JooHyun – Lee (comkiwer)

TS항공노선관리

Hancom Education Co. Ltd.

Problem

Problem

TS항공노선관리

N개의 공항이 있다. 각 공항은 0 이상 N-1 이하의 고유 번호를 가진다.

하루는 24시간이다.

시각의 기준은 모두 통일되어 있으므로, 공항 간의 시차는 고려하지 않아도 된다.

모든 항공 노선은 하루에 한 번 정해진 시각에 운행하는 정기 노선이다.

쉬는 날은 없으며, 취소되는 날도 없다.

항공 노선에 대한 정보로 출발 공항 번호, 도착 공항 번호, 출발 시각, 비행 소요 시간, 티켓 가격이 주어진다.

출발 시각이 빨라지거나 지연되는 경우는 없다. 비행 소요 시간이 줄어들거나 늘어나는 경우도 없다.

따라서 도착 시각은 출발 시각과 비행 소요 시간을 통해 정확히 알아낼 수 있다.

출발 시각과 도착 시각은 항상 정각이다.

Problem

[Fig. 1]은 항공 노선의 한 예시이다.

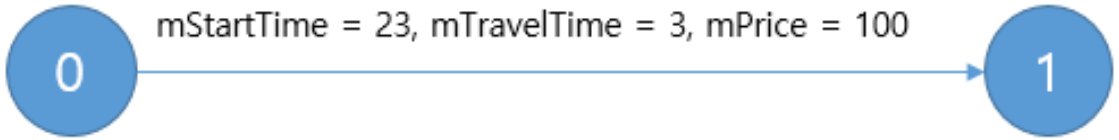
이 항공 노선의 출발 공항 번호는 0, 도착 공항 번호는 1이다.

또한 출발 시각은 23시, 비행 소요 시간은 3시간이다.

하루는 24시간이므로, 도착 시각은 다음날 2시임을 알아낼 수 있다.

즉, 이 항공 노선은 매일 23시에 0번 공항에서 출발하여, 다음 날 2시에 1번 공항에 도착하는 정기 노선이다.

티켓 가격은 100 달러이다.



[Fig. 1]

환승에는 시간이 소요되지 않는다.

어떤 공항에 2시에 도착하면, 그 즉시, 그 공항에서 2시에 출발하는 항공 노선을 이용할 수 있다. 그러나 2시에 도착한 후, 3시에 출발하는 항공 노선을 이용하고자 할 때는 1시간을 대기해야 한다.

Problem

TS항공노선관리

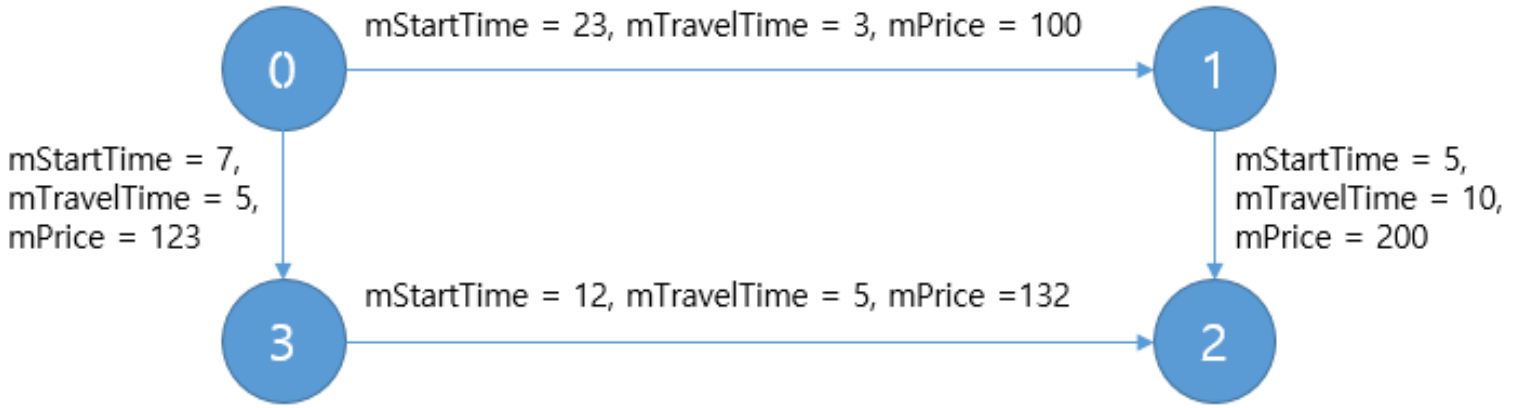
[Fig. 2]는 4개의 항공 노선이 있는 경우의 예시이다.

