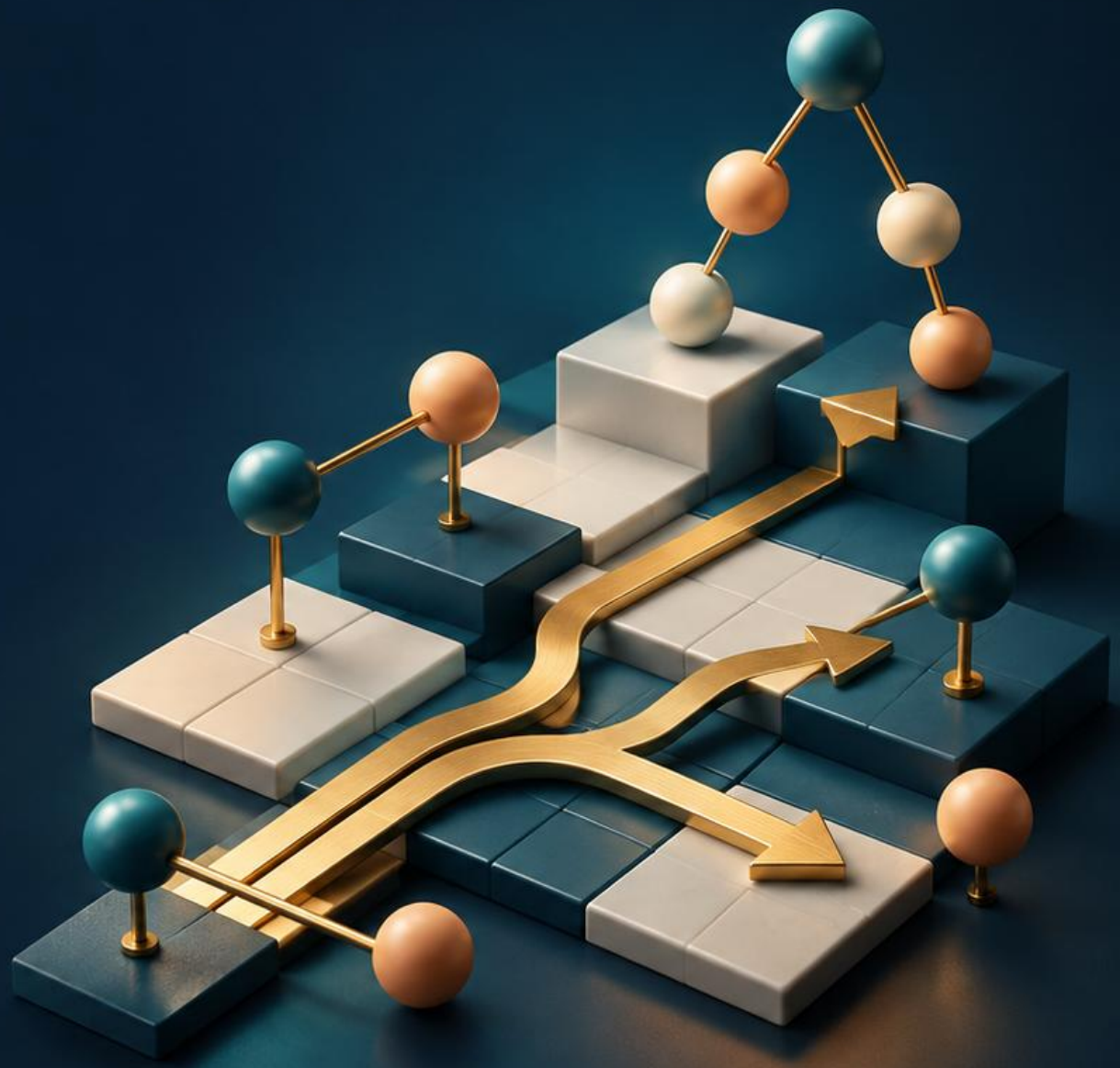
10

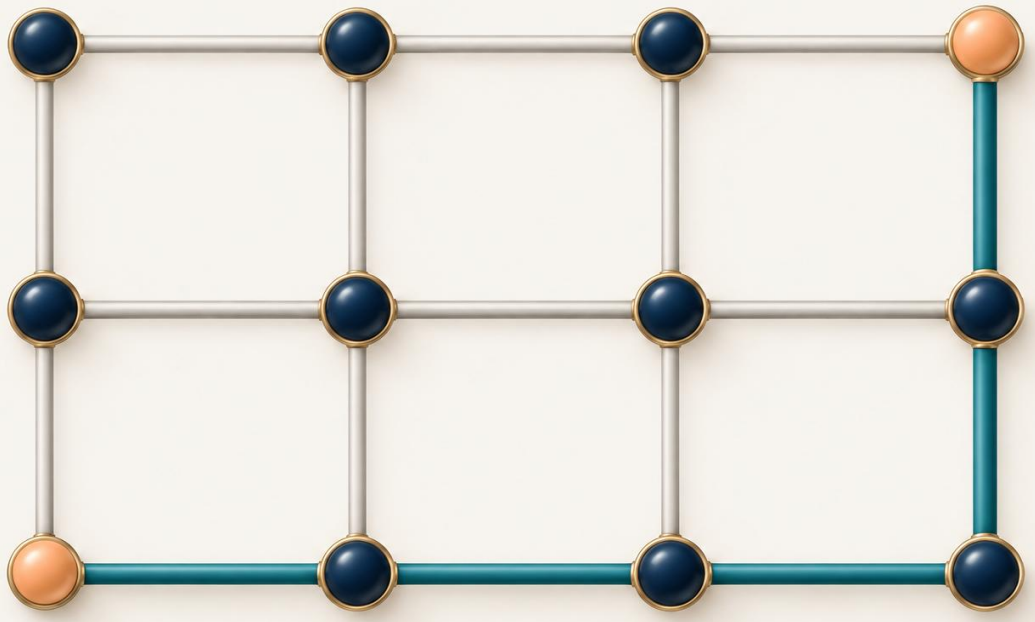
휴리스틱 알고리즘

가까운 도시 먼저 가기처럼 간단한
선택 규칙으로 해를 구하거나
고칩니다.



휴리스틱

가까운 도시 먼저 가기처럼 간단한 선택 규칙으로 해를 구하거나 고칩니다.



오른쪽 세 칸, 위로 두 칸을 간다.

대각선으로 이동할 수 없으면 필요한 이동 수는 3+2=5이다.

10-1 상하좌우 이동 거리

장애물이 없는 격자의 출발점 (x1,y1)에서 도착점 (x2,y2)로 간다. 한 번에 상·하·좌·우 중 한 방향으로 한 칸 움직인다. 대각선 이동은 허용하지 않는다.

01

도착점까지 필요한 최소 이동 횟수를 반환하시오.

`F10-01` `x1, y1, x2, y2`

10-1 입력과 반환값

Python: manhattan(x1, y1, x2, y2)

x1, y1: 출발점 좌표. 각각 −10,000~10,000의 정수
x2, y2: 도착점 좌표. 각각 −10,000~10,000의 정수

반환값

