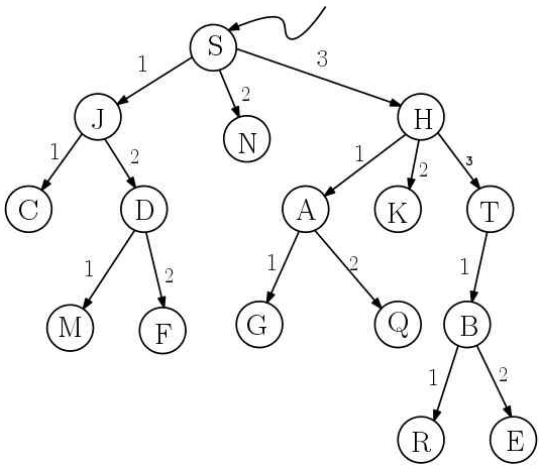
Lecture 8 : Stack의 실전 활용

어떤 집합에 여러 개체가 있을 때, 이들을 빠짐없이, 중복 없이 모두 헤아리는 그것은 매우 중요한 과정인데 이 일에 스택(stack)은 매우 유용하게 사용된다. 알고리즘에서 이 과정을 완전 탐색(exhaustive searching)이라고 부르고 그 방법으로 가장 대표적인 알고리즘 모형으로 되추적 (backtracking)이 있다.

스택에서 제공하는 두 가지의 기본 동작 push( ), pop( )을 사용하기는 매우 쉽다. 그러나 실제 현실에서 stack을 사용해야 하는 경우를 인식하여 직접 활용하기는 쉽지 않은 일이다. 특정한 작업에 핵심 자료구조가 stack임을 인식하는 것은 매우 어려운 일이며, 이것은 다양한 실전 문제를 푸는 과정에서 경험으로 알 수밖에 없는 일이다. 따라서 여러분은 많은 실전 문제와 구체적인 코딩을 통해서 자료구조가 손에 익도록 많은 연습을 해야 한다.

8.1 깜깜한 산에서 길 찾아 내려오기 - 산 정상에 도달하니 밖이 어두워져 앞길을 볼 수가 없게 되었다. 길은 정상에서부터 내려오는데 중간 중간, 다시 방향을 결정해야만 하는 갈래길(branch)이 있다. 여러분은 이 갈림길을 중복 없이 살펴서 민가(民家)로 내려오려고 한다. 어떻게 하면 될까요.

단 중요한 것은 어떤 갈래 길도 빠뜨리면 안 되며, 그리고 이미 한번 간 길(traversed path) 을 또다시 반복하여 방문하지 않도록 한다는 것이다. 즉 모두(all), 그리고 반복 없이 원하는 모든 대상을 방문하는 체계적인 방법, 즉 algorthmic한 방법을 만들어야 한다. 여러분은 S에서 출발한다. 아래 길 갈래 그림을 이용해서 제시한 방법을 설명해 보시오.



해답) S에서 출발하여 처음 만나게 되는 갈래 길은 {J, N, H} 3가지 길이다. 우리는 여러 갈래 길[1]이 있을 경우에는 체계적인 방문을 위하여 방문하지 않은 길 중에서 알파벳 순으로 선택한다. 따라서 S 다음에 방문해야 하는 노드는 {J}이다. 다음 {J}에서 방문할 노드는 {C}가 된다. 이제 {C} 에서는 더 이상 방문할 노드가 없으므로 backtrack이 발생한다. 이 과정을 Stack이 모두 기억해두도록 한다.

아래는 이 과정을 Stack으로 다루는 과정(왼쪽에서 오른쪽으로 변화함)을 보여주고

1) "여러 갈래길 누가 말하나 이 길뿐이라고 여러 갈래길 누가 말하나 저 길뿐이라고" from 김민기의 노래 <길>

있다. 중요한 것은 이 과정의 모든 상태는 linear 구조의 stack 하나로 관리된다는 것이다. 이 과정은 알고리즘 과목에서 다시 자세히 다루어질 것이다.

		C 0/0		D 1/2	D 2/2					
	J 1/2	J 1/2	J 2/2	J 2/2	J 2/2	J 2/2			N 1/1	
S 1/3	S 1/3	S 1/3	S 1/3	S 1/3	S 1/3	S 1/3	S 1/3	S 2/3	S 2/3	
1	2	3	4	5	6	7	8	9	10	11

8.2 프로그램 Interrupt-Return Code 기억하기 (앞 강의록의 예와 같음)
프로그램 A를 시행(execution)하는 도중에 interrupt P가 호출, P 수행 중에 새로운(더 높은 priority 프로그램) 프로세스 R이 다시 호출되었다. 이러한 인터럽트를 모두 만족시키고 다시 원래의 user program A로 되돌아가서 진행 중이었든 프로그램을 재개할 방법을 구상해 보시오.

