알고리즘 숙제 #2 - 201901551 컴퓨터공학부 김정목

1. 문제

당신은 놀이공원 매표소의 직원입니다. 저녁 식사를 마치고 오후 6시부터 오후 9시까지 들어오는 손님들의 입장을 관리하게 되었습니다.

현재 매표소의 개수는 2개이고, 7시 30분부터 야간개장을 위해 2개의 매표소가 추가로 열립니다. 당신에게는 입장하는 손님에 대한 정보가 담긴 “customer.txt” 파일이 존재합니다.
이때, 손님들의 입장을 가장 효율적으로 받는 방법을 알려주세요

출력 형태는 다음과 같다.

```
201901551 김정목
1번 매표소 : AAAEEEDDDDDXXXXXXX
2번 매표소 : CCCFFFFFXXXXGHHHXX
3번 매표소 : XXXXXXXXXXXXYYXXXX
4번 매표소 : XXXXXXXXXXXXXXXXXX
```

구현해야 할 최소 함수는 다음과 같다. (추가로 필요한 함수는 각자 생성)

```
init() := 파일 입출력으로 txt 파일을 읽어오는 함수
run() := 알고리즘을 실행해주는 함수로, 최종적으로 출력형태가 나올 수 있도록 한다
```

조건

1. 손님은 1~6인 가족으로 이루어져 있으며 인당 10분의 시간이 소요됨 (2인 가족 - 20분 소요)

2. 가족의 수는 16이며, 번호는 A~Z 로 있으며, 앞의 여섯 가족은 오후 6시를 기준으로 할 때 도착하여 대기하고 있는 상태이고, 뒤의 열 가족은 오후 7시 30분에 동시에 도착한다고 가정함

이미 도착하여 대기하고 있는 가족들에 대해서는 가장 소요시간이 짧을 수 있는 순서대로 입장을 관리하고, 시간이 같다면 알파벳 순으로 입장을 관리함.

(번호에서 X는 없으며, 파일 입출력으로 알파벳을 미리 읽어서 출력해야 함)

3. 출력형태는 10분 간격으로 매표소가 n번 가족을 작업하고 있는 지를 보여줌 (1번 매표소의 AAAEEEDDDDXX는 오후 6시부터 8시까지 10분 간격으로 A번 가족 30분, E번 가족 30분, D번 가족 40분, X =빈 시간 20분을 의미함)

4. 만약 모든 매표소가 이용 가능하다면 가장 낮은 숫자의 매표소 순으로 우선순위를 가진다 (1번 매표소와 3번 매표소가 이용 가능하면 1번부터 배정)

5. main 함수에는 init() 함수와 run() 함수만 존재할 것

6. 출력 부분의 202301234 홍길동 부분은 본인의 학번/이름으로 작성할 것 7. 작성한 소스코드에 대한 주석을 반드시 작성할 것

2. 코드 설명

주요 함수	설명
void init()	'customer.txt' 파일에서 가족 정보를 읽어와 초기화하는 함수
void run()	스케줄링과 결과를 출력하는 주요 함수를 호출하는 함수
void arrange()	가족 정보 배열 정렬, 매표소 상태 배열 초기화를 해주는 함수
void scheduleTasks()	구성원 수와 도착 시간을 기준으로 최적의 매표소에 할당하는 함수
void printSchedule()	각 매표소에서 어떤 가족 구성원이 언제 처리되는지 출력하는 함수
int compareFamily(const void* a, const void* b)	가족의 구성원 수와 가족의 알파벳을 기준으로 정렬하는 함수

주요 변수	설명
MAX_FAMILY 16	최대 가족 수를 뜻하는 변수
EARLY_ARRIVAL 6	18시 대기 가족과 19시 30분 대기 가족을 구분하기 위한 변수
MAX_TICKET_BOOTH 4	매표소의 총 개수
MAX_ENDTIME 18	최대 스케줄링 시간을 뜻하는 변수

주요 구조체	설명
Family	**가족 정보를 담는 구조체**
char id;	가족의 알바펫을 뜻하는 변수
int familySize;	가족의 구성원 수를 뜻하는 변수
TicketBoot	**매표소 상태를 담는 구조체**
int nextAvailableTime	다음 작업을 처리할 수 있는 가장 빠른 시간을 뜻하는 변수
char tasks[MAX_ENDTIME]	매표소에 할당된 작업을 저장하는 배열

주요 배열	설명
Family families[MAX_FAMILY]	모든 가족의 정보를 담은 배열
TicketBooth ticketBooths[MAX_TICKET_BOOTH]	모든 매표소의 상태를 담은 배열

알고리즘

1. init 함수

- 'customer.txt' 파일에서 가족의 정보를 읽어와 배열에 저장한다.

