JooHyun – Lee (comkiwer)

TS트리탐색

Hancom Education Co. Ltd.

Problem

Problem

TS트리탐색

트리라는 이름의 도시에 경찰이 **한 명** 있다.

경찰은 트리 도시를 돌아다니면서, 도시 내에 발생한 사건들을 해결한다.

경찰은 정해진 규칙에 따라서만 행동한다.

따라서 충분한 정보가 있으면, 특정 시각에 경찰이 어느 곳에 있는 지를 파악할 수 있다.

아래의 설명들을 참고하여, 트리 도시를 지키는 경찰의 위치를 파악하는 프로그램을 작성하라.

Problem

TS트리탐색

[트리 도시]

트리 도시에는 N개의 마을과 N-1개의 도로가 있다.

각 마을은 0부터 N-1까지의 고유 번호를 가진다.

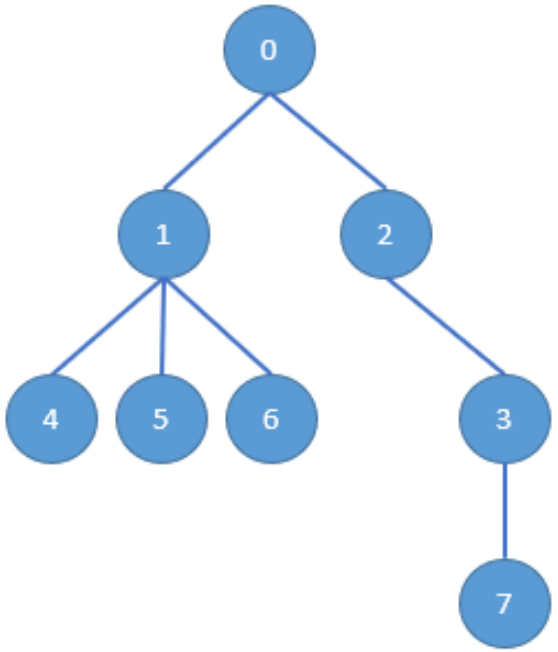
0번 마을 제외한 모든 마을들은 하나의 부모 마을을 가지며, 부모 마을과 도로로 연결되어 있다.

도로의 길이는 모두 동일하다.

도로는 양방향으로 모두 통행 가능하다.

일방통행 도로는 존재하지 않는다.

두 마을 사이의 최단 경로는 항상 존재하며, 유일하다.



[그림. 1]

Problem

[타임스탬프]

`timeStamp` 시각은 트리 도시가 생긴 시점부터 `timeStamp` 시간이 흐른 시점을 의미한다.
`timeStamp` 시각과 `timeStamp+1` 시각 사이의 시간은 항상 `1`시간이다.

[사건]

트리 도시에는 사건이 수시로 발생한다.

사건의 정보에는 발생한 시각, 고유 번호, 발생한 마을의 번호,
처리의 우선순위 등이 있다. 사건 발생은 취소될 수 있다.

목표 사건이란 남아있는 사건 중 가장 우선순위가 높은 사건이다. 남아있는 사건이란 아직 해결 혹은 취소되지 않은 사건을 의미한다.

가장 우선순위가 높은 사건이 여러 개일 경우, 그 중에서 가장 이른 시각에 발생한 사건이 목표 사건이 된다.

Problem

[트리를 지키는 경찰]

트리 도시에는 경찰이 한 명 있다.

0 시각에 경찰은 0번 마을에 위치해 있다.

경찰이 하나의 도로를 건너는 데 걸리는 시간은 항상 1시간이다.

목표 사건이 발생한 마을에 경찰이 1시간 동안 체류하면, 목표 사건이 해결된다.

단, 해당 마을에서 발생한 다른 사건들까지 같이 해결되는 것은 아니다.

목표 사건만 해결된다.

[트리를 지키는 경찰] : 계속

경찰은 timeStamp 시각부터 1시간 동안, 아래의 순서에 맞게 행동한다.

1) timeStamp 시각에 발생 혹은 취소된 사건에 대한 정보를 알아낸다.
이에 소모되는 시간은 없으며, 경찰은 알아낸 정보를 잊어버리지 않는다.

2) 목표 사건을 찾는다. 이에 소모되는 시간은 없다.

3-1) 목표 사건이 없을 경우, timeStamp+1 시각까지 현재 위치에 대기한다.

3-2) 목표 사건이 있고, 목표 사건의 발생 위치와 현재 위치가 동일할 경우,
timeStamp+1 시각까지 목표 사건을 해결한다.

3-3) 목표 사건이 있고, 목표 사건의 발생 위치와 현재 위치가 동일하지 않을 경우,
timeStamp+1 시각까지 목표 사건의 발생 위치를 향해 최단 경로로 이동한다.

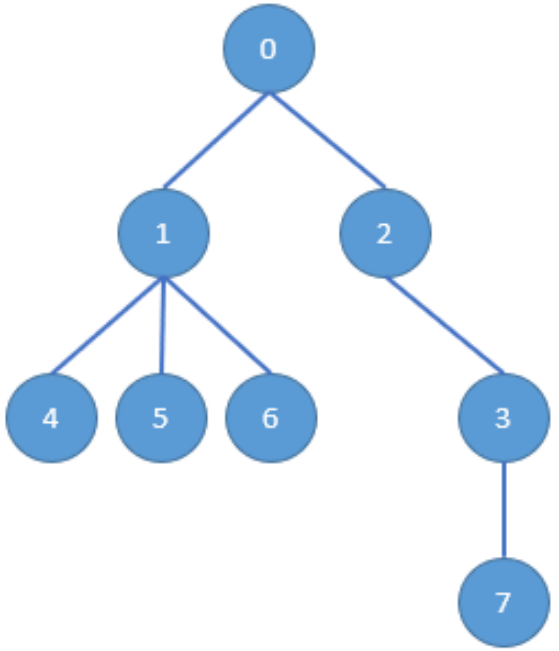
Problem

매 시각, 목표 사건을 새로이 찾는다는 점에 유의하라.

1시간 후의 경찰의 위치를 빠르게 파악할 수 있도록 해놓으면, 문제 풀이에 도움이 될 수 있다. ([표. 1] 참조)

[그림. 1]에서, **경찰의 위치**가 0번 마을이고,
목표 사건의 발생 위치가 7번 마을이면,
1시간 후의 경찰의 위치는 2번 마을이다.

[그림. 1]에서, **경찰의 위치**가 2번 마을이고,
목표 사건의 발생 위치가 7번 마을이면,
1시간 후의 경찰의 위치는 3번 마을이다.