예) 집에서 낮잠을 자는 중에, 전화가 왔다. 전화를 받는 중에 밖에서 택배 기사의 호출이 생겼다. 택배기사에게 문을 열어 물품은 받고 나서 해야 할 동작은 무엇인가요? 이 과정을 stack을 이용해서 체계적으로 해결하는 방법을 제시하시오.

8.3 올바른 괄호 짝(parenthesis pair)을 구해보자. 괄호에는 3가지 종류가 있다. 먼저 올바른 괄호 짝을 정의(define)해 보자. 주어진 입력 스트링이 올바른 괄호 짝인지를 계산하는 프로그램을 스택을 이용해서 작성하시오.

a. (( ) [ ] { { ( ) ( ) } })
b. { ([ )] [[ ]] { } }
c. ((( ) [ )) ( ] [ ) { }

8.4 "괄호값" 구하기. 올바른 괄호의 경우 다음의 괄호 값을 구해보시오.

```
( )= 2, [ ]=3, { }=5
```

괄호값을 계산하는 규칙은 다음과 재귀적으로 정의되어 있다.

```
규칙-1.  ( A ) = Weight(A)*2,
규칙-2.  A B = Weight(A)+Weight(B)
```

a. { (( )) [( )] }

b. { } { { ( ) ( ) } }

c. ((( )) [ ] { } { { ( ) ( ) } }) ( ) ( )

d. [[ ) { })

8.5 Stack을 이용해서 permutation 모두 나열하기

n=6인 리스트 [1, 2, 3, 4, 5, 6]의 6! 개의 permutation을 구하시오.

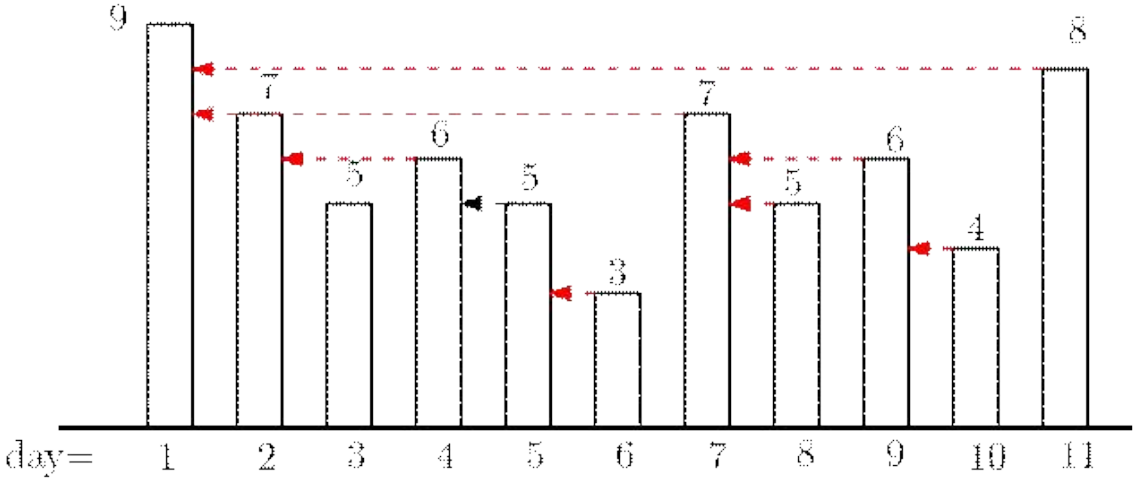
6	5	6			5	1	2	1	6			
5	6	4			1	5	1	2	5			
4	4	5			2	2	5	5	2			
3	3	3			6	6	6	6	1			
2	2	2			4	4	4	4	5			
1	1	1			3	3	3	3	3			

원칙)

```
Stack = ∅
R = { 1, 2, 3, ....  , n }
while ( Stack ≠ ∅ ) {
    if Stack 이 full이 아니면 then
           x = top( )
           x = min { R } ;
           push( x ) ;
           R = R - x ;

    else // Stack이 full임
           print Stack ;
           x = top( );
           if  x의 next(x)가 R에 하나라도 없으면 then
                        z = top( ) ;
                        pop( ) ;
                        R = R ∪ z ;
           else //존재한다면 y = next(x)
                  replace top( ) with y
```

8.6 주가(Stock Price)의 스팬(Span) 계산하기[2)]



다양한 주식 지수가 있다. 예를 들면 KOSPI, KOSPI200, NASDAQ, EURO500 등이 유명한 지수인데 각 지수는 시간별 또는 매일을 최종가격(종가)로 기록된다. 이 주식 시세를 분석하는 다양한 지표가 있는데 그 중 하나는 일일 스팬(daily span)이다. 만일 어떤 i번째 날의 지수가 p_i라고 할 때, 이 날을 기준으로 얼마동안 지수가 떨어지지 않고 상승 유지되었는지의 그 구간이 해당 일자의 스팬 s_i가 된다.