최소 이동 횟수 |x1−x2|+|y1−y2|를 정수로 반환한다.

선택·판정 조건

좌표가 음수여도 같은 격자의 점이다. 거리에는 좌표 차이의 절댓값을 쓴다.
두 점이 같으면 이동하지 않아 답은 0이다.

10-1 가로와 세로 이동 · 같은 점

가로와 세로 이동

출발 (0,0), 도착 (3,2)

반환값 5

가로 세 번과 세로 두 번이 필요하다.

같은 점

출발과 도착이 모두 (2,3)

반환값 0

이동할 필요가 없다.

10-1 음수 좌표 포함 · 대각선 방향

음수 좌표 포함

출발 (2,-1), 도착 (-2,2)

반환값 7

가로 |2−(−2)|=4, 세로 |−1−2|=3으로 합은 7이다.

대각선 방향

출발 (0,0), 도착 (1,1)

반환값 2

오른쪽과 위로 한 번씩 가야 한다. 한 번에 대각선으로 갈 수 없다.

10-1 예제의 계산 과정

출발	도착	가로 차이	세로 차이	합
(0,0)	(3,2)	3	2	**5**
(2,3)	(2,3)	0	0	**0**
(2,−1)	(−2,2)	4	3	**7**

이 거리를 맨해튼 거리라고 부릅니다.

