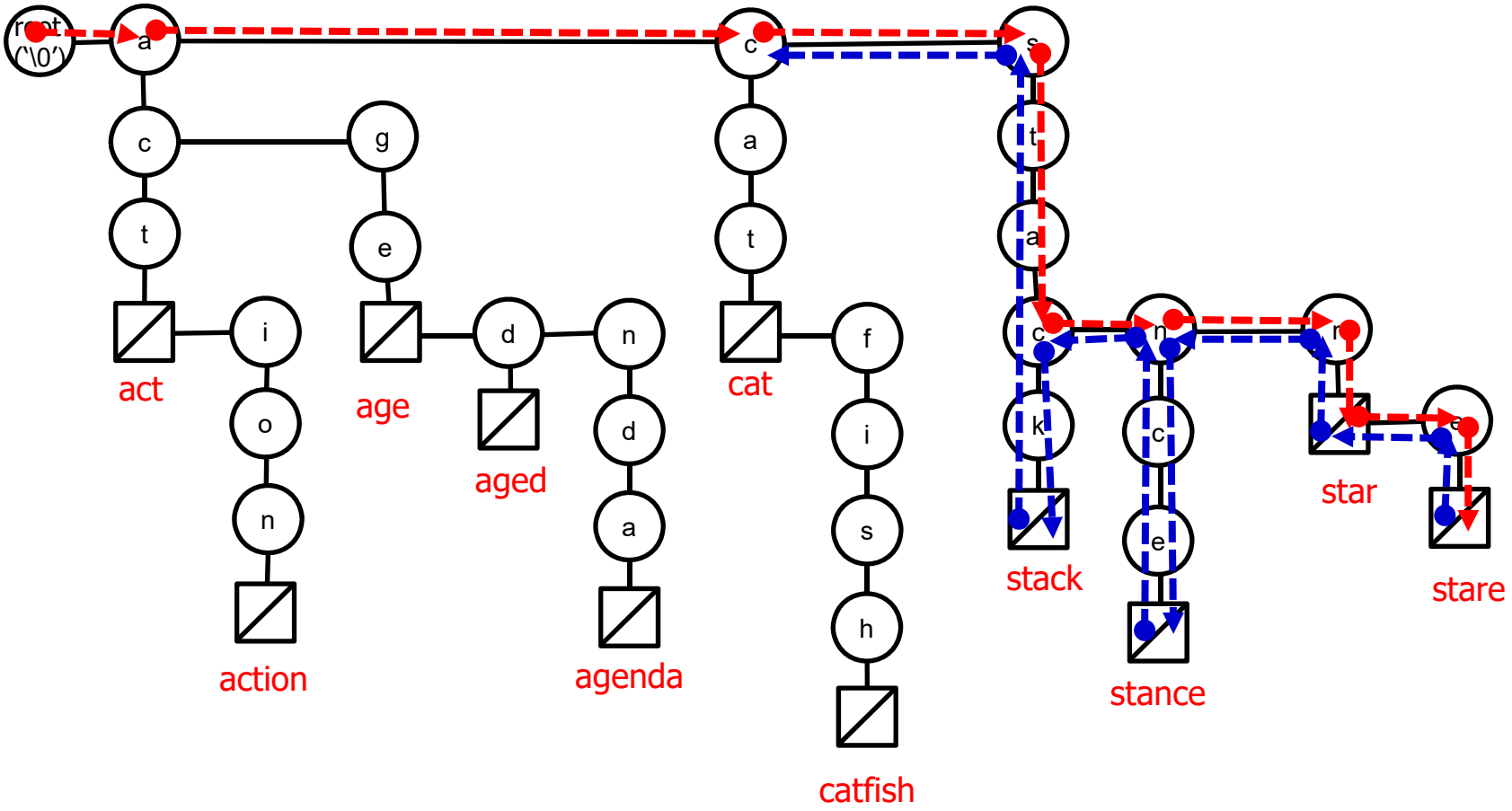
```
/* Trie.h  (10) */

                else if (pTN->getPrev() && !(pTN->getNext()))
                {
                        tmp = pTN;
                        (pTN->getPrev())->setNext(NULL);
                        free(tmp);
                        break;
                }
                else if (!(pTN->getPrev()) && pTN->getNext())
                {
                        tmp = pTN;
                        (pTN->getParent())->setChild(pTN->getNext());
                        pTN = pTN->getNext();
                        pTN->setPrev(NULL);
                        free(tmp);
                        break;
                }
                else
                {
                        tmp = pTN;
                        pTN = pTN->getParent();
                        if (pTN != NULL)
                        pTN->setChild(NULL);
                        free(tmp);
                        if ((pTN == _root) && (pTN->getNext() == NULL) && (pTN->getPrev() == NULL))
                        {
                                cout << "Now, the trie is empty !!" << endl;
                                break;
                        }
                }
        }
        this->num_keys--;
}
```