0번 공항에서 20시에 출발하여, 최대한 빠르게 2번 공항으로 갈 경우, 총 19시간이 소요된다.
그 방법은 아래와 같다.

(0번 공항, 20시) → 3시간 대기 → (0번 공항, 23시) → 3시간 이동 → (1번 공항, 2시)
→ 3시간 대기 → (1번 공항, 5시) → 10시간 이동 → (2번 공항, 15시)

0번 공항에서 2번 공항으로, 최대한 적은 비용으로 이동할 경우, 총 255 달러가 소모된다.
그 방법은 아래와 같다.

(0번 공항) → 123달러 → (3번 공항) → 132달러 → (2번 공항)



[Fig. 2]

Problem

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

void init(int N)

각 테스트 케이스의 처음에 호출된다.

공항은 N개이다.

테스트 케이스의 시작 시, 존재하는 항공 노선은 없다.

Parameters

N : 공항의 수 (3 ≤ N ≤ 60)

Problem

TS항공노선관리

void add(int startAirport, int endAirport, int startTime, int travelTime, int mPrice)

startAirport번 공항에서 endAirport번 공항으로 가는 항공 노선을 추가한다.

출발 시각은 startTime시이다.

비행 소요 시간은 travelTime 시간이다.

티켓 가격은 mPrice 달러이다.

두 공항 사이에 여러 개의 항공 노선이존재할 수 있음에 유의하라.

startAirport와 endAirport는 서로 다름이 보장된다.

Parameters

startAirport : 출발 공항 번호 (0 ≤ startAirport ≤ N-1)

endAirport : 도착 공항 번호 (0 ≤ endAirport ≤ N-1)

startTime: 출발 시각 (0 ≤ startTime ≤ 23)

travelTime : 비행 소요 시간 (1 ≤ travelTime ≤ 23)

mPrice : 티켓 가격 (10 ≤ mPrice ≤ 5,000)

Problem

TS항공노선관리

int minTravelTime(int startAirport, int endAirport, int startTime)

startAirport번 공항에서 startTime시에 출발하여, endAirport번 공항으로 소요 시간을 최대한 적게 하면서 이동할 때의 소요 시간을 반환한다.

startAirport번 공항에서 startTime시에 출발하여, endAirport번 공항으로 갈 수 없으면, -1을 반환한다.

startAirport와 endAirport는 서로 다름이 보장된다.

Parameters

startAirport : 출발 공항 번호 (0 ≤ startAirport ≤ N-1)

endAirport : 도착 공항 번호 (0 ≤ endAirport ≤ N-1)

startTime: 출발 시각 (0 ≤ startTime ≤ 23)

Return

최소 이동 시간

Problem

TS항공노선관리

int minPrice(int startAirport, int endAirport)

startAirport번 공항에서 endAirport번 공항으로 비용을 최대한 적게 하면서 이동할 때의 비용을달러 단위로 반환한다.

비용은 이용한 항공 노선들의 티켓 가격의 총합이다.

startAirport번 공항에서 endAirport번 공항으로 갈 수 없으면, -1을 반환한다.

startAirport와 endAirport는 서로 다름이 보장된다.

Parameters

startAirport : 출발 공항 번호 (0 ≤ startAirport ≤ N-1)

endAirport : 도착 공항 번호 (0 ≤ endAirport ≤ N-1)

Return

최소 비용

Problem

TS항공노선관리

[제약사항]

1. 각 테스트 케이스 시작 시 `init()` 함수가 호출된다.
2. 각 테스트 케이스에서 `add()` 함수의 호출 횟수는 `30,000` 이하이다.
3. 각 테스트 케이스에서 `minTravelTime()` 함수의 호출 횟수는 `1,000` 이하이다.
4. 각 테스트 케이스에서 `minPrice()` 함수의 호출 횟수는 `5,000` 이하이다.