[그림. 1]

Problem

TS트리탐색

1시간 후의 경찰의 위치 (그림. 1)		목표 사건의 발생위치							
		0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
경찰의 위치	*0*	0	1	2	2	1	1	1	2
	1	0	1	0	0	4	5	6	0
	2	0	0	2	3	0	0	0	3
	3	2	2	2	3	2	2	2	7
	4	1	1	1	1	4	1	1	1
	5	1	1	1	1	1	5	1	1
	6	1	1	1	1	1	1	6	1
	7	3	3	3	3	3	3	3	7

[표. 1]

마을의 개수는 350개 이하이며,

함수 호출 시마다 주어지는 `timeStamp`의 값은 5,000,000 이하이다.

아래 `API` 설명을 참조하여 각 함수를 구현하라.

Problem

TS트리탐색

1. `void init(int N, int parent[]);`

   각 테스트 케이스의 처음에 호출된다.

   N은 마을의 개수이다.

   parent는 부모 마을에 대한 정보를 담은 1차원 배열이다.

   i번 마을의 부모 마을은 parent[i]번 마을이다. (1 ≤ i < N)

   parent[0]은 항상 -1이며, 이는 0번 마을의 부모 마을이 없음을 의미한다.

   parent[i]는 항상 i보다 작은 값이다.

   테스트 케이스의 시작 시, 트리 도시에 존재하는 사건은 없다.

   init()이 호출된 시점은 0 시각이며, 이 때, 경찰은 0번 마을에 있다.

   Parameters

   - N : 마을의 개수 (2 ≤ N ≤ 350)
   - parent : 부모 마을에 대한 정보 (-1 ≤ parent[i] ≤ i-1)

Problem

2. `void occur(int timeStamp, int caseID, int townNum, int prior);`

   timeStamp 시각에 townNum번 마을에서 caseID번 사건이 발생한다.

   caseID번 사건의 우선순위는 prior이다.

   첫 사건 발생 시 caseID는 0이고, 이후 occur 함수가 호출될 때마다 1씩 증가한다.

   Parameters

   - timeStamp : 사건이 발생한 시각 (0 ≤ timeStamp ≤ 5,000,000)
   - caseID : 발생한 사건의 고유번호 (0 ≤ caseID ≤ 99,999)
   - townNum : 사건이 발생한 마을의 번호 (0 ≤ townNum ≤ N-1)
   - prior : 사건의 처리 우선순위 (0 ≤ prior ≤ 99,999)

Problem

3. `void cancel(int timeStamp, int caseID);`

   timeStamp 시각에 caseID번 사건을 취소한다.

   caseID번 사건이 발생한 적 있음이 보장된다. 단, 이미 해결되었거나, 취소된 사건일 수는 있다.

   Parameters

   - timeStamp : 사건이 취소된 시각 (0 ≤ timeStamp ≤ 5,000,000)
   - caseID : 취소된 사건의 고유번호 (0 ≤ caseID ≤ 99,999)

Problem

4. `int position(int timeStamp);`

`timeStamp` **시각에 경찰이 위치한 마을의 번호를 반환한다.**

Parameters

`timeStamp` : **시각** (0 ≤ timeStamp ≤ 5,000,000)

Return

`timeStamp` **시각에 경찰이 위치한 마을의 번호**

Problem

TS트리탐색

[제약사항]

1. 각 테스트 케이스 시작 시 `init()` 함수가 호출된다.
2. 각 테스트 케이스에서 `occur()` 함수의 호출 횟수는 `100,000` 이하이다.
3. 각 테스트 케이스에서 `cancel()` 함수의 호출 횟수는 `50,000` 이하이다.
4. 각 테스트 케이스에서 `position()` 함수의 호출 횟수는 `100,000` 이하이다.
5. 각 테스트 케이스에서 함수 호출 시마다 주어지는 `mTimeStamp`의 값은 항상 증가한다.

Problem analysis

Problem analysis : 예제

1. init (N : 8, parent[]: -1, 0, 0, 2, 1, 1, 1, 3)

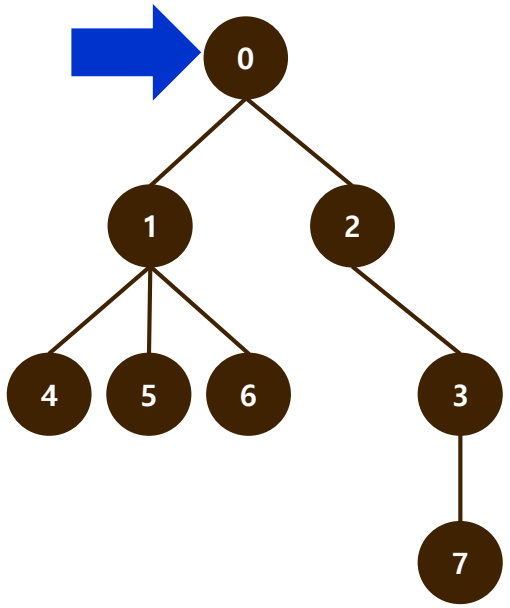
copPos : 0

curTime : 0

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

N명의 각각에 대한 부모 정보가 주어진다.
이를 이용하여 트리 adj[][]를 구성하고
s를 시작점으로 e를 도착점으로 출발 할 때,
1시간 후에 도착하는 노드 정보 nextTown[][]를 구성할 수 있다.

초기 경찰의 위치화 현재시각을 0으로 초기화 한다.

Problem analysis

4. position (timestamp : 0) ans = 0

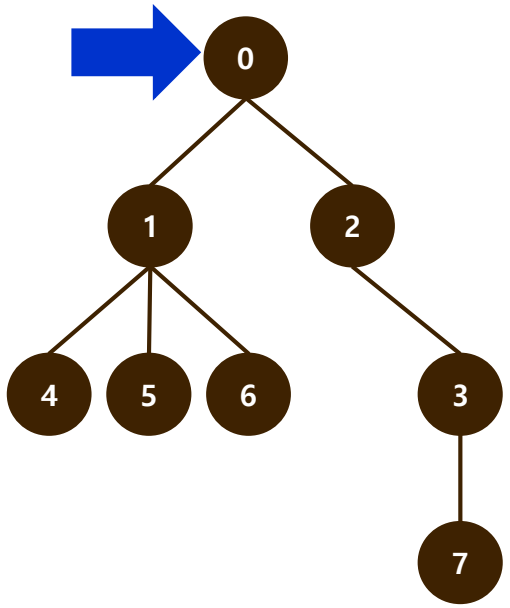
copPos : 0

curTime : 0

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 0에서 경찰의 위치를 묻고 있다.

