

Assignment01



■ 과 목 명: C++프로그래밍및실습

■ 담당교수: 박태준 교수님

■ 제 출 일: 2023.12.09

■ 학 과: 인공지능학부

■ 학 번 : 224479

■ 성 명: 박주훈

2023 년 12 월 09 일

Assignment01

224479 박주훈

전체적인 접근방식

이 과제는 주어진 정규식으로 그에 대한 상태머신을 생성하는 것입니다
따라서 먼저 주어진 정규식을 처리하는 함수를 제작하고 그 함수 내부에서
정규식의 조건에 따른 특수문자들을 따로 처리하는 방법을 사용하였습니다.

makeRegex()

이 함수는 주어진 정규식을 바탕으로 상태머신을 제작하는 함수입니다.

이 함수의 동작원리에 대해 설명 드리도록 하겠습니다

함수의 매개변수는 `const string& str` 이며 반환형은 `vector<node*>` 입니다. 따라서
main 함수에서의 호출 방식은

```
vector<node*> s1 = makeRegex("abcd");
```

와 같습니다.

다음으로는 함수의 내부 동작에 대해 설명하도록 하겠습니다.

이 함수는 입력받은 정규식의 첫 문자부터 차례로 탐색합니다.

➔ 일반문자

일반 문자가 나오면

```
newNode->init(nodeName, isTerminal, matchChar);
s.push_back(newNode);
if (i != 0) {
    auto it = s.end();
    it--;
    it--;
    (*it)->addNode(newNode);
}
```

위와 같은 구문으로 벡터에 노드를 추가하고 이전 노드와 연결합니다 여기서 (I!=0) 을 사용해서 i가 0 즉 연결할 이전 노드가 없을 때 이전 노드의 메모리를 잘못 참조하는 상황을 방지합니다

➔ ‘.’

.이 나오면 이 함수는 일반함수와 동일하게 .을 노드로 만듭니다

하지만 정규표현식에서 .의 의미는 아무 문자를 뜻하므로 makeRegex()함수가 아닌 문자열과 정규식을 테스트하는 input()함수를 수정하였습니다

```
if(state && (match == ch || match == '.')){
    if(isTerminal){
        cout << " >>> accepted by " << match << endl;
    }

    for(int i = 0 ; i < next.size(); i++){
        next[i]->transition();
        //cout << next[i] << next[i] ->name << next[i]->match << next[i]->transited << endl;
    }
}
```

위 코드는 수정된 input() 함수의 내부입니다. 정규식과 문자의 매칭 여부를 검사하는 조건식에서 match 즉 정규식 문자가 .이라면 state 가 1 일 때 매칭성공으로 검출하도록 수정하였습니다.

이러한 수정으로

```
string test1 = "abcdefghi";

vector<node*> s1 = makeRegex("ab.def.hi");
test(s1,test1);
```

위와 같은 코드에서 아래와 같이 정상적으로 정규식 검출이 가능합니다

```
/Users/juhun_park/Downloads/assignment0
test for ' abcdefghi'
>> input a
>> input b
>> input c
>> input d
>> input e
>> input f
>> input g
>> input h
>> input i
>>>> accepted by i

Process finished with exit code 0
```

➔ ‘*’

정규표현식에서 *문자는 특정문자의 0 회 이상의 반복입니다.

즉 A* 는 A 가 없는 경우,A,AA,AAA,AAAA 등을 검출해야 합니다.