Problem analysis

Problem analysis : 예제 분석

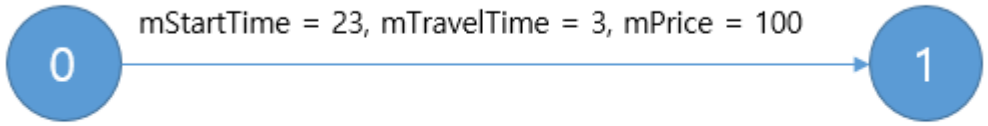
TS항공노선관리

[예제]

Note **부분에서,** (a, b)**는** (a**번 공항,** b**시**)**을 의미한다.**
즉, (0, 23)**는** (0**번 공항,** 23**시**)**를 의미한다.**

[Table 1]

Order	Function	Return	Note	Figure
1	init(6)			
2	add(0, 1, 23, 3, 100)			[Fig. 1]
3	minTravelTime(0, 1, 23)	3	(0, 23)→(1, 2)	
4	minPrice(0, 1)	100	0→1	
5	minTravelTime(0, 2, 0)	-1		
6	minPrice(1, 0)	-1		



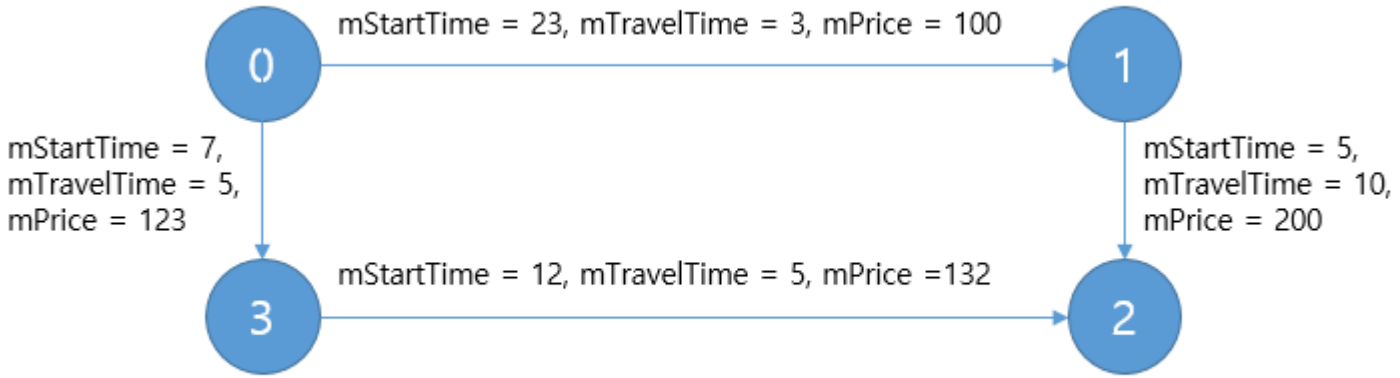
[Fig. 1]

Problem analysis : 예제 분석

TS항공노선관리

[Table 2]

Order	Function	Return	Note	Figure
7	add(0, 3, 7, 5, 123)			
8	add(3, 2, 12, 5, 132)			
9	add(1, 2, 5, 10, 200)			[Fig. 2]
10	minTravelTime(0, 2, 20)	19	(0, 20)→(0, 23)→(1, 2)→(1, 5)→(2, 15)	
11	minPrice(0, 2)	255	0→3→2	

