In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```python
data = pd.read_csv("location od cities.csv")
```

In [4]:
```python
data
```

Out[4]:

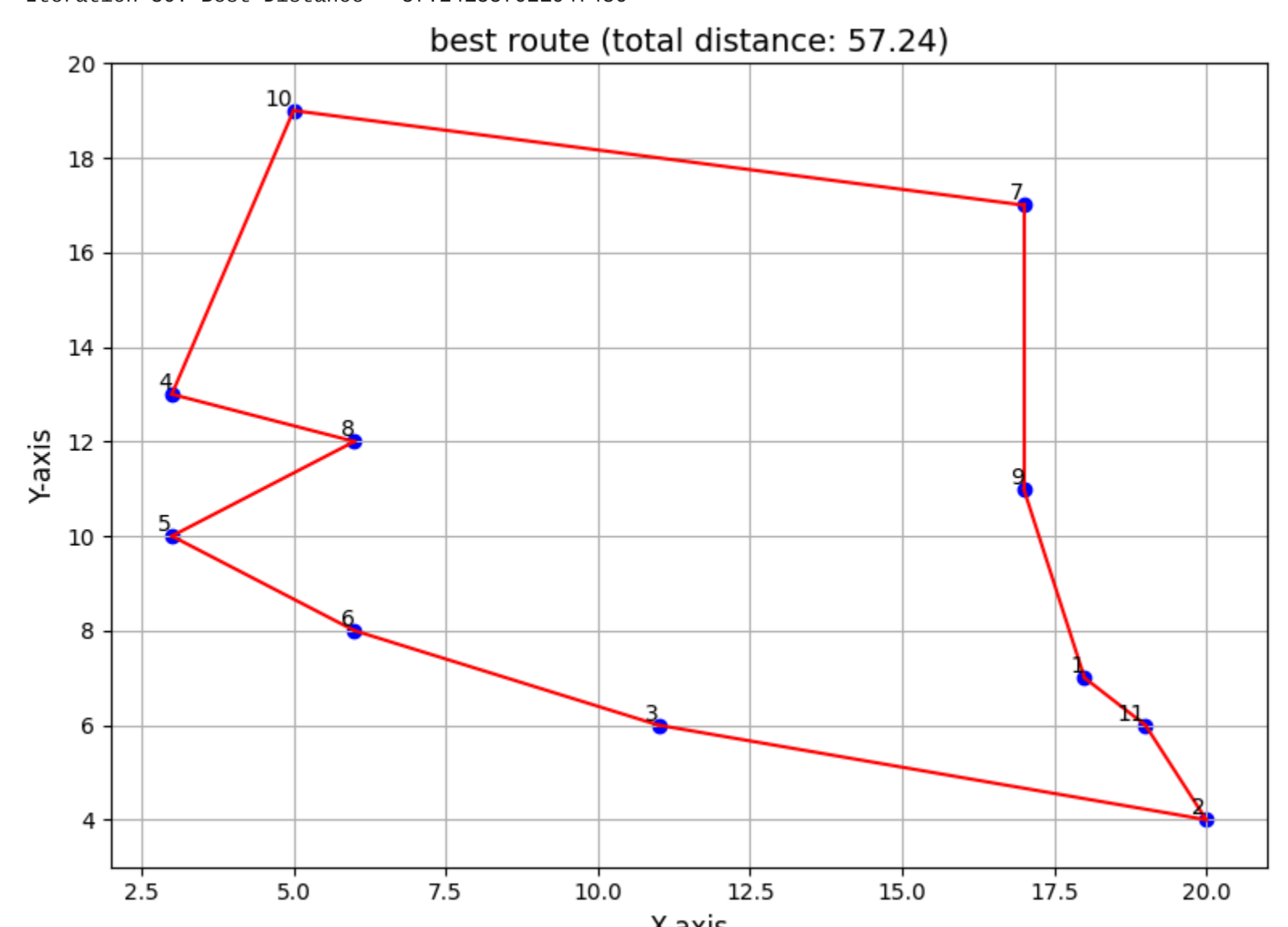
	Cities	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
0	X-axis	18	20	11	3	3	6	17	6	17	5	19
1	Y-axis	7	4	6	13	10	8	17	12	11	19	6

In [2]:
```python
import pandas as pd
import numpy as np
from scipy.spatial.distance import pdist, squareform
import random

def load_city_data(file_path):
    """CSV 파일에서 도시 좌표를 로드합니다."""
    cities_df = pd.read_csv(file_path)
    x_coords = cities_df.iloc[0, 1:].values.astype(float)
    y_coords = cities_df.iloc[1, 1:].values.astype(float)
    city_locations = np.vstack((x_coords, y_coords)).T
    return city_locations