Problem analysis

2. occur(timestamp : 4, caseID : 0, townNum : 4, prior : 1)

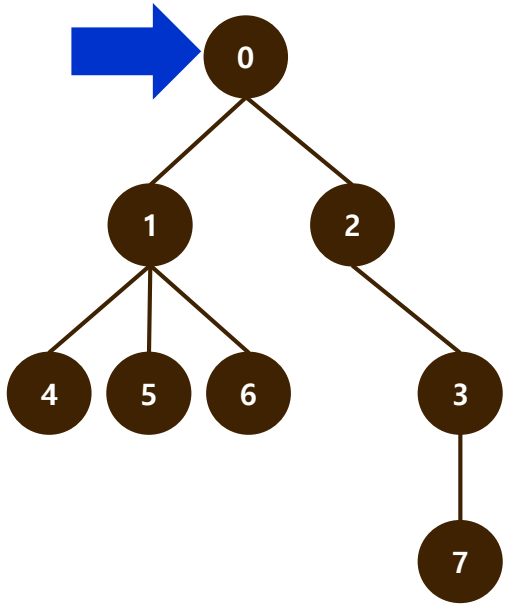
copPos : 0

curTime : 0 -> 4

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
4	0	4	1

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

0번 마을에 위치한 경찰은 시각 4까지 대기한다.

curTime 을 4로 수정한다.
이제 caseID = 0번 사건을 우선순위 대기 목록에 추가한다.

Problem analysis

TS트리탐색

4. position (timestamp : 5) ans = 1

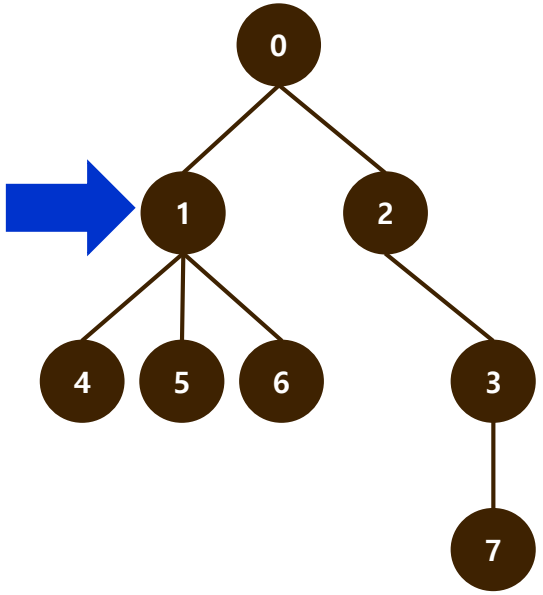
copPos : 0 -> 1

curTime : 4 -> 5

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
4	0	4	1

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

0번 마을에서 4번 마을로 향하는 경우 1시간 후에 도착하는 마을은 1번 마을이다.

시각 5에 1번 마을에 도착한다.

Problem analysis

TS트리탐색

4. position (timestamp : 6) ans = 4

copPos : 1 -> 4

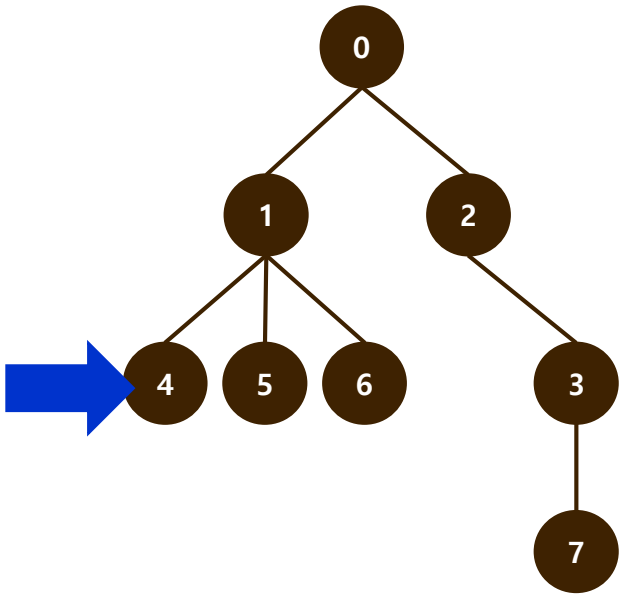
curTime : 5 -> 6

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
4	0	4	1

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

1번 마을에서 4번 마을로 향하는 경우 1시간 후에 도착하는 마을은 4번 마을이다.

시각 6에 4번 마을에 도착하여 0번 사건 해결을 시작한다.

Problem analysis

TS트리탐색

4. position (timestamp : 7) ans = 4

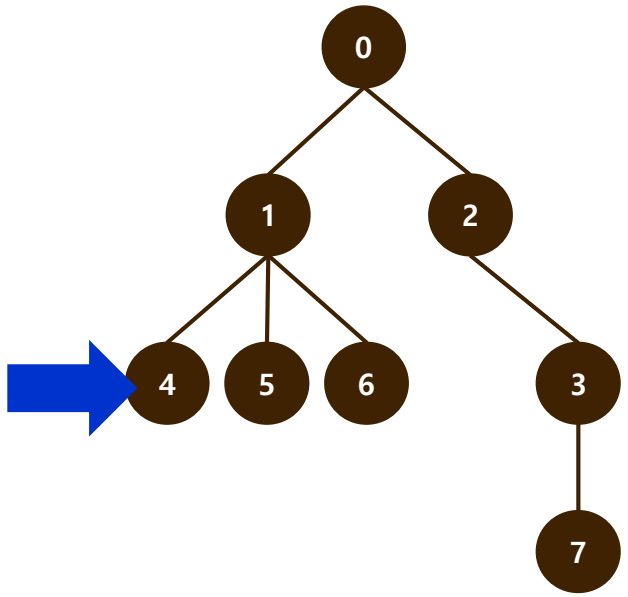
copPos : 4

curTime : 6 -> 7

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
4	0	4	1

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 7에 4번 사건이 해결이 끝나고 다음 사건이 없으므로 대기한다.

Problem analysis

2. occur(timestamp : 8, caseID : 1, townNum : 7, prior : 2)

copPos : 4

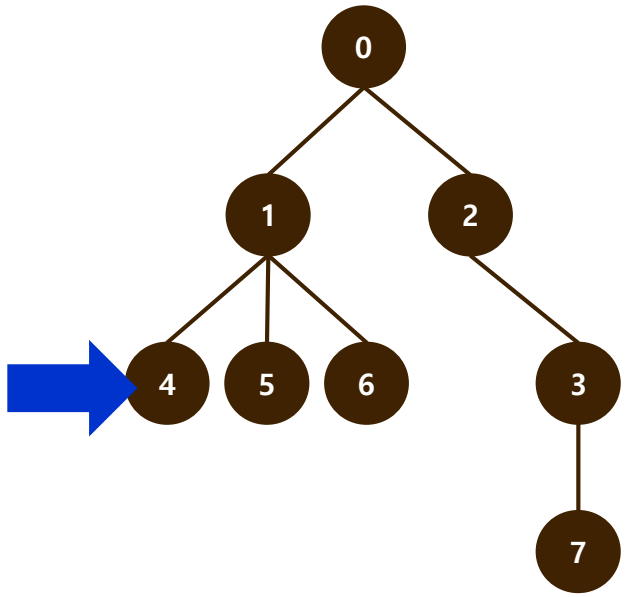
curTime : 7 -> 8

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 8에 1번 사건이 주어진다.
경찰은 4번 마을에서 시각 8까지 대기한다.