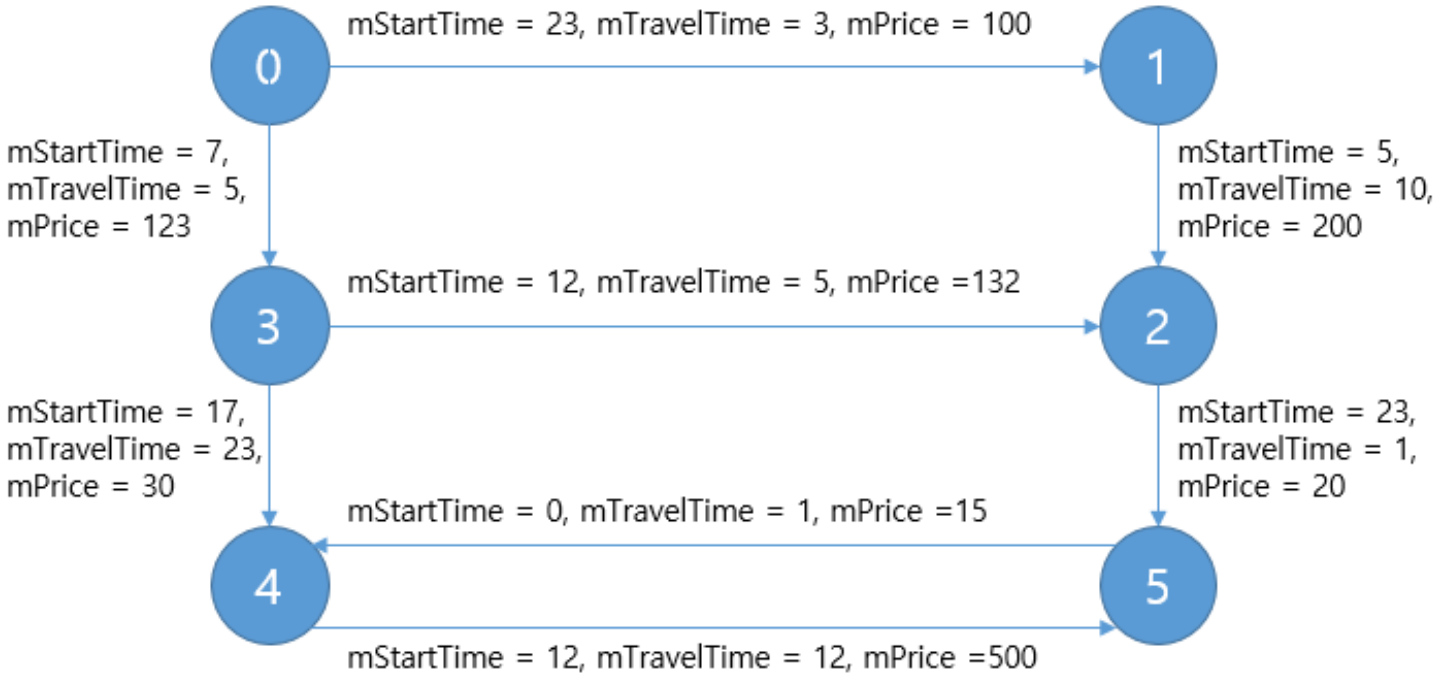


[Fig. 2]

Problem analysis : 예제 분석

[Table 3]

Order	Function	Return	Note	Figure
12	add(3, 4, 17, 23, 30)			
13	add(2, 5, 23, 1, 20)			
14	add(4, 5, 12, 12, 500)			
15	add(5, 4, 0, 1, 15)			[Fig. 3]
16	minTravelTime(3, 4, 12)	13	(3, 12)→(2, 17)→(2, 23)→(5, 0)→(4, 1)	