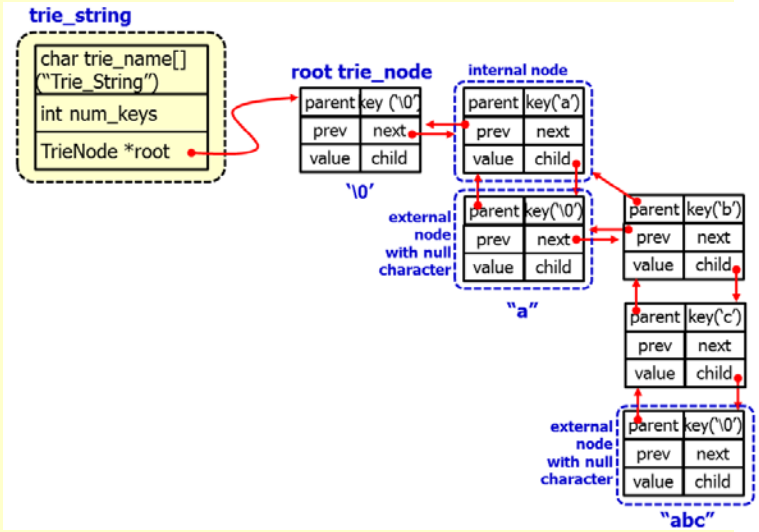
erase trie

◆ trie에 포함된 internal/external trie-node 삭제

- 가장 마지막 key string의 마지막 character 부터 역순으로 삭제

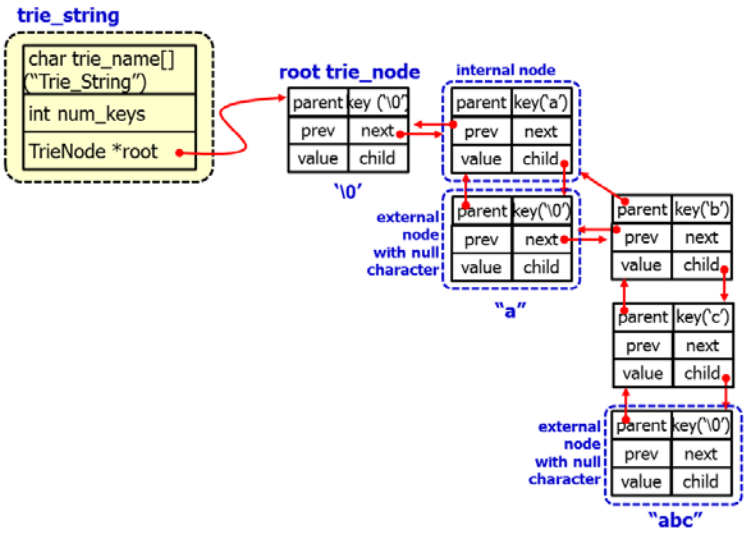


```
/* Trie.h  (11) */

template<typename E>
void Trie<E>::eraseTrie()
{
        TrieNode<E> *pTN;
        TrieNode<E> *pTN_to_be_deleted = NULL;

        if (this->_root == NULL)
                return;
        pTN = this->_root;

        /* delete the last key word first */

        while (pTN != NULL)
        {
                while ((pTN != NULL) &&(pTN->getNext()))
                        pTN = pTN->getNext();
                while (pTN->getChild())
                {
                        if (pTN->getNext())
                                break;
                        pTN = pTN->getChild();
                }
                if (pTN->getNext())
                        continue;

                if (pTN->getPrev() && pTN->getNext())
                {
                        pTN_to_be_deleted = pTN;
                        (pTN->getNext())->setPrev(pTN->getPrev());
                        (pTN->getPrev())->setNext(pTN->getNext());
                        pTN = pTN->getNext();
                        free(pTN_to_be_deleted);
                }
```



```
/* Trie.h  (11) */

            else if (pTN->getPrev() && !(pTN->getNext()))
            {
                    pTN_to_be_deleted = pTN;
                    (pTN->getPrev())->setNext(NULL);
                    pTN = pTN->getPrev();
                    free(pTN_to_be_deleted);
            }
            else if (!(pTN->getPrev()) && pTN->getNext())
            {
                    pTN_to_be_deleted = pTN;
                    (pTN->getParent())->setChild(pTN->getNext());
                    (pTN->getNext())->setPrev(NULL);
                    pTN = pTN->getNext();
                    free(pTN_to_be_deleted);
            }
            else
            {
                    pTN_to_be_deleted = pTN;
                    if (pTN == this->_root)
                    {
                            /* _root */
                            this->num_keys = 0;

                            return;
                    }
                    if (pTN->getParent() != NULL)
                    {
                            pTN = pTN->getParent();
                            pTN->setChild(NULL);
                    }
```