2. arrange 함수

- 총 가족 수인 16을 6(18시 도착)과 10(19시 30분 도착)으로 나눠 compareFamily 함수를 통해 qsort 정렬을 각각 해준다.
- 1번 2번 매표소는 18시 시작(nextAvailableTime=0), 3번 4번 매표소는 19시 30분 시작(nextAvailableTime=9)으로 초기화하며 모든 매표소의 작업 스케줄을 'X' 로 초기화한다.

3. scheduleTasks 함수

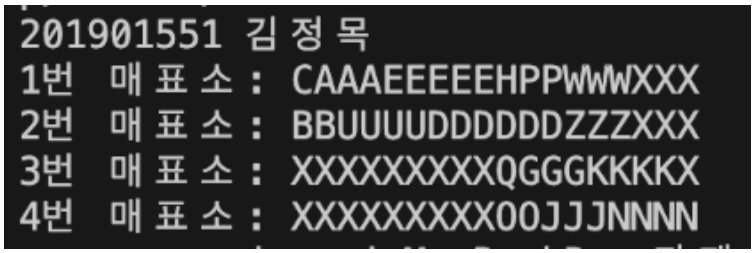
- arrange 함수를 호출하여 데이터를 정렬한다.
- 먼저 18시에 도착한 6 가족은 19시 30분 전까지 1, 2 매표소에서 작업을 진행한다.
- 1번과 2번 매표소 중 어느 매표소에서 가장 빨리 가족을 수용할 수 있는지 확인하고 그 매표소에 가족을 할당한다.
- 매표소에 가족을 할당한 후 해당 매표소의 nextAvailableTime을 가족의 구성원 수만큼 증가시켜 갱신한다.
- 19시 30분 전에 남은 가족이 없어서 빈 시간을 가진 매표소가 있다면 nextAvailableTime을 9로 한다.
- for 문을 두 개로 나눈 이유는 19시 30분에 들어와야 할 가족이 19시 30분 전에 들어오는 것을 방지하기 위해서다.
- 또 첫 번째 for 문 안의 매표소 순회 for 문 j의 반복 범위를 0 ~ 2(1, 2 매표소)가 아닌 0 ~ MAX_TICKET_BOOTH로 둔 이유는 19시 30분이 지나도 들어가지 못한 가족이 있다면, 19시 30분 이후에는 모든 매표소에서 입장시키기 위해서이다.
- 나머지 가족은 모든 매표소에서 위와 같이 작업을 진행한다.

4. printSchedule 함수

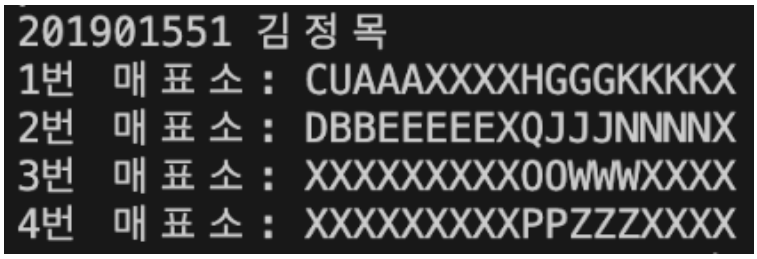
- 각 매표소에서 처리되는 가족 구성원의 스케줄을 출력한다.

3. 실행 출력화면

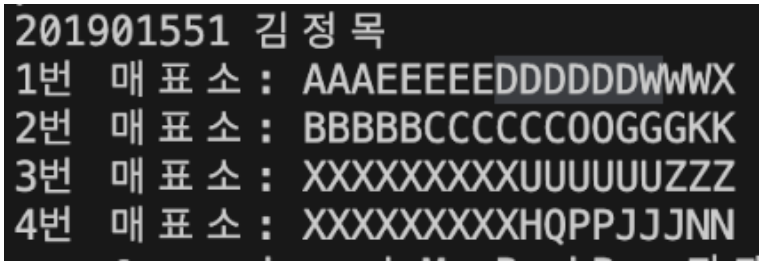
• 이러닝 기준 “customer.txt” 출력 결과



• 18시 대기 가족이 19시 30분 전에 모두 들어가 빈 시간이 생긴 경우 출력 결과 (customer.txt를 수정해서 예외 결과도 정상적으로 출력되는지 확인함)