먼저 makeRegex() 함수에 *문자를 검출하는 조건문을 작성합니다

단, 여기서 *문자가 식의 맨 첫번째에 온 경우 정규식 표현이 불가하기에

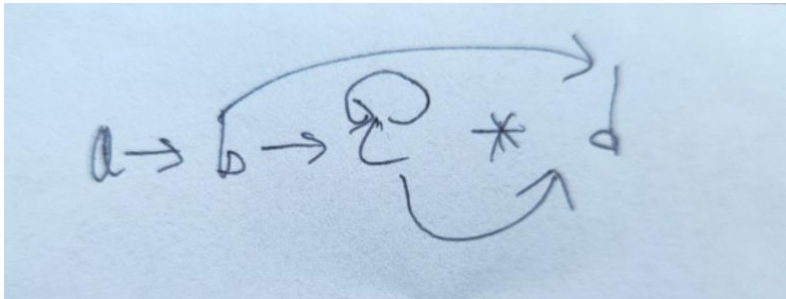
```
if(matchChar == '*' && !s.empty())
```

다음과 같은 조건문을 작성합니다

그리고 이 후 이전노드가 자기자신을 가리키도록 설정합니다 이로써 문자가 반복되는 상황을 처리하였습니다. 그리고 이 후 *문자 이후에 나오는 문자로

새로운 노드를 생성하고 이전노드와 이전의 이전노드가 새로운 노드를 가리키도록 설정합니다. 여기서 이전의 이전노드가 새로운 노드를 가리키는 이유는 특정문자가 0 회 반복되는 경우를 처리해야하기 때문입니다.

즉 노드의 구조는 다음과 같은 형태가 됩니다.



이로써 특정문자의 0 회 이상 반복의 경우를 구현하였습니다

하지만 *문자가 정규식의 맨 끝에 올 경우 이 코드는 정규식을 성공적으로 검출하지 못합니다. 따라서 저는 *문자가 정규식 맨 끝에 올 경우 맨 마지막 노드에 '\0' 즉 null 문자를 삽입합니다 그 이후 .문자의 처리와 비슷하게 input 함수에서 정규식 매칭 여부를 처리합니다.

```

if(!next.empty() &&( (next[0]->match != ((&ch)+1)[0] || (next[0]->match == 'W0' && ((&ch)+1)[0] == 'W0')) && ( state && (next.back()->transited && (next.back()->match == 'W0'))))

```

위 코드는 *문자가 정규식에 마지막에 위치하는 경우 null 문자를 검출하여 정규식 매칭여부를 검출하는 조건문 입니다.

(next[0]->match != ((&ch)+ 1)[0] || (next[0]->match == 'W0' && ((&ch)+ 1)[0] == 'W0')): 이 조건은 다음 노드의 매치 문자와 현재 처리 중인 문자의 다음 문자를 비교합니다. (&ch)+ 1 은 현재 문자 ch 의 주소에 1 을 더해 다음 문자의 주소를 얻고, [0]을 사용해 그 문자를 가져옵니다. 이 조건은 현재 문자가 다음 노드의 매치 문자와 다르거나, 현재 노드와 다음 노드 모두 종료 문자('W0')와 매치되는 경우를 확인합니다.

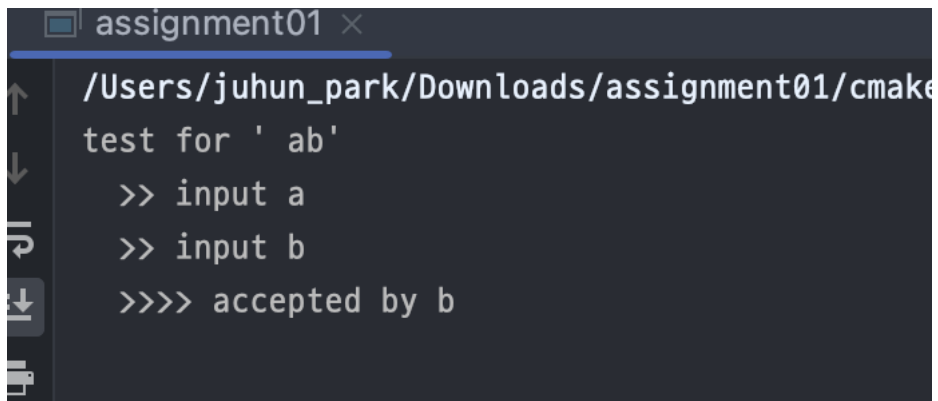
state && (next.back()->transited && (next.back()->match == 'W0')): 현재 노드가 활성화된 상태(state)이고, 다음 노드 리스트의 마지막 노드가 전환되었으며(transited), 그 마지막 노드가 종료 문자('W0')와 매치되는 경우를 확인합니다.