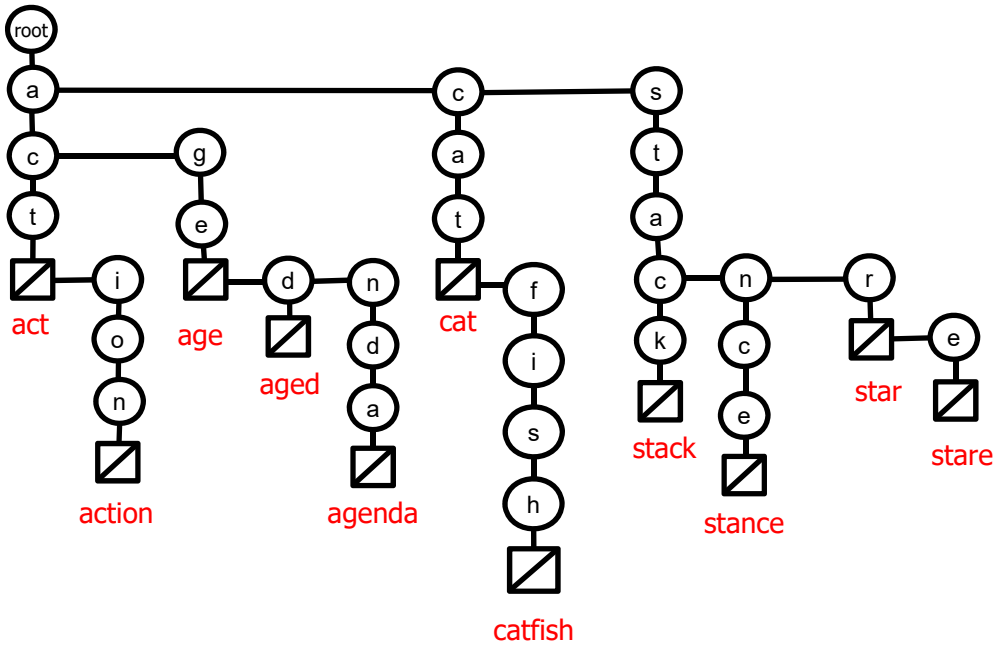
```
/* Trie.h  (11) */

                    else
                    {
                            pTN = pTN-
                            >getPrev();
                    }

                    free(pTN_to_be_deleted);
            } // end if - else
        }  // end while
}
```