시각 8에 1번 사건을 우선순위 큐에 등록한다.

Problem analysis

TS트리탐색

2. occur(timestamp : 9, caseID : 2, townNum : 6, prior : 3)

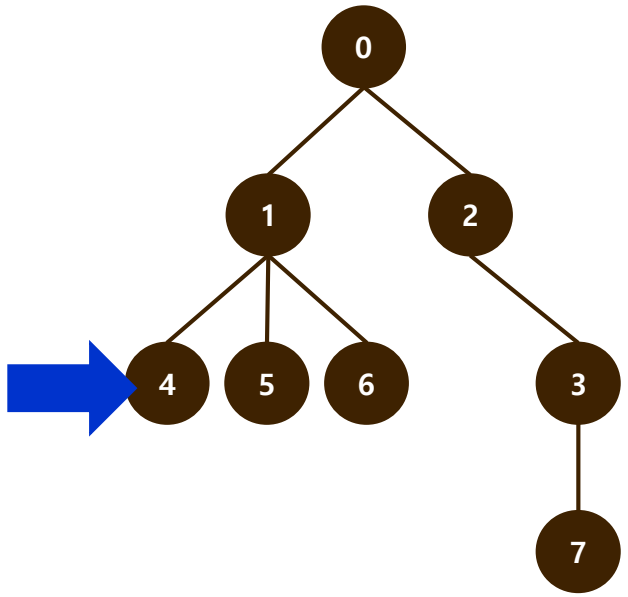
copPos : 4

curTime : 8

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 9에 2번 사건이 주어진다.

시각 8에 4번 마을에서 대기중인 경찰은 7번 마을을 향하여 이동을 시작한다.

Problem analysis

2. occur(timestamp : 9, caseID : 2, townNum : 6, prior : 3)

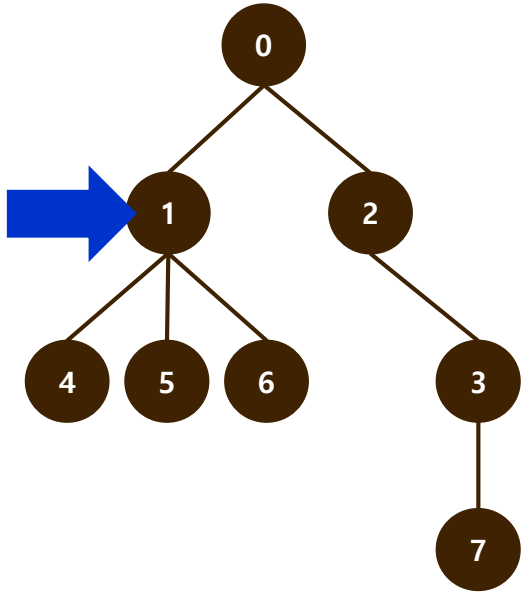
copPos : 4 -> 1

curTime : 8 -> 9

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
9	2	6	3
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 9에 2번 사건이 주어진다.

시각 9에 1번 마을에 도착한다.
이때 경찰은 우선순위가 더 높은 2번 사건을 맡게 된다.

Problem analysis

TS트리탐색

4. position (timestamp : 13) ans = 0

copPos : 1 -> 6

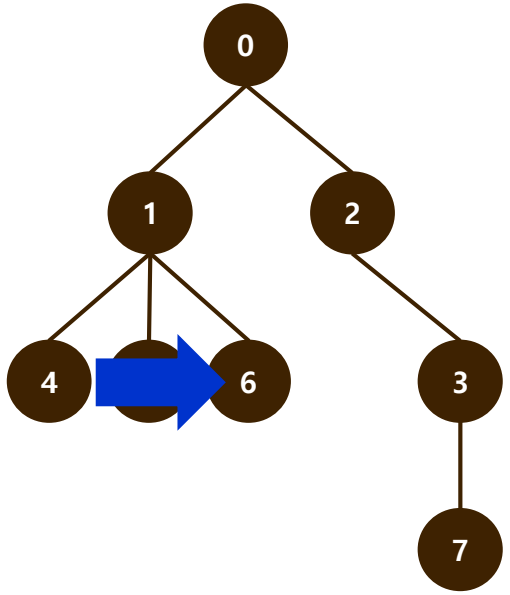
curTime : 9 -> 10

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
9	2	6	3
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 13에 경찰의 위치를 묻고 있다.

경찰은 현재 2번 사건 처리를 위하여 이동중이며 현재 시각 10에 6번으로 이동한다.

Problem analysis

TS트리탐색

4. position (timestamp : 13) ans = 0

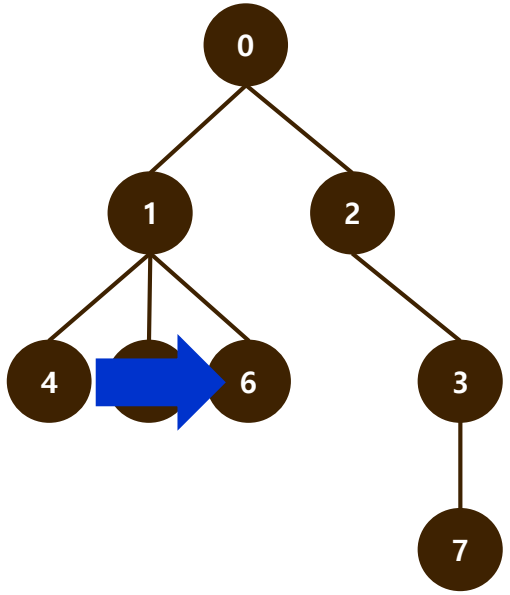
copPos : 6 -> 6

curTime : 10 -> 11

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
9	2	6	3
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 13에 경찰의 위치를 묻고 있다.

2번 사건 처리를 위하여 6번 마을에서 1시간 머문다.

Problem analysis

TS트리탐색

4. position (timestamp : 13) ans = 0

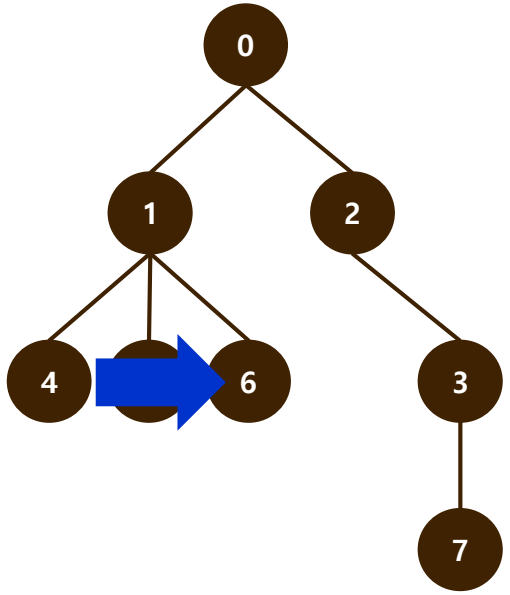
copPos : 6

curTime : 11

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 13에 경찰의 위치를 묻고 있다.