def calculate_distance_matrix(city_locations):
    """도시들 간의 유클리드 거리 행렬을 계산합니다."""
    distance_matrix = squareform(pdist(city_locations, metric='euclidean'))
    return distance_matrix

def calculate_total_distance(route, distance_matrix):
    """주어진 경로의 총 거리를 계산합니다."""
    total_distance = 0
    for i in range(len(route)):
        from_city = route[i - 1]
        to_city = route[i]
        total_distance += distance_matrix[from_city][to_city]
    return total_distance

def generate_initial_solution(num_cities):
    """초기 해를 무작위로 생성합니다."""
    solution = list(range(num_cities))
    random.shuffle(solution)
    return solution

def get_neighbors(solution):
    """현재 해의 이웃 해들을 생성합니다(두 도시의 위치를 교환)."""
    neighbors = []
    for i in range(len(solution)):
        for j in range(i + 1, len(solution)):
            neighbor = solution.copy()
            neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
            neighbors.append(neighbor)
    return neighbors

def tabu_search(distance_matrix, num_cities, tabu_size, num_iterations):
    """탐색 금지 알고리즘을 실행하여 최적의 경로를 찾습니다."""
    # 초기 해 설정
    current_solution = generate_initial_solution(num_cities)
    best_solution = current_solution
    best_distance = calculate_total_distance(best_solution, distance_matrix)

    # 탐색 금지 리스트 초기화
    tabu_list = []

    for iteration in range(num_iterations):
        neighbors = get_neighbors(current_solution)
        neighborhood = []

        for neighbor in neighbors:
            if neighbor not in tabu_list:
                distance = calculate_total_distance(neighbor, distance_matrix)
                neighborhood.append((neighbor, distance))

        if not neighborhood:
            # 모든 이웃이 탐색 금지 리스트에 있는 경우, 탐색 금지 리스트를 비웁니다.
            tabu_list = []
            continue

        # 가장 좋은 이웃 선택
        neighborhood.sort(key=lambda x: x[1])
        best_neighbor, best_neighbor_distance = neighborhood[0]

        # 현재 해 및 최적 해 업데이트
        current_solution = best_neighbor
        if best_neighbor_distance < best_distance:
            best_solution = best_neighbor
            best_distance = best_neighbor_distance

        # 탐색 금지 리스트 업데이트
        tabu_list.append(best_neighbor)
        if len(tabu_list) > tabu_size:
            tabu_list.pop(0)

        print(f"Iteration {iteration + 1}: Best Distance = {best_distance}")

    return best_solution, best_distance

def main():
    # 도시 좌표 데이터 로드
    file_path = 'location od cities.csv'  # CSV 파일의 경로를 지정해주세요.
    city_locations = load_city_data(file_path)

    # 거리 행렬 계산
    distance_matrix = calculate_distance_matrix(city_locations)

    # 탐색 금지 알고리즘 파라미터 설정
    num_cities = len(city_locations)
    tabu_size = 20  # 탐색 금지 리스트의 크기
    num_iterations = 100  # 알고리즘의 반복 횟수