fprint trie

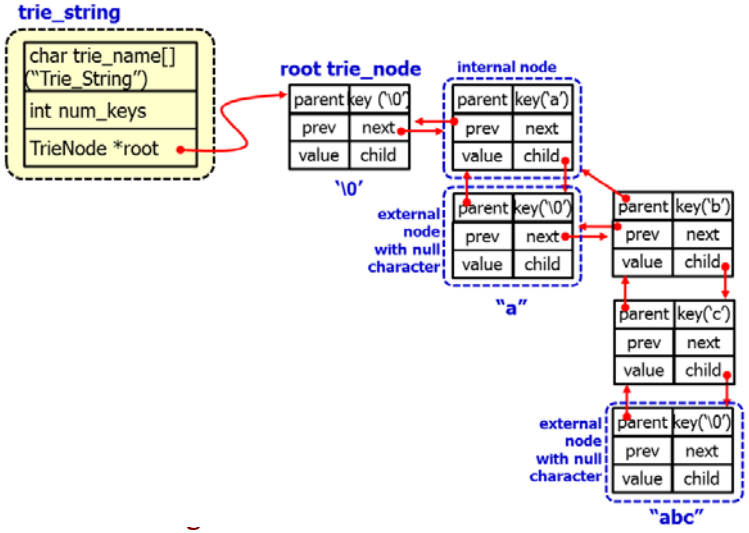
◆ trie에 포함된 모든 key string을 차례로 출력

- 들여쓰기 (indentation)를 사용하여 substring 관계를 표시




```
act          // act
   ion       // action
 ge          //age
   d         //aged
   nda       //agenda
cat          //cat
   fish      //catfish
stack        //stack
   nce       //stance
   r         //star
     e       //stare
```

```
/* Trie.h  (11) */

template<typename E>
void Trie<E>::fprintTrie(ostream& fout)
{
    TrieNode<E> *pTN;
    int indent = 0;

    fout << "trie ( " << this->trie_name << ") with "
         << this->num_keys << " trie_nodes\n";
    if (this->num_keys == 0)
    {
        fout << "Empty trie !" << endl;
        return;
    }
    pTN = this->_root;
    pTN->_fprint(fout, pTN, indent);
}
#endif
```




```
/* TrieNode.h  (3) */

template<typename E>
void TrieNode<E>::_fprint(ostream& fout,
    TrieNode<E> *pTN, int indent)
{
    if (pTN == NULL)
    {
        fout << endl;
        return;
    }
    else
    {
        fout << pTN->key;
        _fprint(fout, pTN->child, indent + 1);
        if (pTN->next == NULL)
            return;
        for (int i = 0; i < indent; i++)
            fout << " ";
        _fprint(fout, pTN->next, indent);
    }
}
```