이제 1번 사건 처리를 위하여 이동을 시작한다.

Problem analysis

TS트리탐색

4. position (timestamp : 13) ans = 0

copPos : 6 -> 1

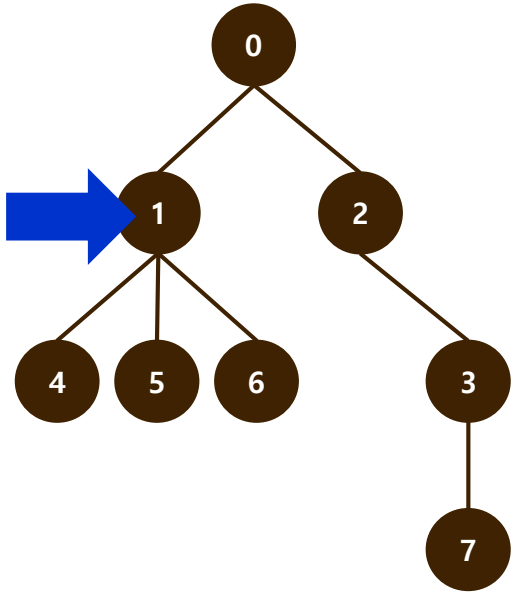
curTime : 11 -> 12

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 13에 경찰의 위치를 묻고 있다.

경찰은 시각 12에 1번 마을에 도착한다.

Problem analysis

TS트리탐색

4. position (timestamp : 13) ans = 0

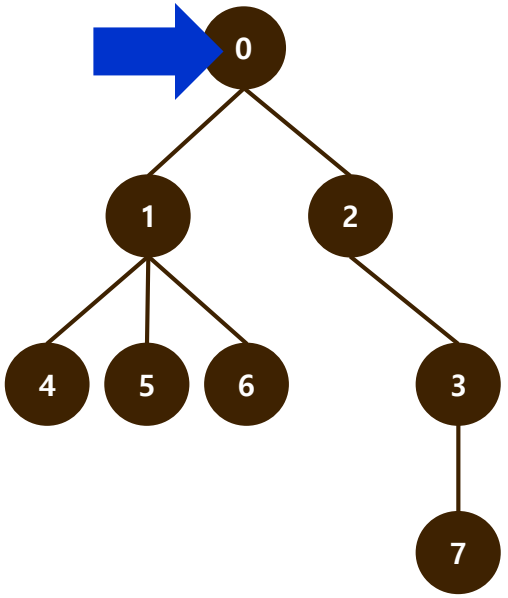
copPos : 1 -> 0

curTime : 12 -> 13

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 13에 경찰의 위치를 묻고 있다.

경찰은 시각 13에 0번 마을에 도착한다.

Problem analysis

2. occur(timestamp : 18, caseID : 3, townNum : 2, prior : 4)

copPos : 1 -> 0

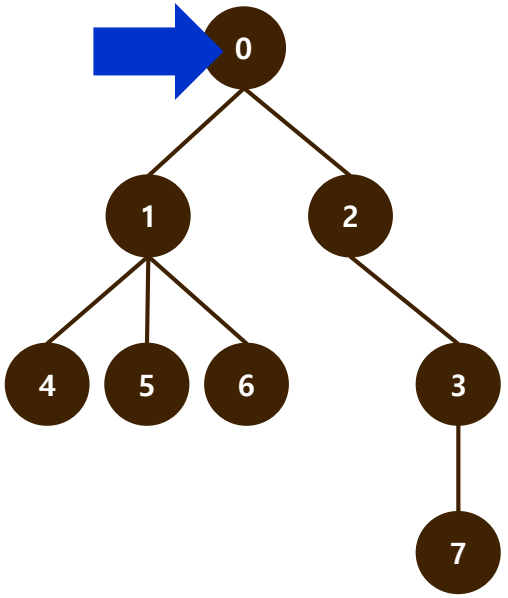
curTime : 12 -> 13

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 18에 3번 사건이 발생한다.

먼저 시각 18까지 우선순위가 가장 높은 1번 사건 처리를 수행한다.

Problem analysis

2. occur(timestamp : 18, caseID : 3, townNum : 2, prior : 4)

copPos : 0 -> 2

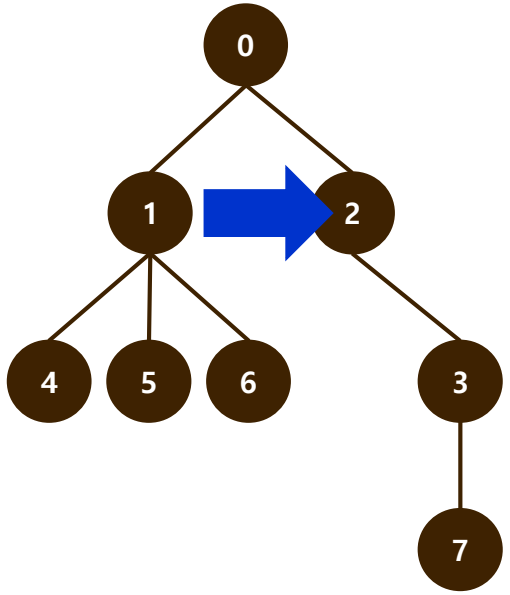
curTime : 13 -> 14

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 18에 3번 사건이 발생한다.

시각 14에 2번 마을에 도착한다.

Problem analysis

2. occur(timestamp : 18, caseID : 3, townNum : 2, prior : 4)

copPos : 2 -> 3

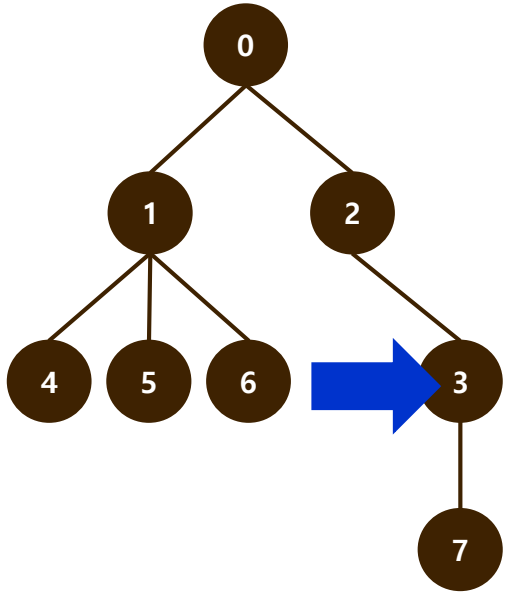
curTime : 14 -> 15

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 18에 3번 사건이 발생한다.