위의 예를 보면 7일의 지수는 $p_7 = 7$인데 이 지수는 2일부터 5일간 그 이상으로 유지되어 왔기 때문에 $s_7 = 5$가 된다. 6일째는 하루 동안 유지되었기 때문에 그 span은 1이 된다. 즉 s_i는 $t_k (k \le i)$에 대하여 연속적으로 $t_k \le t_i$인 날의 수를 말한다. 즉 오늘의 장세가 얼마동안 이어져 왔는지를 보는 것이다. 만일 계속 주식이 오른다면 SPAN의 크기는 매일 증가할 것이다.

day	1	2	3	4	5	6	7	8	9	10	11
s_i	1	1	1	2	1	1	6	1	2	1	10

만일 이 문제를 가장 보편적인 방식으로 해결한다면 2개의 loop으로 해결된다. 날의 수가 만일 N일이라면 시간복잡도(두 배열의 t_i를 비교하는 동작, if $t_i < t_j$, 은 N^2번 걸린다. 이 작업을 stack을 이용해서 N번 만에 해결하는 방법을 생각해보자.

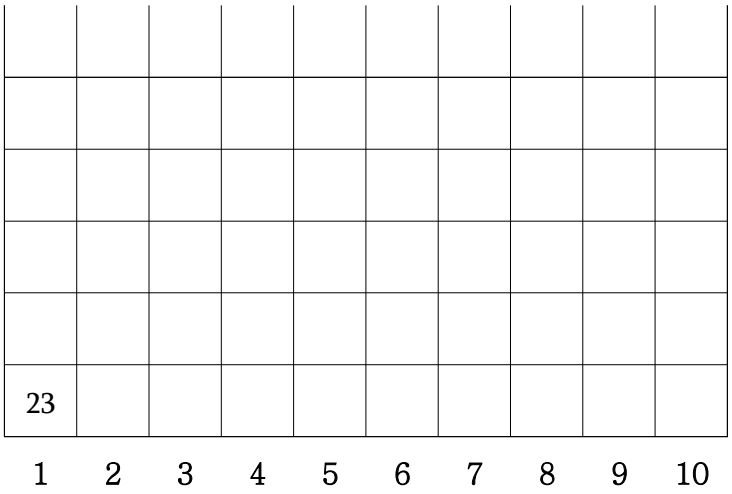
2) 2019년 가을에 추가한 문제

8.7 Next Greater Element(다음 큰 값)[3)]

어떤 배열에 전순서(total order)가 정해져 있는 원소가 저장되어 있다. 우리는 각 원소에 대하여 자신보다 다음에 있는 원소 중에서 가장 가까운 더 큰 원소를 찾으려고 한다. The Next greater Element for an element x is **the first greater** element on the right side of x in array. 이것을 NGE(Next Greater Element)라고 한다. 예를 들어 보자. 만일 NGE가 없는 경우에는 –1로 표시한다.

i	0	1	2	3	4	5	6	7	8
X[i]	23	84	6	56	51	76	40	11	15
NGE	84	-1	56	76	76	-1	-1	15	-1

이 작업을 2개의 Loop을 사용하는 무식한 방법이 아닌 stack을 이용해서 푸는 방법을 생각해보자. 아래 실험용 stack을 이용해서 먼저 손으로 풀어보자.