trie 자료구조의 응용

trie의 C++ 프로그램 모듈 구현 (3) – main()

```
/* main_trie.cpp (1) */
#include <iostream>
#include <fstream>
#include <list>
#include "Trie.h"
#include "TrieNode.h"

using namespace std;

const char *test_strings_A[] =
{
    "a", "ab", "abc", "abcdefg", "abnormal", "abridge", "abreast", "abroad", "absence", "absolute",
    "andrew",
    "zealot", "yacht", "xerox",
    "tina",
    "arcade",
    "timor", "tim", "ti",
    "amy", "aramis",
    "best", "christmas", "beyond", "church",
    "apple", "desk", "echo", "car", "dog", "friend", "golf", "global",
    "ABCD", "XYZ", "Korea"
};
const char *test_strings_B[] =
{
    "act", "action", "age", "aged", "agenda", "cat", "stack", "stance", "star", "stare", "catfish"
};
```

```
/* main_trie.cpp (2) */

void main()
{
        ofstream fout;
        Trie<string> trieStr("TestTrie of Key Strings");
        int num_test_strings = 0;
        int trie_value;
        const char *pTest_Str;
        string sampleStr;
        TrieNode<string> *pTN;

        fout.open("output.txt");
        if (fout.fail())
        {
                printf("Error in opening output file !\n");
                exit;
        }

        /* Testing Basic Operation in trie */
        fout << "Testing basic operations of trie inserting ..... " << endl;
        trieStr.insert("xyz", string("xyz"));
        trieStr.insert("ABCD", string("ABCD"));
        trieStr.insert("ABC", string("ABC"));
        trieStr.insert("AB", string("AB"));
        trieStr.insert("A", string("A"));
        trieStr.insert("xy", string("xy"));
        trieStr.insert("x", string("x"));
        trieStr.fprintTrie(fout);
```

```
Testing basic operations of trie inserting .....
trie ( TestTrie of Key Strings) with 7 trie_nodes

A
 B
  C
   D
x
 y
  z

Testing TrieDestroy...
trie ( TestTrie of Key Strings) with 0 trie_nodes
Empty trie !
```

```
/* main_trie.cpp (3) */

        /*Destroy the trie*/
        fout << "\nTesting TrieDestroy...\n";
        trieStr.eraseTrie();
        trieStr.fprintTrie(fout);

        /* Insert key strings into Trie_Str */
        num_test_strings = sizeof(test_strings_A) /
        sizeof(char *);
        fout << "\nInserting " << num_test_strings << "
        keywords into trie data structure.\n";
        for (int i = 0; i < num_test_strings; i++)
        {
                pTest_Str = test_strings_A[i];
                sampleStr = string(test_strings_A[i]);
                if ((pTest_Str == NULL) ||
                   (*pTest_Str == '\0'))
                        continue;
                trieStr.insert(pTest_Str, sampleStr);
                //fout << "Inserting " << i
                     << "-th key_string " << pTest_Str << ", ";
                //trieStr.fprintTrie(fout);
                //fout.flush();
        }

        fout << "\nResult of the TrieAdd_InOder() for "
             << num_test_strings << " keywords : \n";
        trieStr.fprintTrie(fout);
```

```
Testing TrieDestroy...
trie ( TestTrie of Key Strings) with 0 trie_nodes
Empty trie !
```

```
Result of the TrieAdd_InOder() for 36keywords :
trie ( TestTrie of Key Strings) with 36 trie_nodes

ABCD
Korea
XYZ
a
 b
  c
   defg
  normal
  reast
   idge
   oad
  sence
   olute
 my
 ndrew
 pple
 ramis
  cade
best
  yond
car
 hristmas
  urch
desk
 og
echo
friend
global
 olf
ti
  m
   or
  na
xerox
yacht
zealot
```

```
/* main_trie.cpp (4) */

        fout << "\nTesting trie_findKeyWord for "
        << num_test_strings << " keywords from trie data
        structure.\n";
        for (int i = 0; i < num_test_strings; i++)
        {
                pTest_Str = test_strings_A[i];
                if ((pTest_Str == NULL) || (*pTest_Str == '\0'))
                        continue;
                pTN = trieStr.findKeyWord (pTest_Str);
                if (pTN != NULL)
                {
                        fout << "Trie_findKeyWord ("
                        << pTest_Str << ") = > trie_value("
                        << pTN->getValue() << ")\n";
                }
                else
                {
                        fout << "Trie_findKeyWord ("
                        << pTest_Str << ") = > not found !!\n";
                }
        }

        char prefix[] = "ab";
        List_String predictWords;
        List_String_Iter itr;
        predictWords.clear();
```

