2021 알고리즘 기말고사

(1) 시험시간 : 6월 16일(수요일) 오전 8시 30분 - 10시 20분
(2) 풀이 중간과정을 충분히 기술하여 채점자에게 도움이 되도록 하세요.
풀이 과정이 충분하지 않거나 최종 답만 적으면 감점됩니다.
(3) 뒷면에 답을 쓰게 되면 작성자가 이 사실을 표시하기 바랍니다.

학번	이름	감독자 확인

문제 (배점)	점수	문제 (배점)	점수
1 (30)		4 (50)	
2 (50)		5 (50)	
3 (30)		6 (30)	
		총합계 (240)	

[1] 어떤 배열 X를 특별한 연산 rev(i)만을 이용해서 정렬하고자 한다. rev(i)는 배열의 시작 X[0]부터 X[i]까지를 뒤집는 연산(reversing)이다.

i	0	1	2	3	4	5	6	7	8	9	10	11
X[i]	61	7	32	30	45	11	8	43	25	15	76	23

만일 처음 배열의 내용이 위와 같다면 rev(i)와 그에 따른 변화된 배열의 모습은 다음과 같다.

i	0	1	2	3	4	5	6	7	8	9	10	11
X[i]	61	7	32	30	45	11	8	43	25	15	76	23
rev(3)	30	32	7	11	45	61	8	43	25	15	76	23
rev(7)	43	8	61	45	11	7	32	30	25	15	76	23
rev(2)	8	43	61	45	11	7	32	30	25	15	76	23
rev(9)	15	25	30	32	7	11	45	61	43	8	76	23

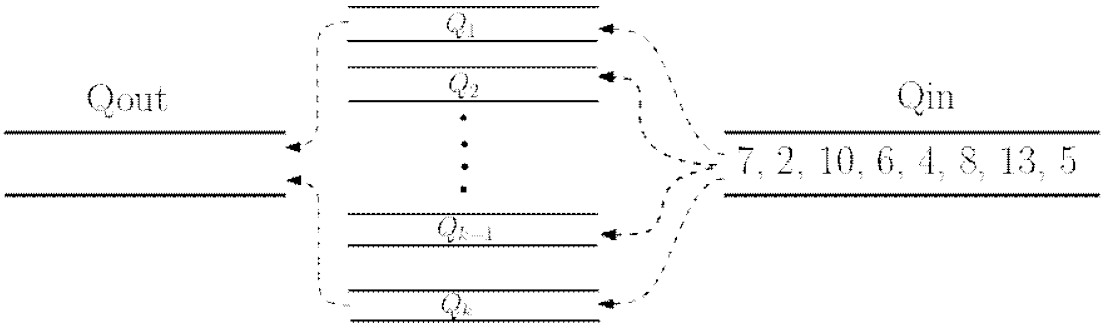
a) 7개 원소가 있는 배열을 대상으로 rev(i)만으로 정렬하는 과정을 보이시오.

STEP		1	2	3	4	5	6	7	8	9	10	11	12	13
적용 연산 rev(i)														
i	X[i]													
0	9													
1	4													
2	3													
3	8													
4	1													
5	6													
6	5													

(b) rev(i) 연산만으로 항상 배열을 정렬할 수 있는 알고리즘을 제시하고 그 최악의 time complexity를 f(N)의 함수로 제시하시오.

[2] Queue sorting.

입력 큐 Qin에 N개의 서로 다른 숫자가 들어있다. Qin의 front에서 나온 숫자는 그 앞에 준비된 중간 k개의 중간(middle) 대기 큐중 하나인 Q_i에 삽입된다. 그리고 이 중간 큐 Q_i에 있는 원소는 최종 출력 큐 Qout로 갈 수 있다. 우리는 이 장치를 통하여 Qin에 들어온 숫자를 Qout에 오름차순으로 정렬하여 저장하고자 한다.



만일 중간 Queue가 1개만 있다면 Qin에 있는 숫자가 미리 정렬되어있지 않는 한 Qout에서 정렬은 불가능하다. 중간 Queue가 N개가 있다면 항상 Qout에서 정렬이 가능하다. 우리는 최소 갯수의 중간 Queue를 사용해서 Qin의 숫자를 모두 Qout에 정렬된 상태로 넣고자 한다. Qin에는 숫자 {1,2,3,4,5,6,7,8,9,10}, 10개의 서로 다른 숫자가 임의의 순서로 저장되어있다고 가정한다.

(a) Qout에서의 정렬에 10개의 중간 큐가 필요한 Qin의 입력 1개의 예를 제시하시오.

