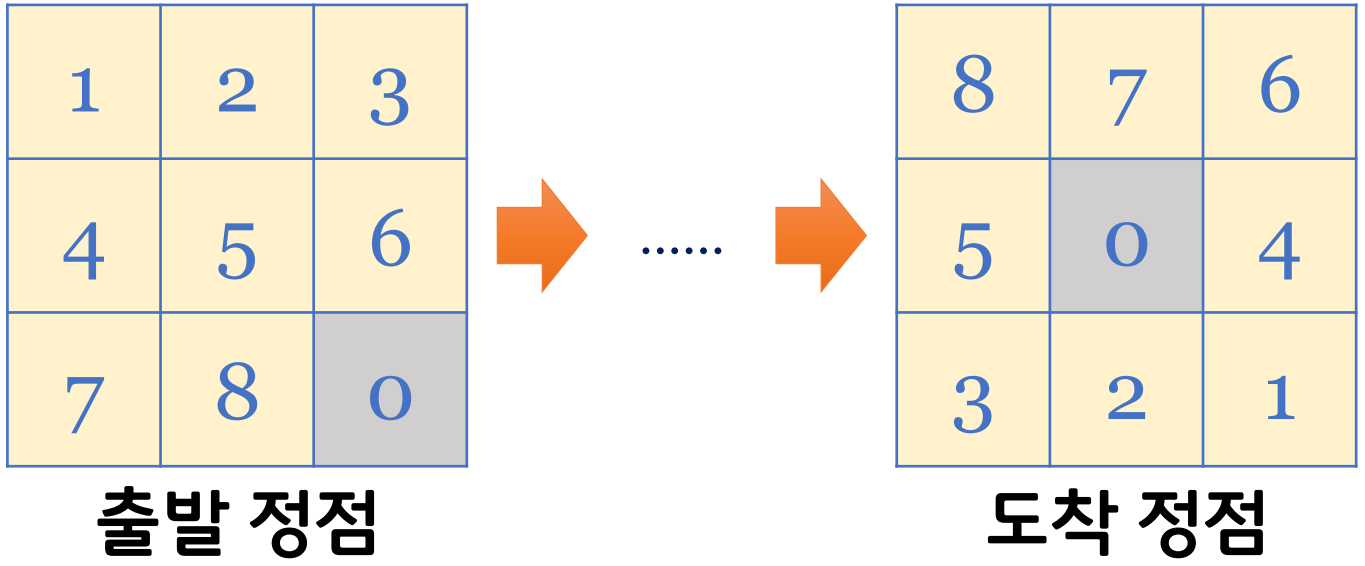
TMItalk: Too Much Information Talk

8-퍼즐 문제와 A* 알고리즘

2. 다익스트라로 풀어 보자.

- 8-퍼즐 문제: 다익스트라로 풀어보기
  - 8-퍼즐 문제는 결국
    - 보드의 상태 공간 그래프에서의 최단 경로 찾기 문제: 이전 영상 참고
  - 최단 경로 찾기라면 다익스트라 알고리즘
    - 출발 정점: 목표 상태
    - 도착 정점: 초기 상태
    - 거리: 중간 상태의 수 + 1



- 그래프의 노드(정점)의 표현:

8	7	6
5	0	4
3	2	1

[8, 7, 6, 5, 0, 4, 3, 2, 1]

876504321

```
def state_to_int(state):
    s = ""
    for i in range(len(state)):
        s += str(state[i])
    return int(s)

def int_to_state(v):
    s = str(v)
    if len(s) != N:
        s = "0" + s
    return list(map(int, s))
```

TMItalk: 8-퍼즐 문제와 A* 알고리즘

- **정점 집합의 초기화:**
  - 모든 도달가능한 정점 집합을 생성: 0부터 $\frac{N!}{2} - 1$까지 인덱스 부여
  - 정점 번호와 정점 상태와의 변환 딕셔너리 생성
    - $vtos$: 정점 번호에서 정점 상태로
    - $stov$: 정점 상태에서 정점 번호로

```
vtos, stov = {}, {} # mapping vertex to state, and vice versa.
for perm in permutations(solved):
    if is_solvable(n, perm):
        v, s = len(vtos), state_to_int(perm)
        vtos[v], stov[s] = s, v
```