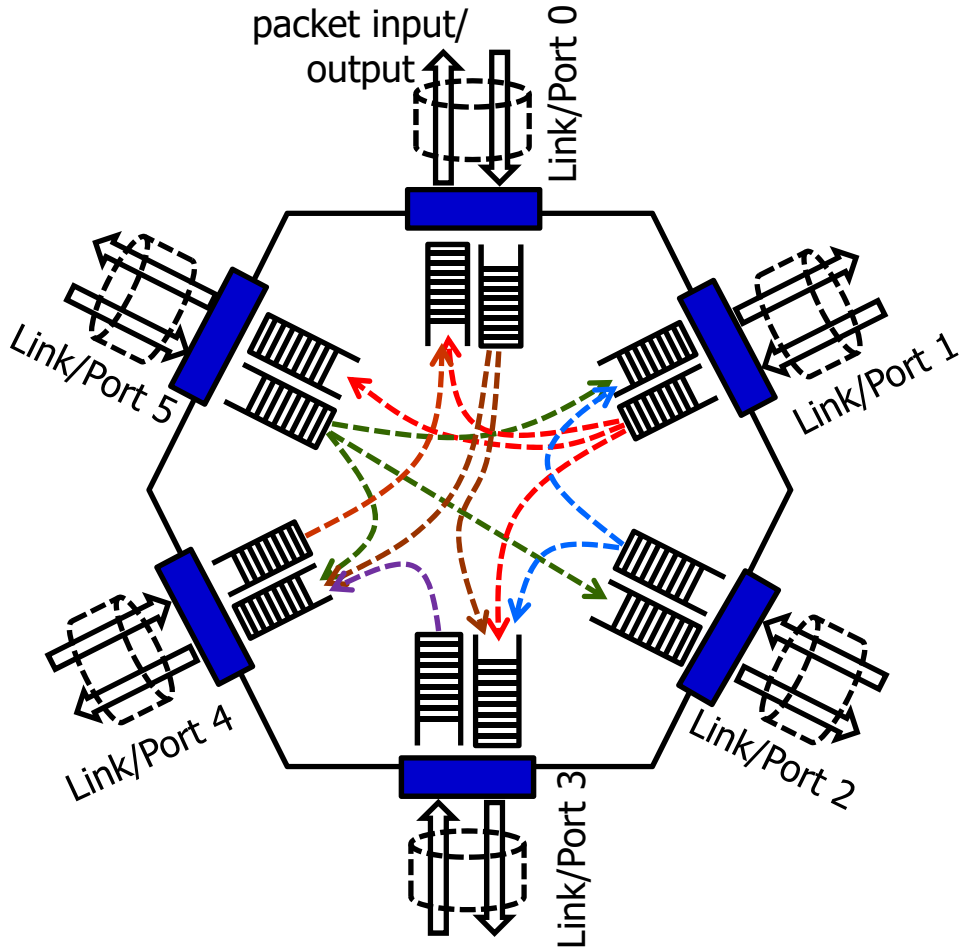
```
Testing trie_find for 36 keywords from trie data structure.
Trie_find (a) = > trie_value(a)
Trie_find (ab) = > trie_value(ab)
Trie_find (abc) = > trie_value(abc)
Trie_find (abcdefg) = > trie_value(abcdefg)
Trie_find (abnormal) = > trie_value(abnormal)
Trie_find (abridge) = > trie_value(abridge)
Trie_find (abreast) = > trie_value(abreast)
Trie_find (abroad) = > trie_value(abroad)
Trie_find (absence) = > trie_value(absence)
Trie_find (absolute) = > trie_value(absolute)
Trie_find (andrew) = > trie_value(andrew)
Trie_find (zealot) = > trie_value(zealot)
Trie_find (yacht) = > trie_value(yacht)
Trie_find (xerox) = > trie_value(xerox)
Trie_find (tina) = > trie_value(tina)
Trie_find (arcade) = > trie_value(arcade)
Trie_find (timor) = > trie_value(timor)
Trie_find (tim) = > trie_value(tim)
Trie_find (ti) = > trie_value(ti)
Trie_find (amy) = > trie_value(amy)
Trie_find (aramis) = > trie_value(aramis)
Trie_find (best) = > trie_value(best)
Trie_find (christmas) = > trie_value(christmas)
Trie_find (beyond) = > trie_value(beyond)
Trie_find (church) = > trie_value(church)
Trie_find (apple) = > trie_value(apple)
Trie_find (desk) = > trie_value(desk)
Trie_find (echo) = > trie_value(echo)
Trie_find (car) = > trie_value(car)
Trie_find (dog) = > trie_value(dog)
Trie_find (friend) = > trie_value(friend)
Trie_find (golf) = > trie_value(golf)
Trie_find (global) = > trie_value(global)
Trie_find (ABCD) = > trie_value(ABCD)
Trie_find (XYZ) = > trie_value(XYZ)
Trie_find (Korea) = > trie_value(Korea)
```

```
/* main_trie.cpp (5) */

        fout << "All predictive words with prefix (" << prefix << ") : ";
        trieStr.findPrefixMatch(prefix, predictWords);
        itr = predictWords.begin();
        for (int i = 0; i < predictWords.size(); i++)
        {
                fout << *itr << "  ";
                ++itr;
        }
        fout << endl;

        /* Testing TrieDeleteKey() */
        printf("\nTesting trie_delete_key for %d keywords
                 from trie data structure.\n", num_test_strings);
        for (int i = 0; i < num_test_strings; i++)
        {
                pTest_Str = test_strings_A[i];
                if ((pTest_Str == NULL) || (*pTest_Str == '\0'))
                        continue;
                fout << "Trie-Deleting (key : " << pTest_Str << ") ...\n";
                trieStr.deleteKeyWord(pTest_Str);
                //trieStr.fprintTrie(fout);
        }
        //trieStr.fprintTrie(fout);

        fout.close();
}
```

```
All predictive words with prefix (ab) : ab  abc  abcdefg
  abnormal  abreast  abridge  abroad  absence  absolute


Trie-Deleting (key : a) ...
Trie-Deleting (key : ab) ...
Trie-Deleting (key : abc) ...
Trie-Deleting (key : abcdefg) ...
Trie-Deleting (key : abnormal) ...
Trie-Deleting (key : abridge) ...
Trie-Deleting (key : abreast) ...
Trie-Deleting (key : abroad) ...
Trie-Deleting (key : absence) ...
Trie-Deleting (key : absolute) ...
Trie-Deleting (key : andrew) ...
Trie-Deleting (key : zealot) ...
Trie-Deleting (key : yacht) ...
Trie-Deleting (key : xerox) ...
Trie-Deleting (key : tina) ...
Trie-Deleting (key : arcade) ...
Trie-Deleting (key : timor) ...
Trie-Deleting (key : tim) ...
Trie-Deleting (key : ti) ...
Trie-Deleting (key : amy) ...
Trie-Deleting (key : aramis) ...
Trie-Deleting (key : best) ...
Trie-Deleting (key : christmas) ...
Trie-Deleting (key : beyond) ...
Trie-Deleting (key : church) ...
Trie-Deleting (key : apple) ...
Trie-Deleting (key : desk) ...
Trie-Deleting (key : echo) ...
Trie-Deleting (key : car) ...
Trie-Deleting (key : dog) ...
Trie-Deleting (key : friend) ...
Trie-Deleting (key : golf) ...
Trie-Deleting (key : global) ...
Trie-Deleting (key : ABCD) ...
Trie-Deleting (key : XYZ) ...
Trie-Deleting (key : Korea) ...
```