(b) 중간 Queue가 반드시 2개 필요한 경우의 Qin 입력 상태를 하나 제시하시오. (즉, 중간 큐의 수가 0 또는 1인 경우에는 정렬할 수 없고 2개부터 가능한 경우)

(c) 중간 Queue가 반드시 3개 필요한 경우의 Qin 입력 상태를 하나 제시하시오. (즉, 중간 큐의 수가 0,1 또는 2인 경우에는 정렬할 수 없고 3개부터 가능한 경우)

(d) 우리는 Qin의 입력 상황을 보고 위 작업을 수행하는 데 필요한 중간 큐의 최소 개수인 k를 구하려고 한다. 이때, k를 구하는 알고리즘과 그 복잡도를 제시하시오.

```
int get_min_Q( int Qin[ ] ) { //Python도 가능
// pseudo코드, 혹은 한글로도 단계별 과정의 설명 가능
```

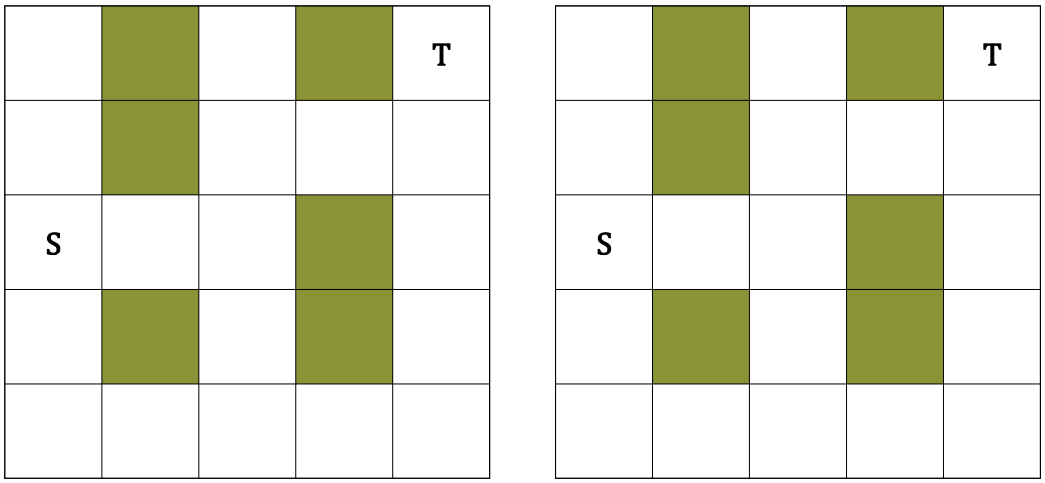
(e) 자신이 위에서 제시한 알고리즘으로 다음 Qin에서 필요한 중간 큐의 최소 개수 k를 제시하시오. (그 계산 과정도 보여야 한다.)

#	**Qin의 초기 상태**	k
1	**4 8 3 7 9 5 10 1 2**	
2	**5 6 7 8 1 3 4 9 10**	
3	**5 1 2 3 4 9 6 8 7**	
4	**3 8 4 7 6 1 10 2 9**	

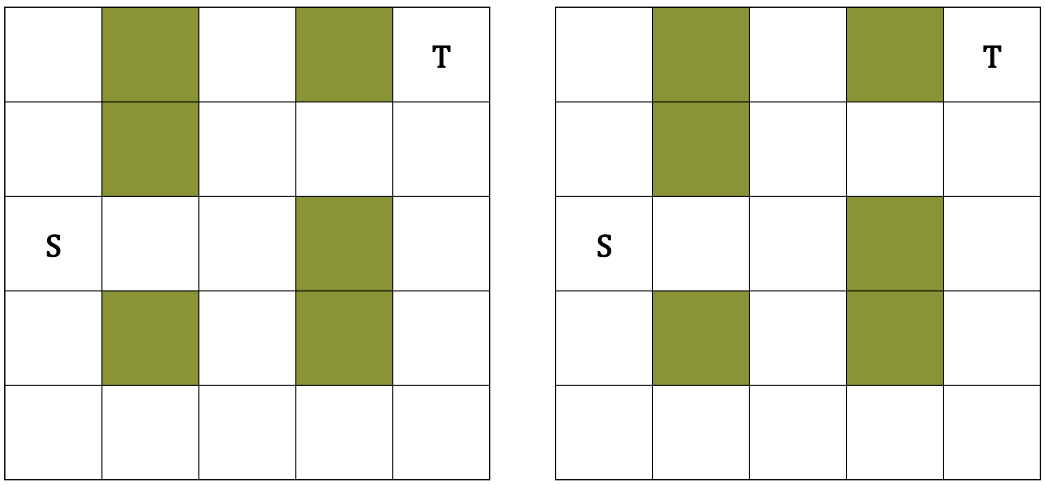
[3] 다음과 같은 장애물이 있는 2차원 공간에서 S에서 출발하여 T까지 도착하는 최단경로를 찾아내고자 한다. 이 작업을 Backtracking과 Branch and Bound로 각각 구하려고 한다. 각 방법에서 구성되는 search tree를 제시하고 아래 grid cell에서 방문되는 순서를 표시하시오.