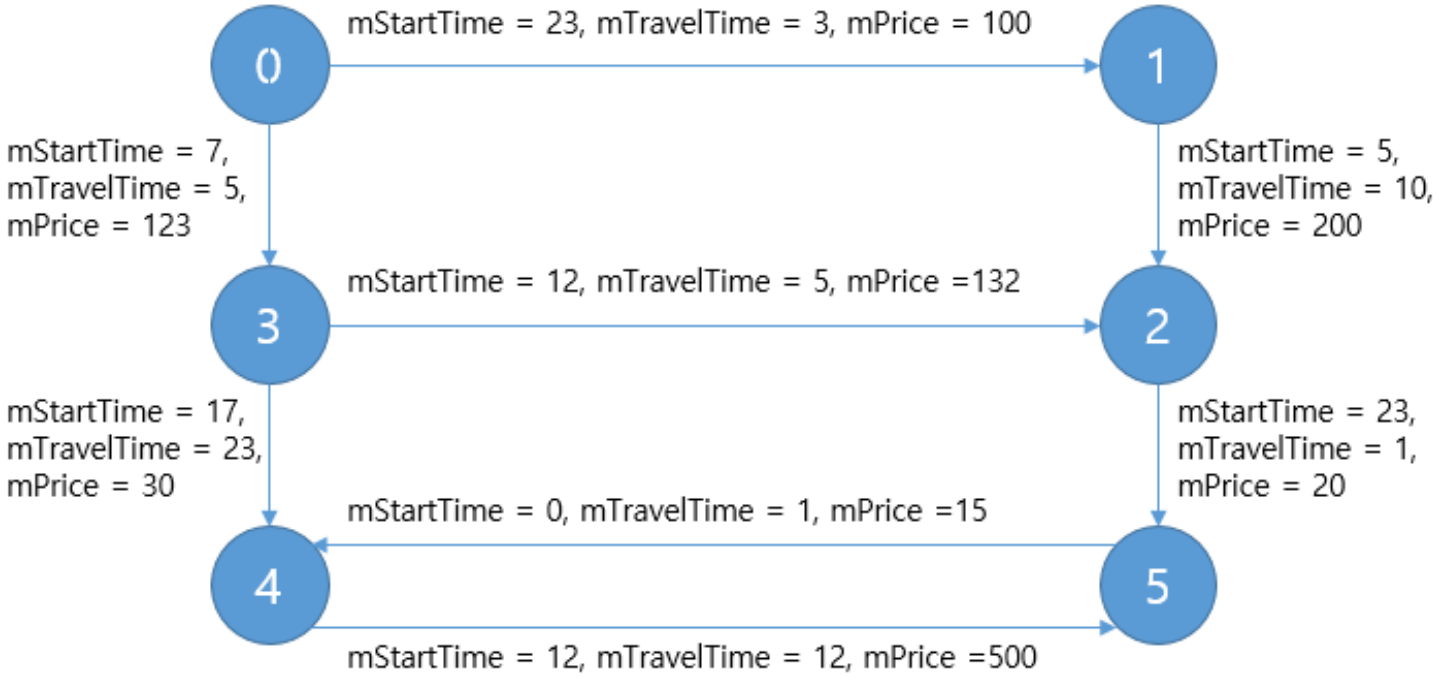


[Fig. 3]

Problem analysis : 예제 분석

[Table 3]

Order	Function	Return	Note	Figure
17	minPrice(3, 4)	30	3→4	
18	minPrice(0, 5)	275	0→3→2→5	
19	minTravelTime(5, 4, 1)	24	(5, 1)→(5, 0)→(4, 1)	
20	minTravelTime(1, 4, 20)	29	(1, 20)→(1, 5)→(2, 15) →(2, 23)→(5, 0)→(4, 1)	



[Fig. 3]

Problem analysis : 제약사항 및 API

TS항공노선관리

- 공항의 수 N은 최대 60이다.
- add() 함수 호출은 최대 30,000 이다.
- minTravelTime() 함수는 최대 1,000번 호출될 수 있다.
- minPrice() 함수는 최대 5,000번 호출될 수 있다.

- 출발지로부터 목적지까지 최단 시간 또는 최소 비용을 구하는 문제이므로
  Shortest Path:최단 경로(거리, 시간, 비용) 문제로 볼 수 있다.

Problem analysis : 제약사항 및 API

- minPrice() 함수에 응답하는 문제는 어렵지 않게 해결할 수 있다.
  - ✓ add(s, e, startT, travelT, price)함수로 새로운 정보가 주어지는 경우
    costTable[60][60]의 costTable[s][e]를 업데이트한다.
    costTable[s][e] = min(costTable[s][e], price);
  - ✓ minPrice() 함수가 호출되는 경우 $O(N^2)$ 또는 O(ElogN)
    Dijkstra 알고리즘을 이용하여 문제를 해결할 수 있다.
  - ✓ N의 최대값이 60이고 minPrice()함수 호출수는 최대 5000이므로
    O(60 * 60 * 5000) 에 충분히 해결 가능하다.

Problem analysis : 제약사항 및 API

TS항공노선관리

- int minTravelTime(int startAirport, int endAirport, int startTime)
  - ✓ startTime에 startAirport를 출발하여 endAirport에 도착하는 최단 시간을 구해야 한다.
  - ✓ 경유지 또는 출발지 공항의 비행기 출발 시간에 따라 얼마간 대기하는 시간을 고려해야 할 수 있다.
  - ✓ minPrice()와는 다른 상황이다. 이를 위한 문제 해결 방법이 필요하다.