Internet Protocol (IP) Router Model

◆ 인터넷 라우터 (IP router)의 기능 구조



인터넷 라우터 (패킷 교환기)의 기본 기능

◆ 인터넷 라우터 (패킷 교환기)의 기본 기능

- 인터넷 연결 정보 파악 및 network topology 구성: nodes, links
- 다른 라우터 들과의 최단 거리 경로를 계산하여 **shortest paths tree** 구성
- 패킷 교환을 위한 **forwarding table** (next hop for a target address) 구성
- 패킷 교환기에서도 패킷 생성 가능
  (인터넷 제어 및 관리를 위한 정보)

◆ IP 라우터에서 패킷이 수신된 후 처리 절차

- 입력 버퍼/큐에 도착한 패킷 확인
- 도착된 패킷의 목적지 주소 확인
- 패킷의 최종 목적지가 자기 자신이면 이 패킷의 전달 경로를 출력하고 폐기
- 패킷의 최종 목적지가 다른 곳인 경우 해당 목적지로 전달하기 위한 최단 거리 경로의 다음 순서 (next hop) 라우터로 연결된 출력 port를 찾고, 그 output buffer/queue에 패킷을 삽입

Packet Forwarding Table

◆ Initialization of Packet Forwarding Table

- for each destination address, the next hop (from this node) is found from the shortest paths tree (calculated by Dijkstra's algorithm)