단, backtracking을 사용할 때 이웃 노드를 살펴보는 순서는 ↑, →, ↓, ←, 이며, Branch and bound를 사용할 경우 **S에서 해당 위치까지의 이동거리 + 그 위치에서 T까지의 직선거리의 합이** 다 작은 쪽을 우선한다. 만일 이 값이 동일할 경우에는 해당 셀의 index (i,j)의 사전식 순서가 더 빠른 쪽을 우선한다. 즉 (2,3)과 (5,1)이 같은 예상거리 값이라고 한다면 (2,3)를 더 우선하여 방문하고 branch를 한다.

단 backtracking이나 branch and bound로 방문할 때 이미 확보한 거리보다 더 긴 경로가 확인되면 바로 cut을 해야 한다. 이런 cut 과정이 발생하면 표시해야 한다.



Backtrack으로 방문할 때의 방문 순서 / Tree



Branch and Bound로 방문할 때의 방문 순서 / Tree

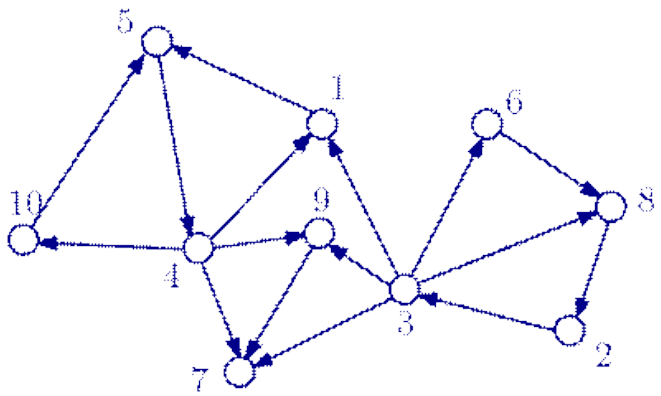
[4] 다음 방향 그래프(directed graph)에서 아래 질문에 답을 하시오.

a) Strongly Connected Digraph의 정의를 기술하시오.

b) |V|=10개인 Digraph 중에서 가장 적은 간선 개수를 가진 strongly connected digraph의 예를 제시하시오.

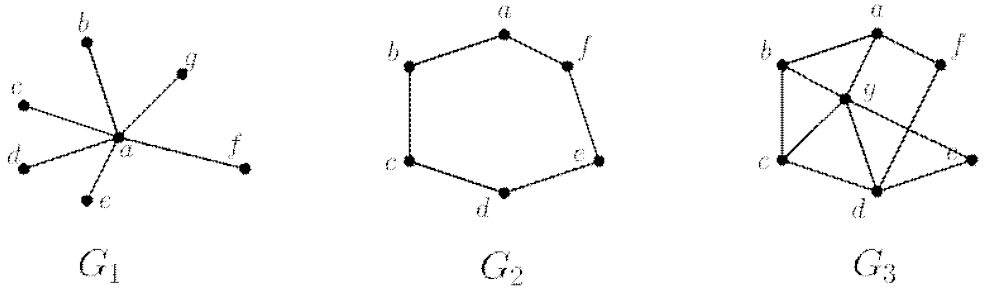
c) 어떤 방향 그래프 G에서 Strongly Connected Component가 무엇인지 그 정의를 기술하시오.

d) 아래 주어진 방향 그래프에서 Strongly Connected Component를 찾는 과정을 강의에서 설명한 *Kosaraju* 알고리즘을 사용하여 보이시오.