시각 15에 3번 마을에 도착한다.

Problem analysis

2. occur(timestamp : 18, caseID : 3, townNum : 2, prior : 4)

copPos : 3 -> 7

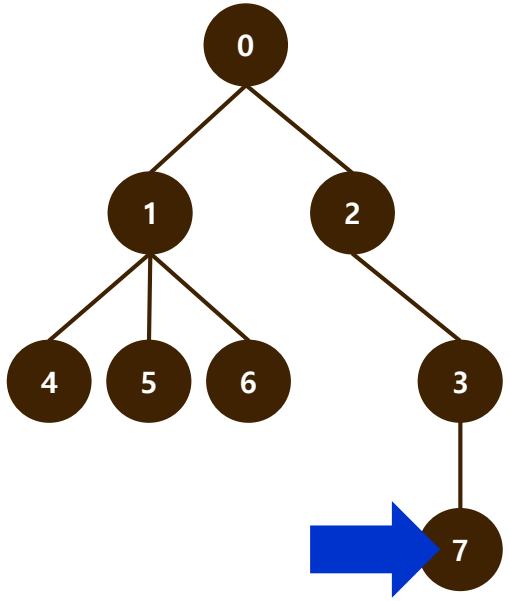
curTime : 15 -> 16

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 18에 3번 사건이 발생한다.

시각 16에 7번 마을에 도착한 후, 사건 처리를 시작한다.

Problem analysis

2. occur(timestamp : 18, caseID : 3, townNum : 2, prior : 4)

copPos : 7 -> 7

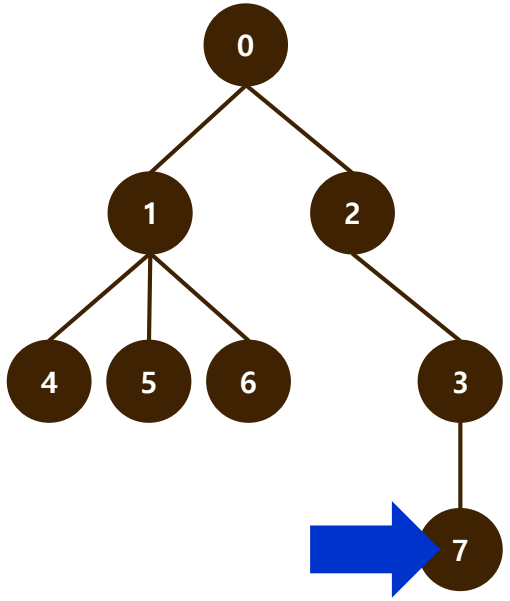
curTime : 16 -> 17

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
8	1	7	2

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 18에 3번 사건이 발생한다.

시각 17에 1번 사건 처리를 완료하고 대기를 시작한다.

Problem analysis

2. occur(timestamp : 18, caseID : 3, townNum : 2, prior : 4)

copPos : 7 -> 7

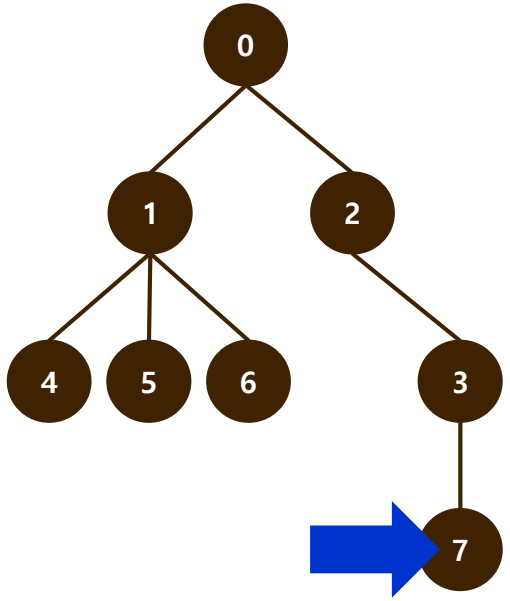
curTime : 17 -> 18

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
18	3	2	4

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 18에 3번 사건이 발생한다.

시각 18에 3번 사건을 대기 목록에 추가한다.

Problem analysis

2. occur(timestamp : 20, caseID : 4, townNum : 5, prior : 6)

copPos : 7 -> 3

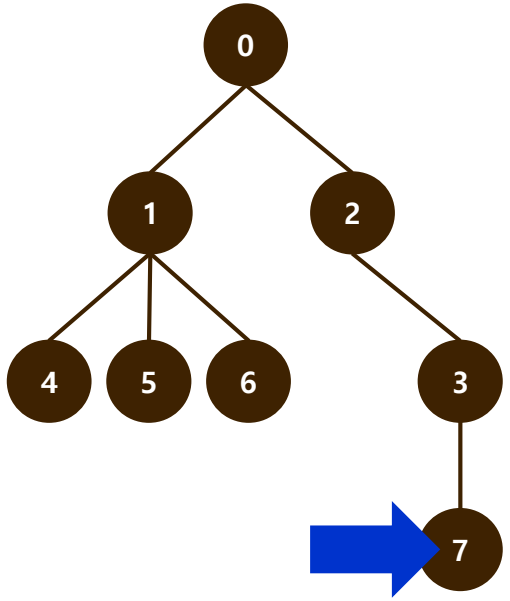
curTime : 18 -> 19

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
18	3	2	4

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 20에 4번 사건이 발생한다.

시각 18에 3번 사건처리를 위하여 이동을 시작한다.

Problem analysis

2. occur(timestamp : 20, caseID : 4, townNum : 5, prior : 6)

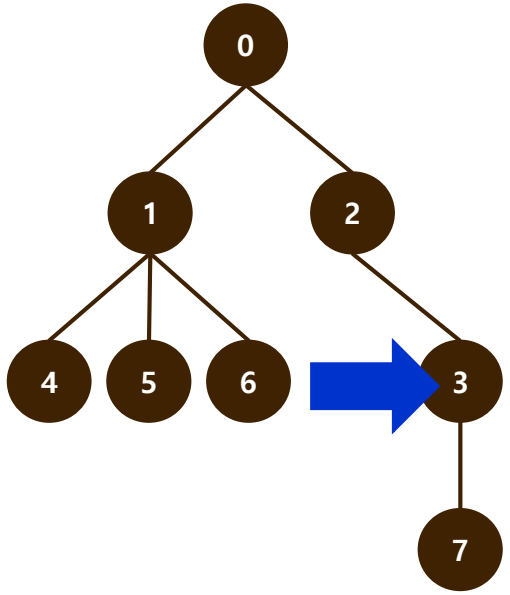
copPos : 7 -> 3

curTime : 18 -> 19

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
18	3	2	4

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 20에 4번 사건이 발생한다.

시각 19에 3번 마을에 도착한다.
사건처리를 위하여 이동을 계속한다.
다음 목적지는 2번 마을이다.

Problem analysis

2. occur(timestamp : 20, caseID : 4, townNum : 5, prior : 6)

copPos : 3 -> 2

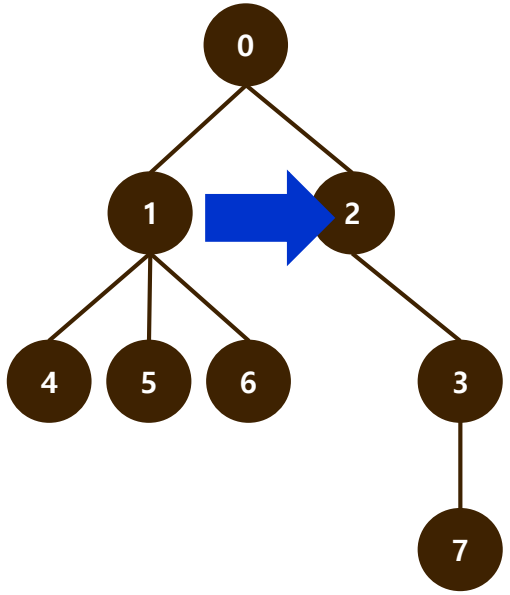
curTime : 19 -> 20

Priority_queue

: prior(desc), tick(asc)

tick	id	town	prior
18	3	2	4

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 20에 4번 사건이 발생한다.

시각 20에 2번 마을에 도착한다.

Problem analysis

2. occur(timestamp : 20, caseID : 4, townNum : 5, prior : 6)

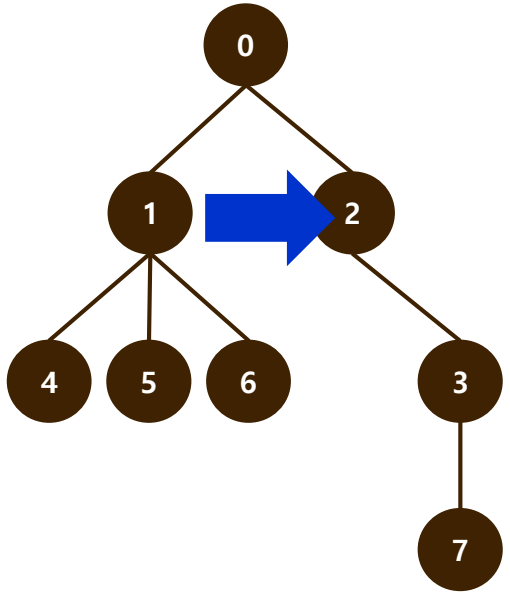
copPos : 3 -> 2

curTime : 19 -> 20

Priority_queue
: prior(desc), tick(asc)

tick	id	town	prior
20	4	5	6
18	3	2	4

adj[][]



nextTown[][]

	0	1	2	3	4	5	6	7
0	0	1	2	2	1	1	1	2
1	0	1	0	0	4	5	6	0
2	0	0	2	3	0	0	0	3
3	2	2	2	3	2	2	2	7
4	1	1	1	1	4	1	1	1
5	1	1	1	1	1	5	1	1
6	1	1	1	1	1	1	6	1
7	3	3	3	3	3	3	3	7

nextTown[s][e] :
s에서 e로 출발한지 한시간 후 도착하는 마을

시각 20에 4번 사건이 발생한다.

시각 20 에 4번 사건을 대기 목록에 추가한다.
3번 사건은 목적지에 도착하였지만 우선순위에 밀려 사건 처리를 시작하지 못한다.

Problem analysis : 제약 조건 & API

TS트리탐색

- 우선순위에 맞게 시간에 따른 처리가 필요한 문제이다.
- 목적지에 도착하였다고 문제가 해결되는 것이 아니다.
  사건을 해결하는데 1시간이 소요된다.
  따라서 목적지에 도착한 시점에 우선순위가 더 높은 사건이 발생한다면
  기존 사건 처리는 우선순위에서 밀려난다.

- 마을의 개수가 350개이고 시간의 흐름도 5백만이다.
- 큰 수도 문자열도 등장하지 않는다.
- 별색(붉은) 글씨로 문제해결전략 가이드를 주고 있다.
  이를 잘 이해하고 구현할 수 있다면 비교적 어렵지 않는 문제가 될 수 있다.

Solution sketch

Solution sketch

- 문제에서 요구하는 각 개체들을 적절한 자료구조로 표현할 수 있다면 많은 부분이 해결된다. 먼저 필요한 자료를 정리해 보자.

- 발생한 사건은 prior의 내림차순, 발생 시각의 오름차순으로 사건처리의 우선순위가 결정된다.
- 따라서 사건은 다음과 같은 클래스로 정의될 수 있다.

```
struct Event {
    int tick;    // 사건이 일어난 시각
    int id;      // 사건 id
    int town;    // 목적도시
    int prior;   // 처리 우선순위
    bool operator<(const Event&t)const {  // 1. 처리우선순위 desc, 2. 일어난 시각 asc
        return (prior != t.prior ? prior < t.prior : tick > t.tick);
    }
};
```

Solution sketch

- 경찰의 상태는 다음 두 가지 속성으로 결정될 수 있다.
  1. 경찰의 위치(어느 마을에 있는가?) : copPos
  2. 경찰 위치를 확인한 최종 시각 : curTime

- 초기 트리 정보가 주어질 때, 구현 가이드라인과 같이
  1시간 후 경찰의 위치를 미리 구해 놓을 수 있다.
  이 표를 nextTown[] [] 이라 하자.

- 이제 앞서 살펴본 예제 분석과 자료구조를 이용하여
  각 api함수별 할 일을 정리해 본다.

Solution sketch

1. `void init(int N, int parent[]);`

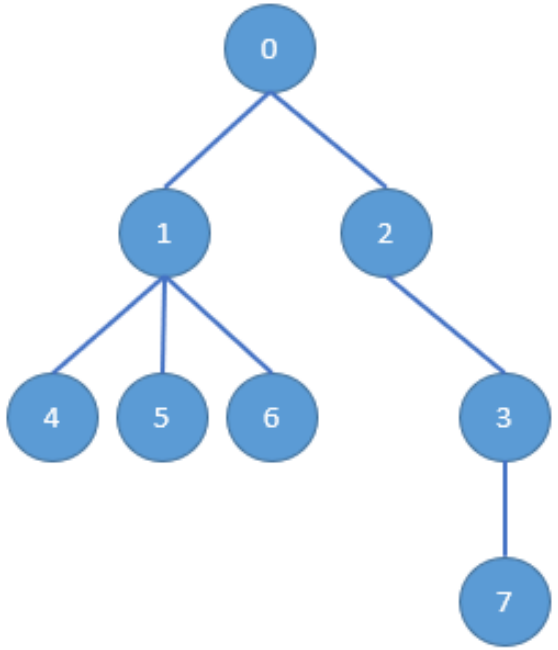
: N(2 <= N <= 350)개의 도시와 도로 정보가 주어진다.

: parent[]를 이용하여 인접 배열 adj[][]를 만들 수 있다.

: adj[][]를 이용하여 dfs 또는 bfs를 이용하여 nextTown[][] 표를 작성할 수 있다.

: nextTown[s][e] : s는 출발 마을, e는 도착 마을일 때,
한 시간 후 경찰이 위치하는 마을

: 경찰의 위치와 현재 시각을 초기화 한다.
copPos = 0, curTime = 0



[그림. 1]

Solution sketch

2. `void occur(int timeStamp, int caseID, int townNum, int prior);`

: timeStamp까지 1 시간 단위로 copPos(경찰의 위치)와 curTime(현재 시각)을 업데이트한다. 이는 update(timeStamp) 함수를 만들어 사용할 수 있다.

: 우선순위 큐에 caseID사건을 등록한다.

3. `void cancel(int timeStamp, int caseID);`

: timeStamp까지 1 시간 단위로 copPos(경찰의 위치)와 curTime(현재 시각)을 업데이트한다. 이는 update(timeStamp) 함수를 만들어 사용할 수 있다.

: caseID사건을 등록을 취소 표시 한다.
정적 배열을 만들어 표시하고 우선순위 큐에서는 lazy update로 처리 할 수 있다.

*lazy update : pq의 top()올라오기 전까지 값의 업데이트를 보류한다.
pq의 top()에 올라오면 그때서야 유효성 판정을 통해 이후 프로세스를 진행한다.

Solution sketch

4. `int position(int timeStamp);`

   : timeStamp까지 1 시간 단위로 copPos(경찰의 위치)와 curTime(현재 시각)을 업데이트한다. 이는 update(timeStamp) 함수를 만들어 사용할 수 있다.

   : copPos(경찰의 위치)를 반환한다.

- occur(), cancel(), position() 함수 모두에서 공통적으로 update(timeStamp) 를 필요로 한다.

  update(timeStamp) 함수를 다음과 같이 구현할 수 있다.

Solution sketch


```
void update(int endTime) {
    for (; curTime < endTime; curTime++) {
        // 1. 최 우선순위 구하기
        while (!pq.empty() && cancelFlag[pq.top().id]) // lazy update
            pq.pop();
        if (pq.empty()) { // 최 우선순위가 없는 경우
            curTime = endTime;
            break;
        }
        // 2. 최 우선순위가 있는 경우
        int destTown = pq.top().town;
        if (destTown == copPos)  // 목적지 도착후 1시간 지남.
            pq.pop();
        else                     // 다음 마을로 : 다음에 확인할때는 1시간이 지난 상태가 됨.
            copPos = nextTown[copPos][destTown];
    }
}
```

Solution sketch

[summay]

- 우선순위에 맞게 시간에 따른 처리가 필요한 문제이다.

- 시간에 다른 처리 문제는 종종 출제 되는 유형이다.
  잘 준비해 두는 것이 필요하다.

- 우선순위를 사용하는 문제는 필수 유형이라고 할 수 있다.
  철처한 준비가 필요하다.

Code example

Code example


```cpp
#include <cstring>
#include <queue>
#include <vector>
using namespace ::std;

const int LM = 351;
const int IDLM = 100000;

int curTime;           // 현재 시각
int copPos;            // 경찰의 현재 위치
int nextTown[LM][LM];  // 목적지를 향하여 이동할 때, 1시간 뒤 위치 테이블
int cancelFlag[IDLM];  // 취소 표시
int visited[LM], vn;

struct Event {
    int tick;    // 사건이 일어난 시각
    int id;      // 사건 id
    int town;    // 목저도시
    int prior;   // 처리 우선순위
    bool operator<(const Event&t)const {  // 1. 처리우선순위 desc, 2. 일어난 시각 asc
        return (prior != t.prior ? prior < t.prior : tick > t.tick);
    }
};
```

Code example


```
priority_queue<Event> pq;  // 우선순위 큐
vector<int> adj[LM];       // 인접 배열

void dfs(int src, int town, int dest) {// nextTown[][]구하기 방법 1 : dfs를 이용
    if (visited[dest] == vn) return;
    visited[dest] = vn;
    nextTown[src][dest] = town;
    for (auto d : adj[dest]) dfs(src, town, d);
}

void bfs(int src, int town, int dest) {// nextTown[][]구하기 방법 2 : bfs를 이용
    queue<int> que;
    visited[town] = vn;
    que.push(town);
    while (!que.empty()) {
        dest = que.front();
        que.pop();
        nextTown[src][dest] = town;
        for (auto d : adj[dest]) {
            if (visited[d] != vn) {
                visited[d] = vn;
                que.push(d);
            }
        }
    }
}
```

Code example


```
void init(int N, int parent[]) {
    curTime = 0;
    copPos = 0;
    memset(cancelFlag, 0, sizeof(cancelFlag));
    pq = {};

    // 인접배열 만들기
    for (int i = 0; i < LM; ++i) adj[i].clear();
    for (int i = 1; i < N; i++) {
        adj[i].push_back(parent[i]);
        adj[parent[i]].push_back(i);
    }

    // nextTown[][]만들기
    for (int s = 0; s < N; s++) {
        visited[s] = ++vn;
        //for (auto m : adj[s]) dfs(s, m, m); // 방법 1
        for (auto m : adj[s]) bfs(s, m, m); // 방법 2
    }
}
```

Code example


```
void update(int endTime) {
    for (; curTime < endTime; curTime++) {
        // 1. 최 우선순위 구하기
        while (!pq.empty() && cancelFlag[pq.top().id])
            pq.pop();
        if (pq.empty()) { // 최 우선순위가 없는 경우
            curTime = endTime;
            break;
        }
        // 2. 최 우선순위가 있는 경우
        int destTown = pq.top().town;
        if (destTown == copPos)  // 목적지 도착후 1시간 지남.
            pq.pop();
        else                     // 다음 마을로 : 다음에 확인할때는 1시간이 지난 상태가 됨.
            copPos = nextTown[copPos][destTown];
    }
}
```

Code example


```
void occur(int timeStamp, int caseID, int townNum, int prior) {
    update(timeStamp);
    pq.push({ timeStamp, caseID, townNum, prior }); // 새로운 사건 발생
}

void cancel(int timeStamp, int caseID) {
    update(timeStamp);
    cancelFlag[caseID] = 1; // 사건 처리 취소.
}

int position(int timeStamp) {
    update(timeStamp);
    return copPos;          // 현재 경찰의 위치
}
```

Thank you.