- 시간과 비용테이블을 어떻게 설계할 것인가?
- add()함수로 주어지는 정보를 어떻게 처리할 것인가?
- minTravelTime() 호출에 어떻게 응답할 것인가?

Solution sketch

Solution sketch

TS항공노선관리

- add()함수 호출에는 timeTable과 priceTable을 업데이트만 하고
  minTravelTime()함수와 minPrice()함수에는
  O(N*N) Dijkstra 알고리즘을 이용하여 답하는 전략을 생각해 보자.

- minPrice()는 일반적인 최단 경로 문제로 해결이 가능하다.

- 하지만 minTravelTime()은 공항 사정에 따라 대기 시간을 고려해야 할 수 있다.
  비행기 이륙 시간은 시간 단위로 이루어 지므로
  timetable[60][60][24]를 선언하고
  timetable[s][e][t]를
  “s공항에 t시각에 도착했을 때, e공항으로 가는 최소 시간 " 으로 정의할 수 있다.
  timeTable[][][]을 업데이트 할 때 비행시간에 대기 시간을 더하여 반영한다면
  일반적인 문제로 바뀌게 되므로 어렵지 않게 해결 가능하다.

Solution sketch

- 시간 정보는 timeTable[60][60][24]에
  비용 정보는 costTable[60][60]에 저장할 수 있다.

- Dijkstra알고리즘을 수행하기 위하여 dist[60], used[60] 을 사용할 수 있다.
- 이를 위한 자료들을 선언하면 아래와 같다.

```
const int LM = 60;
const int INF = 1 << 20;

int N;
int costTable[LM][LM];
int timeTable[LM][LM][24];
int dist[LM], used[LM];
```

Solution sketch

이제 각 API함수별 할 일을 정의해 보자.

void init(int N)

- ✓ 공항의 수 N을 저장한다.
- ✓ 시간과 비용 테이블을 초기화 한다.

```
void init(int N) {
    ::N = N;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            costTable[i][j] = INF;
            for (int k = 0; k < 24; ++k)
                timeTable[i][j][k] = INF;
        }
    }
}
```

Solution sketch

void add(int startAirport, int endAirport, int startTime, int travelTime, int mPrice)

- ✓ costTable과 timeTable을 업데이트 한다.
- ✓ timeTable을 업데이트 할 때, 도착 시간과 출발 시간의 차를 고려하여 업데이트 한다.

```
void add(int s, int e, int st, int travelTime, int mPrice) {
    costTable[s][e] = min(costTable[s][e], mPrice);
    for (int arrivalTime = 0; arrivalTime < 24; ++arrivalTime) {
        int waitingTime = (st - arrivalTime + 24) % 24;
        int travelTimeCost = travelTime + waitingTime;
        timeTable[s][e][arrivalTime] = min(timeTable[s][e][arrivalTime], travelTimeCost);
    }
}
```

Solution sketch

int minTravelTime(int startAirport, int endAirport, int startTime)

- ✓ Dijkstra알고리즘을 이용하여 최단 시간을 구하여 반환한다. 아래 예는 $O(N^2)$.
- ✓ 경로를 구할 수 없는 경우 -1을 반환한다.

```
// 0. init
int i, j, minNode, minTime;
for (i = 0; i < N; ++i) dist[i] = INF, used[i] = 0;
dist[s] = startTime;
for (i = 0; i < N; ++i) {
    minTime = INF;
    for (j = 0; j < N; ++j) {    // 1. extract minNode
        if (!used[j] && dist[j] < minTime)
            minNode = j, minTime = dist[j];
    }
    if (minTime == INF) break;
    used[minNode] = 1;           // 2. check minNode
    if (minNode == e) return minTime - startTime;
    for (j = 0; j < N; ++j) {    // 3. relaxation
        if (!used[j] && dist[j] > minTime + timeTable[minNode][j][minTime % 24])
            dist[j] = minTime + timeTable[minNode][j][minTime % 24];
    }
}
return -1;
```

Solution sketch

int minPrice(int startAirport, int endAirport)