이러한 조건문을 사용하므로써 *가 마지막에 올 경우를 성공적으로 처리할 수 있습니다

예 제:

```
string test1 = "ab";  
  
vector<node*> s1 = makeRegex("abc*");  
test(s1,test1);
```

결과:



```
assignment01 x  
/Users/juhun_park/Downloads/assignment01/cmake  
test for ' ab'  
  >> input a  
  >> input b  
  >>>> accepted by b
```

예 제

```
string test1 = "abccccde";  
  
vector<node*> s1 = makeRegex("abc*de");  
test(s1,test1);
```

결과:

```

/Users/juhun_park/Downloads/assignment
test for ' abccccde'
>> input a
>> input b
>> input c
>> input c
>> input c
>> input c
>> input d
>> input e
>>>> accepted by e

```

예제:

```

string test1 = "abcccc";

vector<node*> s1 = makeRegex("abc*");
test(s1,test1);

```

결과:

```

/Users/juhun_park/Downloads/assignment
test for ' abcccc'
>> input a
>> input b
>> input c
>> input c
>> input c
>> input c
>>>> accepted by c

```

➔ '+'

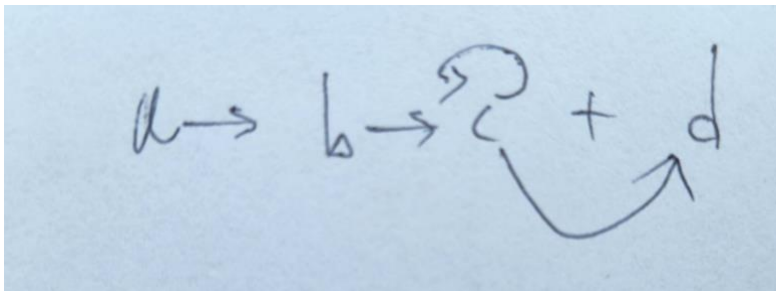
정규식에서 + 는 특정문자가 1 회이상 반복됨을 나타냅니다

예를 들어 A+ 는 A,AA,AAA...등과 매칭됩니다

+ 와 같은 경우는 + 와 아주 비슷하게 처리하였습니다.

단 노드의 연결부분에서 이전의 이전노드와 다음노드의 연결을 하지 않음으로써 1 회 이상 반복됨을 구현하였습니다

노드의 구조는 다음과 같습니다.



또한 + 가 정규식 마지막에 오는경우 또한 *와 동일하게 처리하였습니다

예제:

```
string test1 = "ab";  
  
vector<node*> s1 = makeRegex("abc+");  
test(s1,test1);
```

결과:

```
/Users/juhun_park/Downloads/assignment01/cma  
test for ' ab'  
  >> input a  
  >> input b  
  
Process finished with exit code 0
```

예제:


```
string test1 = "abcc";

vector<node*> s1 = makeRegex("abc+");
test(s1,test1);
```

결과:

```
/Users/juhun_park/Downloads/assign
test for ' abcc'
>> input a
>> input b
>> input c
>> input c
>>>> accepted by c
```

예제:

```
string test1 = "abccde";

vector<node*> s1 = makeRegex("abc+de");
test(s1,test1);
```

결과:

```
assignment 1
test for ' abccde'
>> input a
>> input b
>> input c
>> input c
>> input d
>> input e
>>>> accepted by e
```