V[i]	1	2	3	4	5	6	7	8	9	10
DFS #										
finished #										

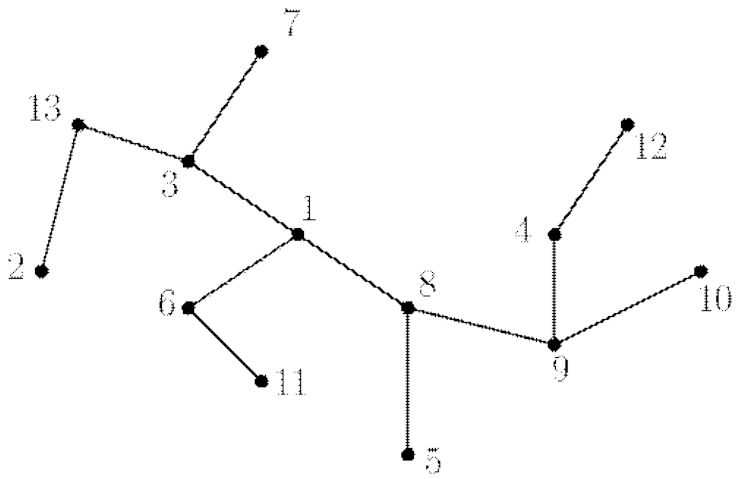
[5] 어떤 정점 v를 선택하면 그와 연결된 모든 인접 간선은 v로 cover 된다고 말한다. 교차로가 정점, 그 교차로와 인접한 도로를 간선라고 했을 때 특정 교차로(정점)에 회전하는 CCTV를 설치하면 그와 인접한 모든 도로(간선)는 해당 정점의 CCTV로 감시(covering)됨과 같다고 생각하면 쉽다. 우리는 주어진 그래프의 모든 간선을 cover하는 정점의 집합 중에서 그 크기가 최소인 Minimum Vertex Cover(MVC)를 구하려고 한다.



G_1 G_2 G_3

위 그래프에서 G1의 MVC는 {a}이다. 즉, a를 선택하면 그와 연결된 6개의 모든 간선이 cover된다. 만일 G2와 같이 길이 6인 cycle이라고 한다면 MVC는 크기는 3이며, 그 집합은 {a,e,c} 혹은 {b,d,f}가 됩니다.

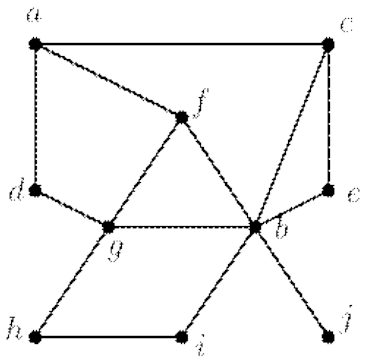
a) 아래 Tree에서 MVC를 구하여 그림에 표시하시오.



b) 일반적인 그래프의 경우 이 문제는 시간복잡도가 지수인 NP-complete 문제임이 밝혀져 있다. 우리는 이 문제를 위하여 Greedy 전략을 사용하는 알고리즘을 사용하려 한다. 이 전략은 가장 많은 간선을 covering 하는 기준으로 정점을 선택하여 모든 간선이 cover될 때까지 수행하는 것이다. 만일 새롭게 cover할 수 있는 간선수가 같을 경우에는 정점을 표시한 알파벳의 사전식 순서가 빠른 것을 기준으로 선택한다.

이 그리디 알고리즘을 이용하여 G3의 MVC를 구해보자. 이 경우 제일 처음 선택하는 정점은 5개의 간선을 cover하는 {g}가 된다. 그 다음에는 추가 3개의 간선을 새롭게 cover할 수 있는 {d}를 선택한다. 그리고 아직 남아있는 uncovered 간선 {(c,d), (b,a), (a,f)}를 위해서는 2개까지 cover할 수 있는 정점이 {a,b}인데 알파벳 순서가 빠른 {a}를 선택한다. 이제 남아있는 것은 하나의 간선 (b,c)이고 이것을 cover할 수 있는 것은 {b,c}이므로 이중에서 빠른 {b}를 선택한다. 따라서 그리디 알고리즘에 의하면 구해지는 MVC는 {g,d,a,b}가 되고 MVC의 크기는 4가 된다.

이 그리디 알고리즘으로 아래 그래프에서 MVC를 구하는 과정을 보이시오.



c) 그리디 알고리즘으로 MVC를 구했을 때 최적이 되지 못하는 그래프의 예를 제시하시오. 단, 그래프는 최소 4개 이상의 정점으로 구성되어야 한다. (그리디로 구한 MVC보다 더 적은 크기의 MVC가 있는 그래프를 제시해야 한다.)

[6] 9개의 동전 {c1, c2, ... , c9} 중에 약간 가벼운 가짜 동전이 4개 존재한다. 이것을 찾아내기 위하여 양팔 저울을 사용할 때 advesary를 고려한 최악의 경우를 제시하고 그 최악의 경우 몇 번 저울을 사용해야 하는지를 나타내는 decision tree를 그리시오.