10-1 방향과 상관없이 필요한 칸 수를 더한다

x좌표 차이는 가로로 움직여야 하는 칸 수이고 y좌표 차이는 세로 칸 수이다. 왼쪽이나 아래로 가면 차이가 음수가 될 수 있지만 이동 횟수는 음수가 될 수 없다. 두 차이의 절댓값을 더한다. 이런 거리를 맨해튼 거리라고 부른다.

10-1 코드에서 사용할 값

코드	뜻
x1−x2	방향에 따라 양수나 음수가 나오는 가로 차이
abs(x1−x2)	부호를 없앤 가로 이동 칸 수
abs(y1−y2)	세로 이동 칸 수

절댓값 abs는 음수의 부호를 없애 거리처럼 양수로 읽게 합니다.

10-1 Python

```python
def manhattan(x1, y1, x2, y2):
    return abs(x1-x2) + abs(y1-y2)
```

시간 O(1), 추가 공간 O(1)

10-1 C 함수의 인자

함수 원형

int manhattan(int x1, int y1, int x2, int y2)

배열과 작업 공간

네 인자는 Python과 같은 뜻이다.

10-1 C

```c
int manhattan(int x1, int y1, int x2, int y2) {
    int dx=x1-x2, dy=y1-y2;
    if (dx<0) dx=-dx;
    if (dy<0) dy=-dy;
    return dx+dy;
}
```

시간 O(1), 추가 공간 O(1)

10-2 가까운 도시부터 방문하는 순회

0번 도시에서 출발한다. 현재 도시에서 아직 방문하지 않은 도시 중 이동 비용이 가장 작은 곳으로 간다. 비용이 같으면 번호가 작은 도시를 고른다. 모든 도시를 방문하면 0번으로 돌아온다.

02

이 선택 규칙을 그대로 실행한 왕복 비용을 반환하시오.

`F10-02` `dist`

10-2 입력과 반환값

Python: nearest_neighbor(dist)

dist: 도시 1~9개의 정사각 비용 표. 값은 0~100, 대각선은 0

반환값

가까운 도시부터 방문한 경로의 비용을 정수로 반환한다. 모든 순서를 비교한 최솟값과 다를 수 있다.

선택·판정 조건

모든 이동이 가능하며 방향별 비용은 달라도 된다.
이미 방문한 도시는 다음 후보에서 뺀다. 마지막 귀환 비용은 더한다.
도시가 하나이면 0이다.

10-2 기본 순회 · 동률은 작은 번호

기본 순회

```
dist=[[0,3,1],[3,0,2],[1,2,0]]
```

반환값 6

0에서 2를 고르고 1로 간 뒤 귀환해 1+2+3=6이다.

동률은 작은 번호

```
dist=[[0,1,1],[5,0,1],[2,7,0]]
```

반환값 4

0에서 1과2의 비용이 같아 1을 먼저 고른다. 비용은 1+1+2=4이다.

10-2 가까운 선택의 한계 · 두 도시

가까운 선택의 한계

```
dist=[[0,1,2,9],[1,0,1,9],[2,1,0,50],[9,9,5
0,0]]
```

반환값 61

규칙의 경로 0,1,2,3,0은 61이다. 더 싼 경로가 있어도 규칙대로 계산한 61을 반환한다.

두 도시

```
dist=[[0,5],[8,0]]
```

반환값 13

왕복 비용 5+8이다.

10-2 도시 하나

도시 하나

```
dist=[[0]]
```

반환값



추가로 방문할 도시가 없다.

10-2 예제의 계산 과정

현재 도시	갈 수 있는 도시와 비용	선택	누적 합
0	1: 3, 2: 1	2	**1**
2	1: 2	1	**3**
1	방문 완료. 0으로 귀환: 3	0	**6**

모든 순회를 비교하지 않고, 그 순간 가장 가까운 도시를 하나 고릅니다.

10-2 먼저 가까운 곳을 골라 생기는 결과

네 도시 사례에서는 0에서 비용1인 1을 고르고, 1에서 비용1인 2를 고른다. 남은 3으로 가는 비용은 50이어서 전체 비용은 61이 된다. 0,2,1,3,0은 비용21이지만, 이 함수는 전체 최적 경로를 찾는 함수가 아니라 주어진 가까운 도시 선택 규칙을 구현한다.