- ✓ Dijkstra알고리즘을 이용하여 최단 시간을 구하여 반환한다. 아래 예는 $O(N^2)$.
- ✓ 경로를 구할 수 없는 경우 -1을 반환한다.

```
// 0. init
int i, j, minNode, minCost;
for (i = 0; i < N; ++i) dist[i] = INF, used[i] = 0;
dist[s] = 0;
for (i = 0; i < N; ++i) {
    minCost = INF;
    for (j = 0; j < N; ++j) {    // 1. extract minNode
        if (!used[j] && dist[j] <  minCost)
            minNode = j, minTime = dist[j];
    }
    if (minCost  == INF) break;
    used[minNode] = 1;           // 2. check minNode
    if (minNode == e) return  minCost;
    for (j = 0; j < N; ++j) {    // 3. relaxation
        if (!used[j] && dist[j] > minCost + costTable[minNode][j])
            dist[j] = minCost + costTable[minNode][j];
    }
}
return -1;
```

Solution sketch

[Summary]

- 최단 경로(거리) 알고리즘을 문제해결에 사용할 수 있다.
- 최단 경로 알고리즘에는 Dijkstra, Floyd-Warshall 등이 있다.
  (Bellman-Ford, SPFA, A, ...)*
- 현재까지 Dijkstra 알고리즘으로 풀 수 있는 경우의 문제만 출제됨
  - ✓ $O(N^2)$ 뿐만 아니라 O(ElogV) 방법으로 풀어야 하는 문제도 출제됨.
  - ✓ 앞으로 종종(자주) 나올 것으로 예상되므로 Dijkstra 알고리즘은 확실이 연습 해두자.

[Dijkstra 기출문제]

- 22년 10월 1일 [H2255]물류허브(py)
- 22년 11월 4일 [H2262]주거지검색서비스

Code example

Code example

TS항공노선관리

```
const int LM = 60;
const int INF = 1 << 20;
int N;
int costTable[LM][LM];
int timeTable[LM][LM][24];
int dist[LM], used[LM];

inline int min(int a, int b) { return a < b ? a : b; }

void init(int N) {
    ::N = N;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            costTable[i][j] = INF;
            for (int k = 0; k < 24; ++k)
                timeTable[i][j][k] = INF;
        }
    }
}
```

Code example

TS항공노선관리

```
void add(int s, int e, int st, int travelTime, int mPrice) {
    costTable[s][e] = min(costTable[s][e], mPrice);
    for (int arrivalTime = 0; arrivalTime < 24; ++arrivalTime) {
        int waitingTime = (st - arrivalTime + 24) % 24;
        int travelTimeCost = travelTime + waitingTime;
        timeTable[s][e][arrivalTime] = min(timeTable[s][e][arrivalTime], travelTimeCost);
    }
}
```

Code example


```
int minTravelTime(int s, int e, int startTime) {
    int i, j, minNode, minTime;
    for (i = 0; i < N; ++i) dist[i] = INF, used[i] = 0;
    dist[s] = startTime;
    for (i = 0; i < N; ++i) {
        minTime = INF;
        for (j = 0; j < N; ++j) {
            if (!used[j] && dist[j] < minTime) {
                minNode = j, minTime = dist[j];
            }
        }
        if (minTime == INF) break;
        used[minNode] = 1;
        if (minNode == e) return minTime - startTime;
        for (j = 0; j < N; ++j) {
            if (!used[j] && dist[j] > minTime + timeTable[minNode][j][minTime % 24]) {
                dist[j] = minTime + timeTable[minNode][j][minTime % 24];
            }
        }
    }
    return -1;
}
```

Code example


```
int minPrice(int s, int e) {
    int i, j, minNode, minCost;
    for (i = 0; i < N; ++i) dist[i] = INF, used[i] = 0;
    dist[s] = 0;

    for (i = 0; i < N; ++i) {
        minCost = INF;
        for (j = 0; j < N; ++j) {
            if (!used[j] && dist[j] < minCost) {
                minNode = j, minCost = dist[j];
            }
        }
        if (minCost == INF) break;
        used[minNode] = 1;
        if (minNode == e) return minCost;
        for (j = 0; j < N; ++j) {
            if (!used[j] && dist[j] > minCost + costTable[minNode][j]) {
                dist[j] = minCost + costTable[minNode][j];
            }
        }
    }
    return -1;
}
```

Thank you.