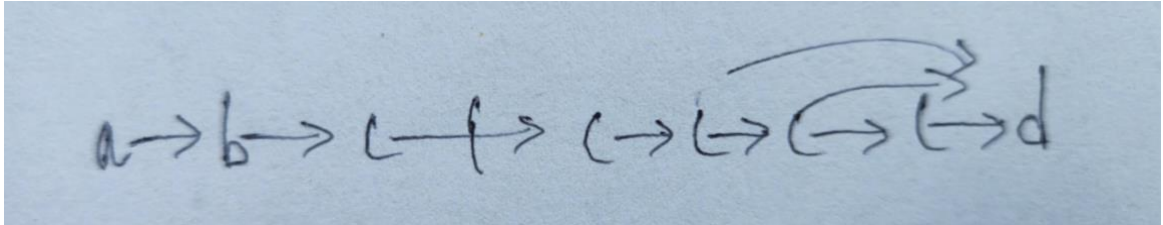
➔ '{,}'

{a,b}는 정규표현식에서 a 번 이상 b 번 이하 반복입니다.

즉 A{3,5} 정규식은 AAA,AAAA,AAAAA 와 매칭됩니다

이를 구현하기 위해서 저는 내부 카운터가 아닌 상태머신 구조를 이용하여 구현하였습니다. 최소 반복횟수는 링크를 '}' 문자 다음 노드로 연결하지 않고 최소 반복 이후 다음 노드들은 '}' 다음노드들과 연결을 시켜 최소 반복 횟수가 지난 후 언제든지 다음 노드들을 스킵할 수 있도록 구성하였습니다

아래는 abc{3,5}d 의 상태머신 구조입니다



{,}가 정규식 마지막에 나왔을 경우는 *,+ 와 동일하게 처리하였습니다

예제:

```
string test1 = "abccde";  
  
vector<node*> s1 = makeRegex("abc{3,5}de");  
test(s1,test1);
```

결과:

```
assignment1  
/Users/juhun_park/Downloads/a  
test for ' abccde'  
>> input a  
>> input b  
>> input c  
>> input c  
>> input d  
>> input e
```

예 제:

```
string test1 = "abccccde";  
  
vector<node*> s1 = makeRegex("abc{3,5}de");  
test(s1,test1);
```

결과:

```
/Users/juhun_park/Downloads/a  
test for ' abccccde'  
>> input a  
>> input b  
>> input c  
>> input c  
>> input c  
>> input c  
>> input d  
>> input e  
>>>> accepted by e
```

예 제:

```
string test1 = "abccccc";  
  
vector<node*> s1 = makeRegex("abc{3,5}");  
test(s1,test1);
```

결과:

```
/Users/juhun_park/Downloads/a  
test for ' abccccc'  
>> input a  
>> input b  
>> input c  
>> input c  
>> input c  
>> input c  
>> input c  
>>>> accepted by c
```

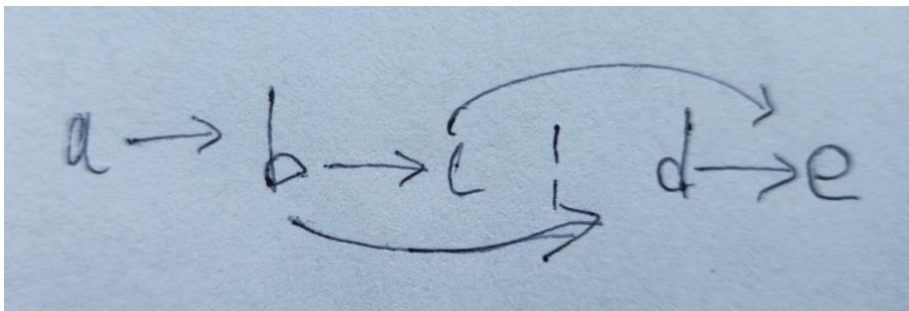
→ '|'

|는 정규표현식에서 A 또는 B 를 의미합니다.