10-2 코드에서 사용할 값

변수	뜻
current	지금 있는 도시
used[v]	도시 v를 이미 방문했는지 표시
nxt	이번에 선택한 다음 도시
total+dist[current][0]	마지막 귀환까지 포함한 비용

지금 가까운 곳을 고르는 규칙입니다. 모든 경로 중 최소 비용과 다를 수 있습니다.

10-2 Python (1/2)

```python
def nearest_neighbor(dist):
    n = len(dist)
    used = [False]*n
    used[0] = True
    current = 0
    total = 0
    for step in range(n-1):
        nxt = -1
        for v in range(n):
            if not used[v]:
                if nxt == -1 or dist[current][v] < dist[current][nxt]:
                    nxt = v
```

10-2 Python (2/2)

```
13            total += dist[current][nxt]
14            used[nxt] = True
15            current = nxt
16        return total + dist[current][0]
```

시간 $O(n^2)$, 방문 배열 $O(n)$

10-2 C 함수의 인자

함수 원형

int nearest_neighbor(const int dist[][9], int n)

배열과 작업 공간

dist는 비용 표, n은 도시 수이다.

10-2 C

```
1   int nearest_neighbor(const int dist[][9], int n) {
2       int used[9]={0}, current=0, total=0;
3       used[0]=1;
4       for (int step=0; step<n-1; ++step) {
5           int next=-1;
6           for (int v=0; v<n; ++v)
7               if (!used[v] && (next<0 || dist[current][v]<dist[current][next]))
8                   next=v;
9           total+=dist[current][next];
10          used[next]=1;
11          current=next;
12      }
13      return total+dist[current][0];
14  }
```

시간 $O(n^2)$, 방문 배열 $O(n)$

10-3 새 후보를 받아들일지 판단하기

비용을 줄이는 과정에서 현재 해를 새 후보로 바꿀지 결정한다. 후보 비용이 현재보다 작거나 같으면 항상 받아들인다. 더 비싼 후보는 온도 temp와 주어진 숫자 u를 사용해 아래 규칙으로 판단한다.

03

새 후보를 받아들이면 1, 거부하면 0을 반환하시오.

F10-03 current, candidate, temp, u

10-3 입력과 반환값

Python: accept_move(current, candidate, temp, u)

current, candidate: 현재 비용과 새 후보 비용. 각각 0~10,000의 정수
temp: 온도. 0~1000의 실수
u: 비교에 쓸 숫자. 0 이상 1 미만의 실수

반환값

수락 여부인 정수 1 또는 0을 반환한다. 새 비용을 반환하지 않는다.

선택·판정 조건

candidate≤current이면 temp와 u에 관계없이 1이다.
candidate>current이고 temp=0이면 0이다.
더 비싸고 temp>0이면 p=exp((current−candidate)/temp)를 계산한다. u<p이면 1, 그 외에는 0이다.
새 난수를 만들지 말고 입력 u를 쓴다. 양의 온도에서 u=0이면 수락한다.

10-3 더 싼 후보 · 비용이 같음

더 싼 후보

```
current=10, candidate=8, temp=0, u=0.9
```

반환값 1

비용을 줄이므로 온도가 0이어도
받아들인다.

비용이 같음

```
current=10, candidate=10, temp=0, u=0.99
```

반환값 1

같은 비용도 첫 조건에 따라 수락한다.

10-3 더 비싸고 온도 0 · 작은 u

더 비싸고 온도 0

```
current=10, candidate=12, temp=0, u=0
```

반환값 0

온도 0에서 더 비싼 후보는 거부한다.
u=0이어도 이 조건이 먼저이다.

작은 u

```
current=10, candidate=12, temp=2, u=0.2
```

반환값 1

$p=\exp(-1)\approx 0.368$이고 $0.2<p$이다.

10-3 큰 u · 온도 2

큰 u

같은 비용과 온도, `u=0.8`

반환값 0

0.8은 약 0.368보다 크므로 거부한다.

온도 2

`current=10, candidate=12, temp=2, u=0.5`

반환값 0

0.5는 수락 확률 약 0.368보다 크다.

10-3 온도만 4로 변경

온도만 4로 변경

```
current=10, candidate=12, temp=4, u=0.5
```

반환값

p=exp(−0.5)≈0.607로 커져 이번에는 수락한다.