◆ Longest Prefix Matching

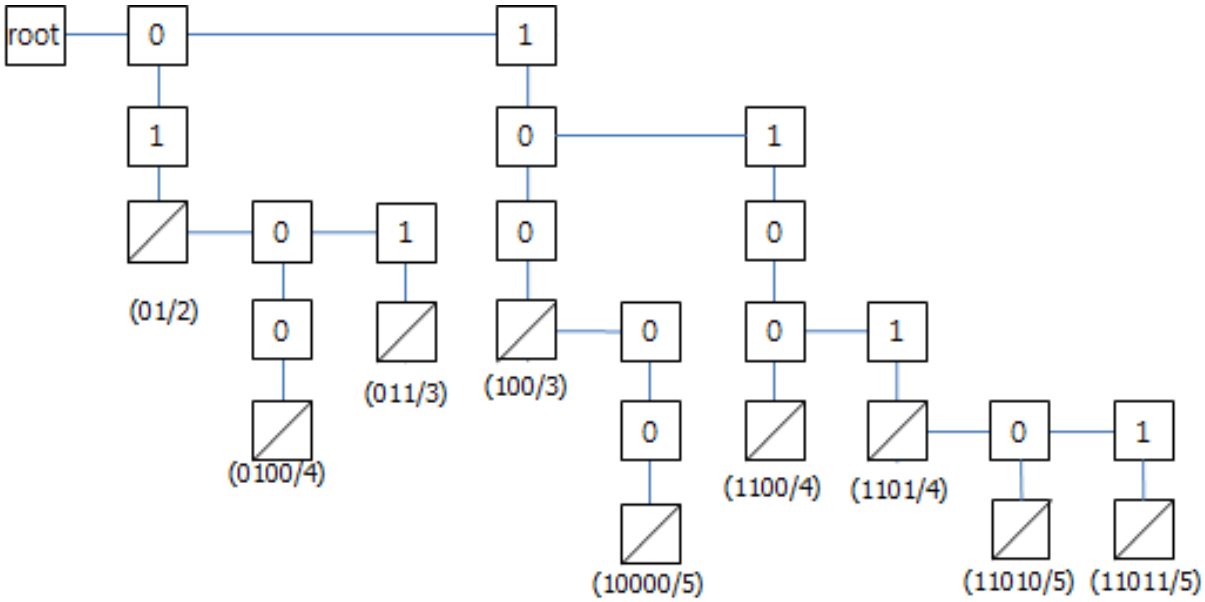
- Packet forwarding table에 있는 항목 중, 목적지 주소의 첫 부분 (prefix)가 가장 많이 일치하는 항목을 찾아 이 출력 port로 전달
- Longest prefix matching을 위하여 packet forwarding table 을 trie 구조로 구현할 수 있음
- IPv4의 경우 최대 32 비트 크기의 목적지 주소 (destination address)를 가지며, 최대 32번의 비교로 next hop (또는 목적지 도달) 결정

Longest Prefix Matching

◆ Longest Prefix Matching을 위한 Packet Forwarding Table

목적지 주소 prefix (32비트)				prefix 길이	prefix bit masking	출력 port
64	0	0	0	4	11110000 00000000 00000000 00000000	port 1
192	0	0	0	4	11110000 00000000 00000000 00000000	port 2
192	168	0	0	12	11111111 11110000 00000000 00000000	port 3
192	168	100	16	28	11111111 11111111 11111111 11110000	port 4
192	168	100	0	24	11111111 11111111 11111111 00000000	port 5
192	168	100	224	27	11111111 11111111 11111111 11100000	port 6
192	168	100	248	29	11111111 11111111 11111111 11111000	port 7

Longest Prefix Matching 구조 구현 예



Homework 12

Homework 12

12.1 일부 입력된 단어로 구성될 수 있는 예측구문 (predictive text)을 찾아주기 위한 trie 자료구조를 C++ 프로그램으로 구현하고 기능을 시험하라.

(1) C++ 클래스 구현

- template class TrieNode
- template class Trie

(2) 총 100 단어 (TOEIC vocabulary)를 trie에 입력

(3) trie의 내부 구조를 살펴볼 수 있도록 들여쓰기를 사용하여 전체 출력 기능 구현 및 시험

(4) 1 ~ 5 길이의 접두어 (prefix)를 입력 받고, 이 접두어로 구성될 수 있는 모든 단어들을 찾아 출력하는 프로그램을 구현 및 기능 시험

(5) trie에 포함된 단어들 중 지정된 단어를 삭제하는 기능 구현 및 시험

(6) 전체 trie를 삭제하는 기능 구현 및 시험