즉 정규식 ABC|DE 는 ABCE,ABDE 와 매칭됩니다

위 코드에서 |가 검출되면 |문자 이후에 있는 문자로 새로운 노드를 만들고 그 노드를 이전의 이전 노드와 연결시킵니다 그 이후 |문자 다음의 다음 문자로 새로운 노드를 만들고 연결을 구성합니다

아래는 |를 사용한 상태머신(ABC|DE)의 구조입니다



|는 A|B 가 정규식의 맨 마지막에 나오는 경우를 null 문자로 처리하지 않고 changeIsTerminal()이라는 함수를 작성,사용하여 단말노드들의 isTerminal 값을 True 로 만들어 정규식 매칭을 구현하였습니다

예제:

```
string test1 = "abce";  
  
vector<node*> s1 = makeRegex("abc|de");  
test(s1,test1);
```

결과:

```
assignment01 x
/Users/juhun_park/Downloads/assignment01/cmake-bui
test for ' abce'
>> input a
>> input b
>> input c
>> input e
>>>> accepted by e

Process finished with exit code 0
```

예 제:

```
string test1 = "abde";

vector<node*> s1 = makeRegex("abc|de");
test(s1,test1);
```

결 과:

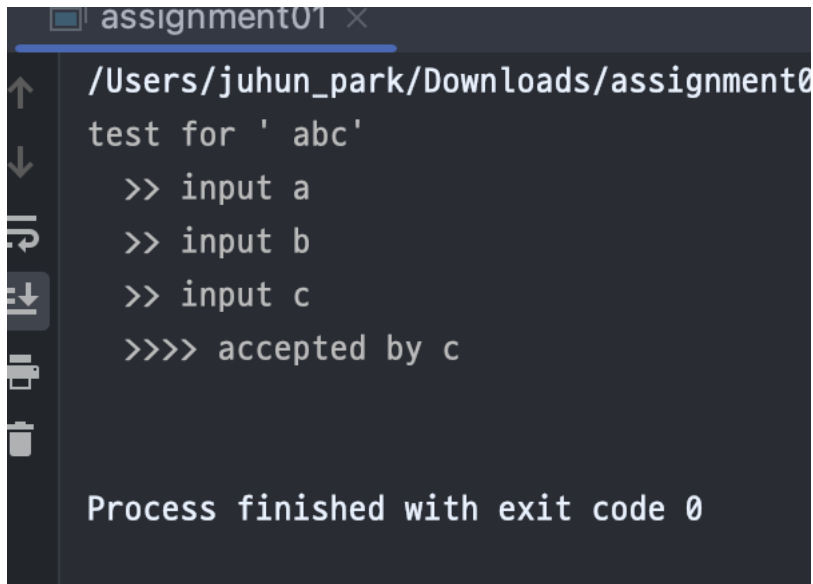
```
/Users/juhun_park/Downloads/assign
test for ' abde'
>> input a
>> input b
>> input d
>> input e
>>>> accepted by e
```

예 제:

```
string test1 = "abc";

vector<node*> s1 = makeRegex("abc|d");
test(s1,test1);
```

결 과:



```
assignment01 x
/Users/juhun_park/Downloads/assignment0
test for ' abc'
>> input a
>> input b
>> input c
>>>> accepted by c

Process finished with exit code 0
```

➔ 결과, 소감

저는 makeRegex() 함수를 제작하고 input() 함수를 수정하여 정규표현식 처리기를 구현하였습니다. 최대한 많은 경우의 수를 생각하여 작성하였으나 부족한 부분이 있을것이라고 생각합니다.

이 과제를 수행하면서 가장 많은 고민을 하였던 부분은 정규표현식 문법(+,*,{,|)이 정규식 마지막에 왔을 때 처리하는 부분이였습니다. 저는 null 문자를 사용하여 구현하였지만 훨씬 더 간단하고 좋은 방법이 있을것이라는 생각이 듭니다. 또한, {,}을 구현할 때도 내부카운터를 사용하는 방법이 아닌 상태머신의 연결구조를 이용하여 구현하였기 때문에 다른 방법이 존재할것이라고 생각합니다.

대학교를 와서 접한 과제 중 가장 어렵고 힘든 과제였습니다.

그 만큼 저 스스로에게도 많은 배움과 성장(특히 디버거 사용 실력)을 주었던 과제라고 생각합니다.

제 과제에 대한 교수님의 피드백을 듣고 싶습니다!!!