1

10-3 예제의 계산 과정

후보의 상태	온도	판단
현재보다 작거나 같음	어떤 값이든	**항상 수락 1**
현재보다 큼	0	**거부 0**
현재보다 큼	0보다 큼	**u와 수락 확률 비교**

좋은 후보부터 먼저 처리하고, 더 나쁜 후보일 때만 확률을 계산합니다.

10-3 좋은 후보부터 처리하고 나눗셈은 나중에 한다

조건의 순서가 중요하다. 좋은 후보나 같은 비용의 후보는 먼저 수락하고, 남은 경우에서 온도가 0인지 확인한다. 그다음에만 확률식을 계산하므로 0으로 나누지 않는다. exp는 지수 함수이며 Python의 math.exp, C의 math.h에 있는 exp를 사용한다. 이 한 단계의 판단은 담금질 기법에서 쓰는 수락 규칙이다.

exp는 주어진 수학 함수입니다

Python

import math 뒤 math.exp(값)을 호출합니다.

C

#include <math.h> 뒤 exp(값)을 호출합니다.

아주 작은 확률

양의 온도에서는 확률이 0보다 크므로 u=0이면 수락합니다. 반올림으로 0이 되는 경우도 코드에서 먼저 처리합니다.

10-3 코드에서 사용할 값

변수	예제에서의 뜻
current, candidate	현재 비용 10, 새 비용 12
temp	온도 2. 더 비싼 후보를 받아들이는 정도에 영향
probability	$\exp((10-12)/2)=\exp(-1)\approx 0.368$
u	주어진 0..1 사이 숫자. 새 난수를 만들지 않음

이런 수락 규칙을 쓰는 방법을 담금질 기법이라고 부릅니다.

10-3 Python

```python
import math
def accept_move(current, candidate, temp, u):
    if candidate <= current: return 1
    if temp == 0: return 0
    if u == 0: return 1
    probability = math.exp((current-candidate)/temp)
    return int(u < probability)
```

기본 수학 연산을 상수 시간으로 볼 때 시간 O(1), 공간 O(1)

10-3 C 함수의 인자

함수 원형

int accept_move(int current, int candidate, double temp, double u)

배열과 작업 공간

temp와 u는 double형이다. 나머지 조건은 Python과 같다.

10-3 C

```c
#include <math.h>
int accept_move(int current, int candidate, double temp, double u) {
    if (candidate<=current) return 1;
    if (temp==0) return 0;
    if (u==0) return 1;
    double probability=exp((current-candidate)/temp);
    return u<probability;
}
```

기본 수학 연산을 상수 시간으로 볼 때 시간 O(1), 공간 O(1)

10-4 도시 두 위치를 한 번 교환한 최소 비용

도시를 방문하는 원래 경로 route가 주어진다. 첫 위치의 도시0은 그대로 두고, 나머지 위치 중 두 곳을 골라 한 번 맞바꾼다. 아무것도 바꾸지 않은 원래 경로도 후보로 포함한다.

04

이 후보들의 왕복 비용 중 가장 작은 값을 반환하시오.

`F10-04` `dist, route`

10-4 입력과 반환값

Python: best_swap_cost(dist, route)

dist: 도시 1~9개의 정사각 비용 표. 각 값 0~100, 대각선 0
route: 0으로 시작하며 0~n−1을 한 번씩 포함하는 길이 n의 배열

반환값

원래 경로 또는 한 번 교환한 경로의 최소 비용을 반환한다. 바꾼 경로 자체는 반환하지 않는다.

선택·판정 조건

모든 이동이 가능하고 방향별 비용은 달라도 된다. 마지막 귀환 비용도 포함한다.
각 후보는 원래 경로에서 새로 만든다. 한 번 좋아진 경로에서 계속 교환을 누적하지 않는다.
두 위치를 고를 수 없으면 원래 비용을 반환한다. 입력 route의 내용도 바꾸지 않는다.

10-4 교환해서 개선 · 원래 경로가 더 쌈

교환해서 개선

```
dist=[[0,3,1,8],[3,0,8,2],[1,8,0,4],[8,2,4,
0]], route=[0,1,2,3]
```

반환값 10

2와3의 위치를 바꾼 [0,1,3,2]의 비용은 10이다. 원래 비용은 23이다.

원래 경로가 더 쌈

```
dist=[[0,1,9],[4,0,2],[3,6,0]],
route=[0,1,2]
```

반환값 6

원래 비용은 1+2+3=6이고, 교환하면 9+6+4=19라 그대로 둔다.

10-4 같은 표, 다른 시작 경로 · 교환할 두 위치가 없음

같은 표, 다른 시작 경로

같은 `dist, route=[0,2,1]`

반환값 6

1과2의 위치를 바꾸면 앞 사례의 비용6인 경로가 된다.

교환할 두 위치가 없음

`dist=[[0,5],[8,0]], route=[0,1]`

반환값 13

첫 위치를 고정하면 남은 위치가 하나뿐이라 원래 비용을 반환한다.

10-4 도시 하나

도시 하나

```
dist=[[0]], route=[0]
```

반환값

바꿀 위치도 없고 왕복 비용도 0이다.



교환 예제의 이동 비용 표

아래 표와 원래 경로 [0,1,2,3]을 사용합니다.

출발	도착0	도착1	도착2	도착3
도시0	0	3	1	**8**
도시1	3	0	8	**2**
도시2	1	8	0	**4**
도시3	8	2	4	**0**

원래 비용은 3+8+4+8=23. 2와 3의 위치를 바꾸면 3+2+4+1=10입니다.

10-4 예제의 계산 과정

바꾸는 위치	경로	왕복 비용
안 바꿈	0,1,2,3	**23**
1번과 2번	0,2,1,3	**19**
1번과 3번	0,3,2,1	**23**
2번과 3번	0,1,3,2	**10**

가장 작은 비용은 10입니다. 모든 후보는 원래 경로에서 한 번만 바꿉니다.

10-4 후보 하나를 계산한 뒤 원래 순서로 돌린다

[0,1,2,3]에서 바꿀 수 있는 위치 쌍은 (1,2),(1,3),(2,3)이다. 각 쌍을 바꾼 비용을 계산한 다음 같은 두 위치를 다시 바꿔 복구한다. 복구하지 않으면 다음 후보가 원래 경로에서 두 번 바뀐 경로가 되어 문제의 조건과 달라진다.

10-4 코드에서 사용할 값

코드	뜻
best=cost()	원래 경로 비용 저장
i는 1부터	0번 위치의 출발 도시는 고정
j는 i+1부터	같은 두 위치를 중복 비교하지 않음
교환 코드가 두 번 등장	계산 전 교환, 계산 후 복구

개선된 경로에서 계속 바꾸는 방식이 아니라, 원래 경로의 한 번 교환만 비교합니다.

10-4 Python (1/2)

```python
def best_swap_cost(dist, route):
    route = route[:]
    n = len(route)
    def cost():
        total = 0
        for i in range(n):
            u = route[i]
            v = route[(i+1)%n]
            total += dist[u][v]
        return total
```

10-4 Python (2/2)

```
11        best = cost()
12        for i in range(1, n):
13            for j in range(i+1, n):
14                route[i], route[j] = route[j], route[i]
15                best = min(best, cost())
16                route[i], route[j] = route[j], route[i]
17        return best
```

위치 쌍 $O(n^2)$ × 경로 비용 $O(n)$, 시간 $O(n^3)$, 복사 배열 $O(n)$

10-4 C 함수의 인자

함수 원형

int best_swap_cost(const int dist[][9], int n, const int route[])

배열과 작업 공간

dist는 비용 표, n은 도시 수, route는 길이 n인 방문 순서이다.

10-4 C (1/2)

```c
static int swap_route_cost(const int d[][9], int n, const int r[]) {
    int sum=0;
    for (int i=0; i<n; ++i) sum+=d[r[i]][r[(i+1)%n]];
    return sum;
}
int best_swap_cost(const int dist[][9], int n, const int route[]) {
    int r[9];
    for (int i=0; i<n; ++i) r[i]=route[i];
    int best=swap_route_cost(dist,n,r);
    for (int i=1; i<n; ++i) for (int j=i+1; j<n; ++j) {
        int t=r[i];
        r[i]=r[j];
        r[j]=t;
        int value=swap_route_cost(dist,n,r);
        if (value<best) best=value;
```

10-4 C (2/2)

```
16              t=r[i];
17              r[i]=r[j];
18              r[j]=t;
19          }
20          return best;
21      }
```

위치 쌍 $O(n^2)$ × 경로 비용 $O(n)$, 시간 $O(n^3)$, 복사 배열 $O(n)$