• 18시 대기 가족이 19시 30분까지 들어가지 못 할 경우 출력 결과 (customer.txt를 수정해서 예외 결과도 정상적으로 출력되는지 확인함)

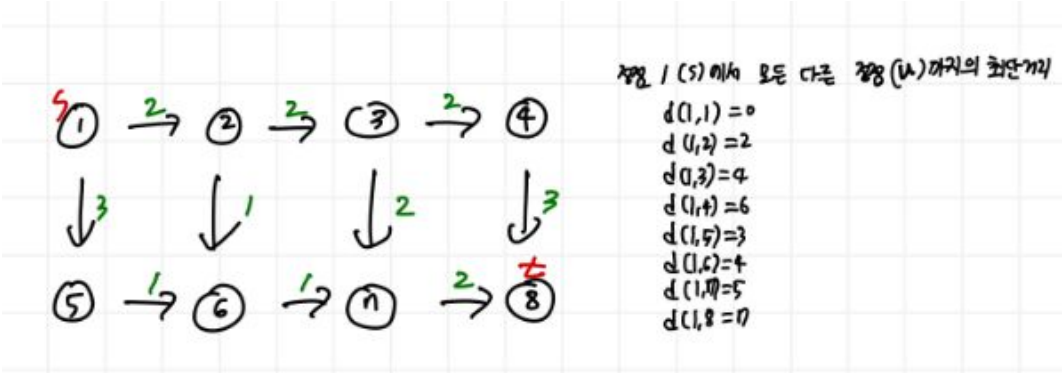


4. 문제풀이 - 1

49. 가중치를 가진 방향 그래프 G가 주어지고 G에 있는 점 s로부터 각각 다른 점 u까지의 최단 거리 d(s,u)만 주어질 때, s로부터 특정한 점 t까지 최단 경로를 찾는 O(n+m) 시간 알고리즘을 작성하라. 단, n과 m은 G에 있는 정점과 간선의 수이다.
- 정점 u까지의 간선(w,u)에서 d(s,w) + 가중치(w,u) = d(s,u)라면 w는 최단 경로상의 경유지임을 인지하고 pseudo code로 작성

1. 인접 리스트는 각 정점에 연결된 간선을 효율적으로 저장하며, 특정 정점에 대한 들어오는 간선 확인 작업은 해당 정점의 차수에 비례하여 O(정점의 차수) 시간이 소요된다. 그러므로 아래 알고리즘은 그래프 G를 인접 리스트로 가정했다.
2. 정점 t에서 시작하여 s까지 역추적한다.
3. 각 정점 t에 대해서, 조건 d(s,w) + weight(w,t) = d(s,t)를 만족하는 정점 w를 찾는다.
4. 해당 조건을 만족하는 정점 w가 있으면, w는 최단 경로 상에서 t의 앞에 오는 정점이다.
5. 이 과정을 s까지 반복한다.

아래로 알고리즘을 예를 들어보겠다.



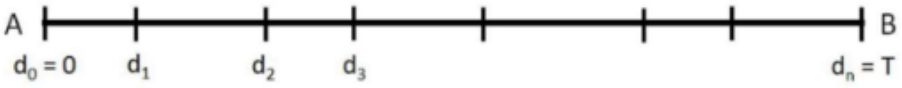
1. 정점 8(t) - 정점 7이 조건을 만족한다. d(1,7) + weight(7,8) = 5 + 2 = 7 (7은 8 앞에 온다.)
2. 정점 7 - 정점 6이 조건을 만족한다. d(1,6) + weight(6,7) = 4 + 1 = 5 (6은 7 앞에 온다.)
3. 정점 6 - 정점 5가 조건을 만족한다. d(1,5) + weight(5,6) = 3 + 1 = 4 (5은 6 앞에 온다.)
4. 정점 5 - 정점 1(s)이 조건을 만족한다. d(1,1) + weight(1,5) = 0 + 3 = 3 (1은 5 앞에 온다.)
5. 결론적으로 정점 1(s)에서 8(t)까지 최단 경로는 1 -> 5 -> 6 -> 7 -> 8 이다.

```
def 최단경로(G, d, 시작점, 종료점): // G는 인접 리스트다.
    경로 = []
    현재_정점 = 종료점

    while 현재_정점 != 시작점:
        경로.insert(0, 현재_정점)
        찾음 = False
        // 현재 정점에 대한 들어오는 간선만 확인
        for (w, u) in G[현재_정점]: // 현재_정점과 연결된 간선들만 순회한다.
            if d[시작점][w] + weight(w, u) == d[시작점][u]:
                현재_정점 = w
                찾음 = True
                break
        if not 찾음:
            return "시작점에서 종료점까지 경로가 존재하지 않는다."

    경로.insert(0, 시작점)
    return 경로
// 시간 복잡도:
// 1. 모든 정점에 대해 한 번씩 검사: O(n)
// 2. 모든 간선에 대해 한 번씩 검사: O(m)
// 결론적으로, 시간 복잡도는 O(n+m)이다.
```