    # 탐색 금지 알고리즘 실행
    best_route, best_distance = tabu_search(distance_matrix, num_cities, tabu_size, num_iterations)

    # 결과 출력
    print("\n최적 경로:", best_route)
    print("최적 경로의 총 거리:", best_distance)

if __name__ == "__main__":
    main()
```

```
Iteration 1: Best Distance = 94.08913423949593
Iteration 2: Best Distance = 89.48122838929231
Iteration 3: Best Distance = 76.49863604230384
Iteration 4: Best Distance = 71.85834585754608
Iteration 5: Best Distance = 63.465221583443984
Iteration 6: Best Distance = 57.38251823801222
Iteration 7: Best Distance = 57.241557021947486
Iteration 8: Best Distance = 57.241557021947486
Iteration 9: Best Distance = 57.241557021947486
Iteration 10: Best Distance = 57.241557021947486
Iteration 11: Best Distance = 57.241557021947486
Iteration 12: Best Distance = 57.241557021947486
Iteration 13: Best Distance = 57.241557021947486
Iteration 14: Best Distance = 57.241557021947486
Iteration 15: Best Distance = 57.241557021947486
Iteration 16: Best Distance = 57.241557021947486
Iteration 17: Best Distance = 57.241557021947486
Iteration 18: Best Distance = 57.241557021947486
Iteration 19: Best Distance = 57.241557021947486
Iteration 20: Best Distance = 57.241557021947486
Iteration 21: Best Distance = 57.241557021947486
Iteration 22: Best Distance = 57.241557021947486
Iteration 23: Best Distance = 57.241557021947486
Iteration 24: Best Distance = 57.241557021947486
Iteration 25: Best Distance = 57.241557021947486
Iteration 26: Best Distance = 57.241557021947486
Iteration 27: Best Distance = 57.241557021947486
Iteration 28: Best Distance = 57.241557021947486
Iteration 29: Best Distance = 57.241557021947486
Iteration 30: Best Distance = 57.241557021947486
Iteration 31: Best Distance = 57.241557021947486
Iteration 32: Best Distance = 57.241557021947486
Iteration 33: Best Distance = 57.241557021947486
Iteration 34: Best Distance = 57.241557021947486
Iteration 35: Best Distance = 57.241557021947486
Iteration 36: Best Distance = 57.241557021947486
Iteration 37: Best Distance = 57.241557021947486
Iteration 38: Best Distance = 57.241557021947486
Iteration 39: Best Distance = 57.241557021947486
Iteration 40: Best Distance = 57.241557021947486
Iteration 41: Best Distance = 57.241557021947486
Iteration 42: Best Distance = 57.241557021947486
Iteration 43: Best Distance = 57.241557021947486
Iteration 44: Best Distance = 57.241557021947486
Iteration 45: Best Distance = 57.241557021947486
Iteration 46: Best Distance = 57.241557021947486
Iteration 47: Best Distance = 57.241557021947486
Iteration 48: Best Distance = 57.241557021947486
Iteration 49: Best Distance = 57.241557021947486
Iteration 50: Best Distance = 57.241557021947486
Iteration 51: Best Distance = 57.241557021947486
Iteration 52: Best Distance = 57.241557021947486
Iteration 53: Best Distance = 57.241557021947486
Iteration 54: Best Distance = 57.241557021947486
Iteration 55: Best Distance = 57.241557021947486
Iteration 56: Best Distance = 57.241557021947486
Iteration 57: Best Distance = 57.241557021947486
Iteration 58: Best Distance = 57.241557021947486
Iteration 59: Best Distance = 57.241557021947486
Iteration 60: Best Distance = 57.241557021947486
Iteration 61: Best Distance = 57.241557021947486
Iteration 62: Best Distance = 57.241557021947486
Iteration 63: Best Distance = 57.241557021947486
Iteration 64: Best Distance = 57.241557021947486
Iteration 65: Best Distance = 57.241557021947486
Iteration 66: Best Distance = 57.241557021947486
Iteration 67: Best Distance = 57.241557021947486
Iteration 68: Best Distance = 57.241557021947486
Iteration 69: Best Distance = 57.241557021947486
Iteration 70: Best Distance = 57.241557021947486
Iteration 71: Best Distance = 57.241557021947486
Iteration 72: Best Distance = 57.241557021947486
Iteration 73: Best Distance = 57.241557021947486
Iteration 74: Best Distance = 57.241557021947486
Iteration 75: Best Distance = 57.241557021947486
Iteration 76: Best Distance = 57.241557021947486
Iteration 77: Best Distance = 57.241557021947486
Iteration 78: Best Distance = 57.241557021947486
Iteration 79: Best Distance = 57.241557021947486
Iteration 80: Best Distance = 57.241557021947486
Iteration 81: Best Distance = 57.241557021947486
Iteration 82: Best Distance = 57.241557021947486
Iteration 83: Best Distance = 57.241557021947486
Iteration 84: Best Distance = 57.241557021947486
Iteration 85: Best Distance = 57.241557021947486
Iteration 86: Best Distance = 57.241557021947486
Iteration 87: Best Distance = 57.241557021947486
Iteration 88: Best Distance = 57.241557021947486
Iteration 89: Best Distance = 57.241557021947486
Iteration 90: Best Distance = 57.241557021947486
Iteration 91: Best Distance = 57.241557021947486
Iteration 92: Best Distance = 57.241557021947486
Iteration 93: Best Distance = 57.241557021947486
Iteration 94: Best Distance = 57.241557021947486
Iteration 95: Best Distance = 57.241557021947486
Iteration 96: Best Distance = 57.241557021947486
Iteration 97: Best Distance = 57.241557021947486
Iteration 98: Best Distance = 57.241557021947486
Iteration 99: Best Distance = 57.241557021947486
Iteration 100: Best Distance = 57.241557021947486