$0 \longleftrightarrow 123456780$

$1 \longleftrightarrow 123456708$

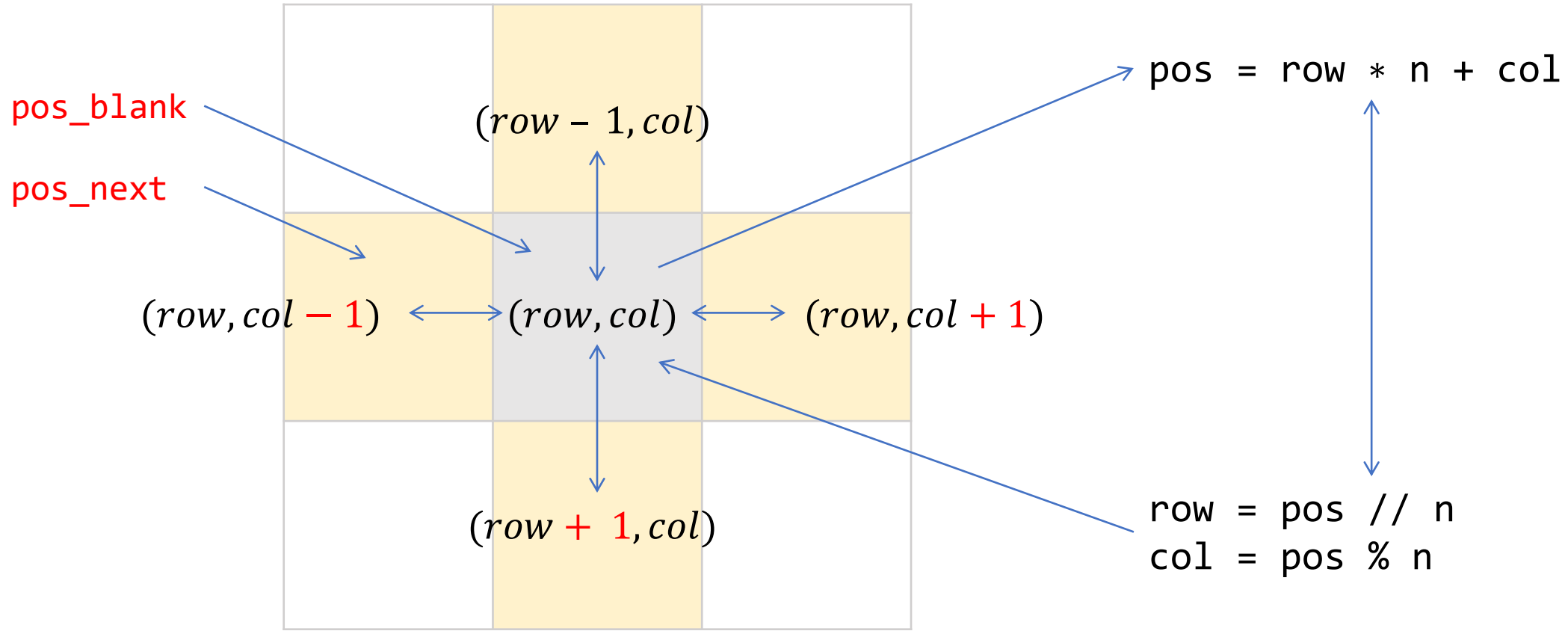
```
start = stov[state_to_int(solved)]
target = stov[state_to_int(board)]
bypass = dijkstra(start, target, len(vtos))
path = find_path(start, target, bypass)
for v in path[::-1]:
    state = int_to_state(vtos[v])
    for i in range(len(state)):
        print(state[i], end = " ")
        if (i % n == n - 1):
            print()
    print()
print('the number of steps to solve it:', len(path) - 1)
```

- **다익스트라 알고리즘 구현:**

```
def dijkstra(start, target, vsize):
    length = [INF for _ in range(vsize)]
    bypass = [0 for _ in range(vsize)]
    length[start] = 0
    heap = []
    heapq.heappush(heap, (length[start], start))
```

```
while len(heap) > 0:
    u = heapq.heappop(heap)[1]
    if u == target:
        return bypass
    for v in get_neighbors(u):
        if length[v] < 0:
            continue # skip vertices already visited.
        elif length[v] > length[u] + 1:
            length[v] = length[u] + 1
            bypass[v] = u
            heapq.heappush(heap, (length[v], v))
    length[u] = -1 # mark u as visited
return bypass
```

- 이웃 정점 찾기:




```
dir = [(-1, 0), (1, 0), (0, -1), (0, 1)] # up, down, left, right
```

TMItalk: 8-퍼즐 문제와 A* 알고리즘

```
def get_neighbors(u):
    state = int_to_state(vtos[u])
    pos_blank = state.index(0)
    row, col = pos_blank // n, pos_blank % n
    neighbors = []
    for i in range(len(dir)):
        nextrow, nextcol = row + dir[i][0], col + dir[i][1]
        if 0 <= nextrow < n and 0 <= nextcol < n:
            pos_next = nextrow * n + nextcol
            board = state[:]
            board[pos_blank], board[pos_next] = board[pos_next], board[pos_blank]
            neighbors.append(stov[state_to_int(board)])
    return neighbors
```

TMItalk: 8-퍼즐 문제와 A* 알고리즘

```
puzzle.3.2.in:

4
0 3
2 1
```

```
0 3
2 1

2 3
0 1

2 3
1 0

2 0
1 3

0 2
1 3

1 2
0 3

1 2
3 0

the number of steps to solve it: 6
```

TMItalk: 8-퍼즐 문제와 A* 알고리즘

```
puzzle.8.4.in:

9
0 8 7
6 5 4
3 2 1
```

```
0 8 7
6 5 4
3 2 1

6 8 7
0 5 4
3 2 1

6 8 7
3 5 4
0 2 1

……

1 2 3
4 5 6
7 8 0

the number of steps to solve it: 28
```

TMItalk: 8-퍼즐 문제와 A* 알고리즘

- 다익스트라로 15-퍼즐도 풀 수 있을까?
  - 8-퍼즐을 푸는 데 걸리는 시간: 약 2~3초 내외
  - 15-퍼즐에 적용하려면?
    - 일단 상태 공간이 15!(약 20조)이므로 모든 벡터를 생성조차 할 수 없다.
    - 상태 공간을 명시적으로 생성하지 않더라도
    - 현재 구현으로는 목적지에 도달할 때까지 모든 인접 노드를 방문한다.
  - 결론: 15-퍼즐을 풀려면 A* 알고리즘을 적용해야 한다.

Any Questions?

주니온TV@Youtube

자세히 보면 유익한 코딩 채널