5. 문제풀이 - 2

65. 도시 A에서 도시 B로 가려고 하는데 그 거리 (T km)가 너무 멀어서 며칠 동안을 자동차로 운전해야만 한다. 미리 지도에서 도로 가의 주유소들의 위치를 찾고 출발점으로부터 거리를 다음과 같이 계산하였다. 단, d0 = 0 < d1 < d2 < .... < dn-1 < dn = T이다.

A의 시작점 = d_0, d_1, d_2, ... , d_n = B의 도착점



가득 찬 연료 탱크를 가지고 C km를 주행할 수 있는 자동차로 도시 A에서 도시 B로 가는데 필요한 최소 주유 횟수를 찾기 위한 알고리즘을 제안하고, 제안한 알고리즘이 왜 최소 횟수의 주유만으로 A로부터 B에 도착할 수 있는지를 설명(증명)하라.
- 최소 주유 횟수를 만들기 위한 조건이 핵심 아이디어로, 이에 대한 pseudo code와 해당 알고리즘에 대한 증명 관련 설명을 작성

<핵심 아이디어>
1. 주유소 위치와 주행 가능한 거리 정보를 기반으로 주유소를 선택한다.
2. 시작 위치에서 출발하며, 현재 위치에서 현재 연료로 갈 수 있는 가장 먼 주유소를 선택한다.
3. 선택한 주유소까지 이동한 후, 다음으로 현재 연료로 갈 수 있는 가장 먼 주유소를 선택한다.
4. 이런 식으로 주유 간격을 최대화로 하여 도시 B에 도달할 때까지 반복한다.
5. 문제에서 A에서 B로 도착을 가정을 했으므로 가득 찬 연료 C로 들릴 수 없는 주유소는 존재하지 않는다고 가정하겠다.

• 만약 현재 위치에서 최대한의 연료로 다음 주유소까지 도달할 수 있다면, 중간에 추가 주유를 하지 않아도 된다. 결국 불필요한 주유소 방문 없이 최소한의 주유 횟수로 도달한다.

• 이 알고리즘을 따르면 항상 다음 주유소까지 가기에 충분한 연료를 보유하며, 목적지에 도달할 때까지 최소 주유 횟수를 보장한다.

<증명>
1. 현재 위치에서 주어진 연료로 도달 가능한 거리 내에 여러 개의 주유소가 있을 때, 중간의 주유소를 선택한다고 가정한다.
2. 이 선택으로 인해 다음 주유소까지의 거리가 현재 연료의 최대 주행 거리보다 길어진다면, 중간에 다시 주유를 해야한다.
3. 반면, 처음에 가능한 한 가장 멀리 있는 주유소를 선택했다면, 이후에 더 적게 주유할 가능성이 높아진다.
4. 따라서 주어진 연료로 도달 가능한 거리 내에서 가능한 한 가장 멀리 있는 주유소를 선택하는 것이 최소 주유 횟수로 도시 B에 도착하는 방법이다.

```
def 최소_주유_횟수(주유소_위치, T, C):
    현재_위치 = 0
    주유_횟수 = 0
    현재_연료 = C

    while 현재_위치 < T:
        최대_주유소_도달_거리 = 현재_위치 + 현재_연료
        선택한_주유소 = None

        // 현재 연료로 도달 가능한 가장 먼 주유소를 선택
        for 주유소 in 주유소_위치:
            if 현재_위치 < 주유소 <= 최대_주유소_도달_거리:
                선택한_주유소 = 주유소

        // 선택한 주유소로 이동하고 연료 충전
        주유_횟수 += 1
        현재_연료 -= (선택한_주유소 - 현재_위치)
        현재_위치 = 선택한_주유소
        현재_연료 = C
    return 주유_횟수
```