최적 경로: [7, 4, 5, 2, 1, 10, 0, 8, 6, 9, 3]
최적 경로의 총 거리: 57.241557021947486
```

In [3]:
```python
import pandas as pd
import numpy as np
from scipy.spatial.distance import pdist, squareform
import random
import matplotlib.pyplot as plt

def plot_route(city_locations, best_route, best_distance):
    """도시들 간의 최적 경로를 시각화합니다."""
    plt.figure(figsize=(8, 6))

    # 도시들을 플롯
    x_coords = city_locations[:, 0]
    y_coords = city_locations[:, 1]
    plt.scatter(x_coords, y_coords, c='blue', marker='o')

    # 도시 번호를 표시
    for i, (x, y) in enumerate(zip(x_coords, y_coords)):
        plt.text(x, y, str(i+1), fontsize=10, ha='right', va='bottom')  # 도시 번호를 1부터 시작

    # 최적 경로를 선으로 연결 (경로는 0부터 시작하는 인덱스를 사용)
    for i in range(len(best_route)):
        start_city = best_route[i - 1] - 1
        end_city = best_route[i] - 1
        plt.plot([x_coords[start_city], x_coords[end_city]], [y_coords[start_city], y_coords[end_city]], 'r-', linewidth=1.5)

    plt.title(f'best route (total distance: {best_distance:.2f})', fontsize=14)
    plt.xlabel('X-axis', fontsize=12)
    plt.ylabel('Y-axis', fontsize=12)
    plt.grid(True)

    # X, Y 축의 범위를 자동 조정하여 모든 점이 잘 보이도록 설정
    plt.xlim(min(x_coords) - 1, max(x_coords) + 1)
    plt.ylim(min(y_coords) - 1, max(y_coords) + 1)

    plt.tight_layout()  # 레이아웃을 조정하여 모든 요소가 잘 보이도록 설정

    # 전체보기 명령어를 사용하여 전체 화면으로 그래프 표시
    plt.show()

def tabu_search(distance_matrix, num_cities, tabu_size, num_iterations):
    """탐색 금지 알고리즘을 실행하여 최적의 경로를 찾습니다."""
    # 초기 해 설정
    current_solution = generate_initial_solution(num_cities)
    best_solution = current_solution
    best_distance = calculate_total_distance(best_solution, distance_matrix)

    # 탐색 금지 리스트 초기화
    tabu_list = []

    for iteration in range(num_iterations):
        neighbors = get_neighbors(current_solution)
        neighborhood = []

        for neighbor in neighbors:
            if neighbor not in tabu_list:
                distance = calculate_total_distance(neighbor, distance_matrix)
                neighborhood.append((neighbor, distance))

        if not neighborhood:
            # 모든 이웃이 탐색 금지 리스트에 있는 경우, 탐색 금지 리스트를 비웁니다.
            tabu_list = []
            continue

        # 가장 좋은 이웃 선택
        neighborhood.sort(key=lambda x: x[1])
        best_neighbor, best_neighbor_distance = neighborhood[0]

        # 현재 해 및 최적 해 업데이트
        current_solution = best_neighbor
        if best_neighbor_distance < best_distance:
            best_solution = best_neighbor
            best_distance = best_neighbor_distance

        # 탐색 금지 리스트 업데이트
        tabu_list.append(best_neighbor)
        if len(tabu_list) > tabu_size:
            tabu_list.pop(0)

        print(f"Iteration {iteration + 1}: Best Distance = {best_distance}")

    # 도시 번호를 1부터 시작하도록 조정
    best_solution = [city + 1 for city in best_solution]

    return best_solution, best_distance

def main():
    # 도시 좌표 데이터 로드
    file_path = 'location od cities.csv'
    city_locations = load_city_data(file_path)

    # 거리 행렬 계산
    distance_matrix = calculate_distance_matrix(city_locations)

    # 탐색 금지 알고리즘 파라미터 설정
    num_cities = len(city_locations)
    tabu_size = 20  # 탐색 금지 리스트의 크기
    num_iterations = 80  # 알고리즘의 반복 횟수