i	0	1	2	3	4	5	6	7	8
X[i]	23	84	6	56	51	76	40	11	15
NGE	⇑								
NGE									
NGE									
NGE									

3) 2019년 가을에 추가한 문제

8.8 Stack을 이용한 미로 찾아가기: 진행하는 방향은 전체 방향(global)으로 기준으로 해서 다음과 같은 순서로 합니다. = 오른쪽→아래쪽→왼쪽→위쪽 }

	■				X
				■	■
■		■	■		
1,3					
1,2		■	■		■
E 1,1	2,1	3,1	■	5,1	6,1

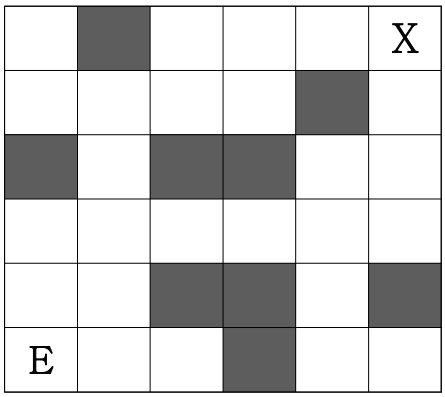
					X
E					

Visited[i][j] = {0,1}

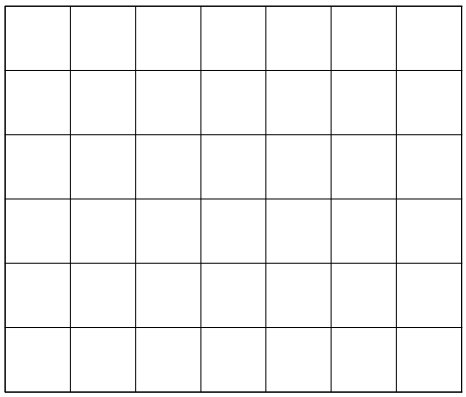
비록 처음 시작하는 방향이 "이상"해도, 당장 보기에는 비효율적이어도, 이 방법은 길(path)이 존재하는 경우에는 반드시 그것을 찾아주는 매우 안정된 방법이다.

1,3														
1,2														
1,1														

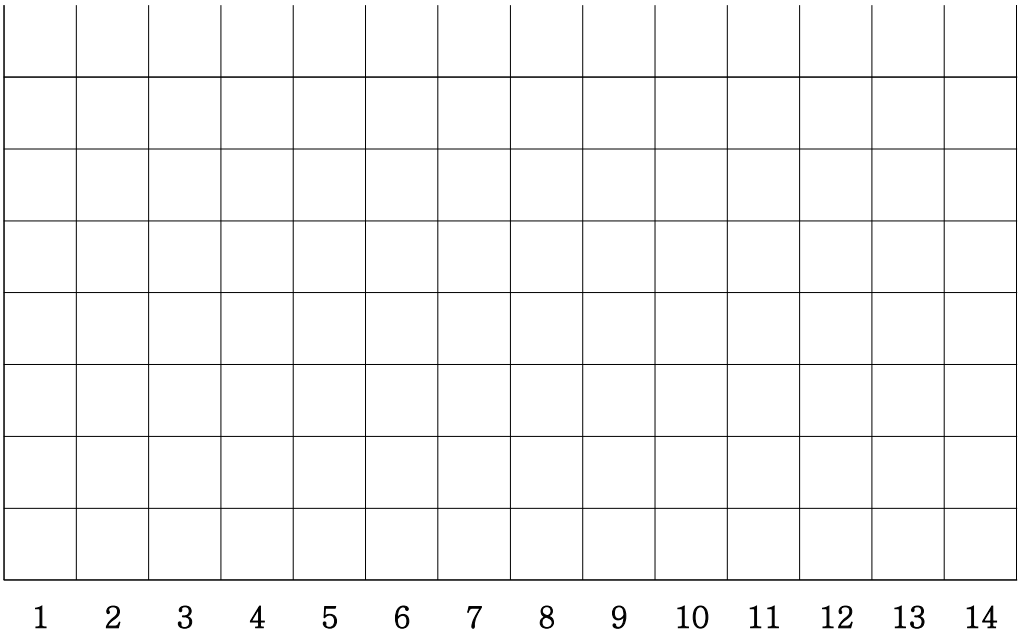
8.4 현재 있는 위치에서 주위를 돌아보는 방문순서는 { 위쪽→아래쪽→오른쪽→왼쪽 } 그 경우 방문 되는 셀(cell)의 순서를 계산해 보시오.

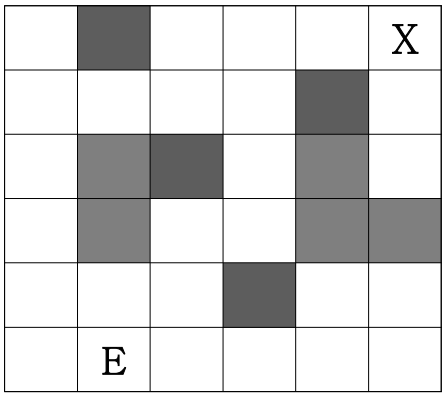


원래 구조

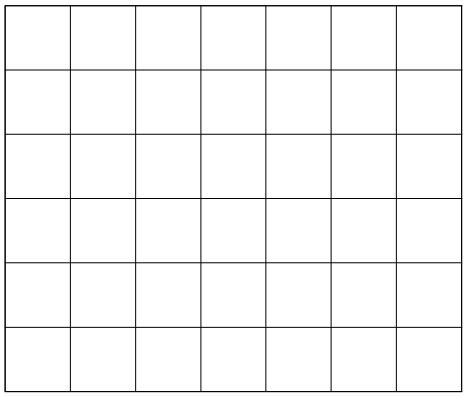


Visited[ ][ ]





원래 구조



Visited[ ][ ]