    # 타부금지 알고리즘 실행
    best_route, best_distance = tabu_search(distance_matrix, num_cities, tabu_size, num_iterations)

    # 최적 경로 시각화
    plot_route(city_locations, best_route, best_distance)

    print("\n최적 경로:", best_route)
    print("최적 경로의 총 거리:", best_distance)

if __name__ == "__main__":
    main()
```

```
Iteration 1: Best Distance = 80.33866626429624
Iteration 2: Best Distance = 71.37411463419154
Iteration 3: Best Distance = 68.07328783093249
Iteration 4: Best Distance = 64.99348925776597
Iteration 5: Best Distance = 59.87309950142853
Iteration 6: Best Distance = 57.241557021947486
Iteration 7: Best Distance = 57.241557021947486
Iteration 8: Best Distance = 57.241557021947486
Iteration 9: Best Distance = 57.241557021947486
Iteration 10: Best Distance = 57.241557021947486
Iteration 11: Best Distance = 57.241557021947486
Iteration 12: Best Distance = 57.241557021947486
Iteration 13: Best Distance = 57.241557021947486
Iteration 14: Best Distance = 57.241557021947486
Iteration 15: Best Distance = 57.241557021947486
Iteration 16: Best Distance = 57.241557021947486
Iteration 17: Best Distance = 57.241557021947486
Iteration 18: Best Distance = 57.241557021947486
Iteration 19: Best Distance = 57.241557021947486
Iteration 20: Best Distance = 57.241557021947486
Iteration 21: Best Distance = 57.241557021947486
Iteration 22: Best Distance = 57.241557021947486
Iteration 23: Best Distance = 57.241557021947486
Iteration 24: Best Distance = 57.241557021947486
Iteration 25: Best Distance = 57.241557021947486
Iteration 26: Best Distance = 57.241557021947486
Iteration 27: Best Distance = 57.241557021947486
Iteration 28: Best Distance = 57.241557021947486
Iteration 29: Best Distance = 57.241557021947486
Iteration 30: Best Distance = 57.241557021947486
Iteration 31: Best Distance = 57.241557021947486
Iteration 32: Best Distance = 57.241557021947486
Iteration 33: Best Distance = 57.241557021947486
Iteration 34: Best Distance = 57.241557021947486
Iteration 35: Best Distance = 57.241557021947486
Iteration 36: Best Distance = 57.241557021947486
Iteration 37: Best Distance = 57.241557021947486
Iteration 38: Best Distance = 57.241557021947486
Iteration 39: Best Distance = 57.241557021947486
Iteration 40: Best Distance = 57.241557021947486
Iteration 41: Best Distance = 57.241557021947486
Iteration 42: Best Distance = 57.241557021947486
Iteration 43: Best Distance = 57.241557021947486
Iteration 44: Best Distance = 57.241557021947486
Iteration 45: Best Distance = 57.241557021947486
Iteration 46: Best Distance = 57.241557021947486
Iteration 47: Best Distance = 57.241557021947486
Iteration 48: Best Distance = 57.241557021947486
Iteration 49: Best Distance = 57.241557021947486
Iteration 50: Best Distance = 57.241557021947486
Iteration 51: Best Distance = 57.241557021947486
Iteration 52: Best Distance = 57.241557021947486
Iteration 53: Best Distance = 57.241557021947486
Iteration 54: Best Distance = 57.241557021947486
Iteration 55: Best Distance = 57.241557021947486
Iteration 56: Best Distance = 57.241557021947486
Iteration 57: Best Distance = 57.241557021947486
Iteration 58: Best Distance = 57.241557021947486
Iteration 59: Best Distance = 57.241557021947486
Iteration 60: Best Distance = 57.241557021947486
Iteration 61: Best Distance = 57.241557021947486
Iteration 62: Best Distance = 57.241557021947486
Iteration 63: Best Distance = 57.241557021947486
Iteration 64: Best Distance = 57.241557021947486
Iteration 65: Best Distance = 57.241557021947486
Iteration 66: Best Distance = 57.241557021947486
Iteration 67: Best Distance = 57.241557021947486
Iteration 68: Best Distance = 57.241557021947486
Iteration 69: Best Distance = 57.241557021947486
Iteration 70: Best Distance = 57.241557021947486
Iteration 71: Best Distance = 57.241557021947486
Iteration 72: Best Distance = 57.241557021947486
Iteration 73: Best Distance = 57.241557021947486
Iteration 74: Best Distance = 57.241557021947486
Iteration 75: Best Distance = 57.241557021947486
Iteration 76: Best Distance = 57.241557021947486
Iteration 77: Best Distance = 57.241557021947486
Iteration 78: Best Distance = 57.241557021947486
Iteration 79: Best Distance = 57.241557021947486
Iteration 80: Best Distance = 57.241557021947486
```




```
최적 경로: [3, 2, 11, 1, 9, 7, 10, 4, 8, 5, 6]
최적 경로의 총 거리: 57.241557021947486
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: