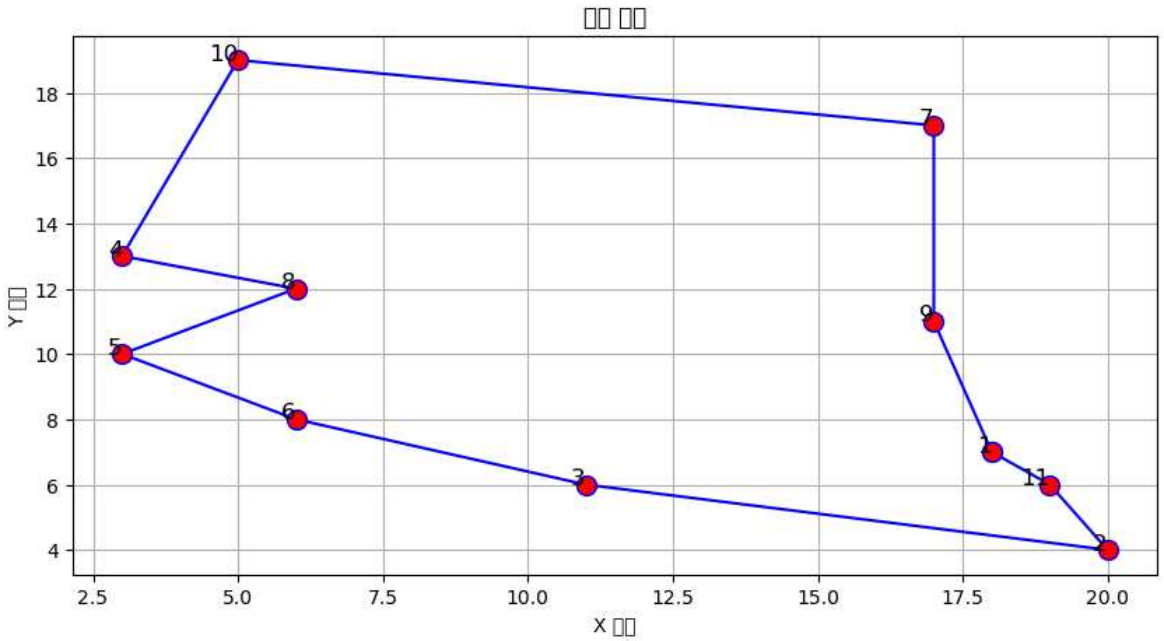
In [ ]:

In [53]:
```
import random
import math
import matplotlib.pyplot as plt

# 초기화 함수
def initialize_tabu_search(start_node, tabu_size, aspiration_level, num_nodes):
    # 단계 1.1: 타부 속성, 타부 목록 크기, 열망수준과 종료조건을 결정하고, 타부 목
    current_solution = [start_node]
    tabu_list = []
    best_cost = float('inf')
    remaining_nodes = list(range(num_nodes))
    remaining_nodes.remove(start_node)
    random.shuffle(remaining_nodes)
    # 단계 1.2: 초기 가능해를 생성한다.
    current_solution.extend(remaining_nodes)
    return current_solution, tabu_list, best_cost, aspiration_level, tabu_size

# 이웃 해 생성 함수
def get_neighbors(current_solution, num_nodes):
    neighbors = []
    for i in range(1, len(current_solution) - 1):
        for j in range(i + 1, len(current_solution)):
            neighbor = current_solution[:]
            neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
            neighbors.append(neighbor)
    return neighbors

# 적합도 평가 함수
def evaluate_solution(solution, graph):
    cost = 0
    for i in range(len(solution) - 1):
        cost += graph[solution[i]][solution[i + 1]]
    cost += graph[solution[-1]][solution[0]]  # 마지막 도시에서 시작 도시로 돌아
    return cost

# 허용 이동 집합 필터링 함수
def filter_neighbors(neighbors, tabu_list, aspiration_level, best_cost, graph):
    allowed_moves = []
    for neighbor in neighbors:
        cost = evaluate_solution(neighbor, graph)
        # 단계 2.2: 열망수준을 만족하지 못하는 타부 이동을 허용 이동 집합에서 제
        if neighbor not in tabu_list or cost < best_cost - aspiration_level:
            allowed_moves.append((neighbor, cost))
    return allowed_moves

# 현재 해와 최선 해 갱신 함수
def update_current_and_best_solution(allowed_moves, current_solution, best_solut
    # 단계 3.1: 허용 이동 집합에서 최우수 해를 현재 해로 둔다.
    best_move = min(allowed_moves, key=lambda x: x[1])
    best_neighbor, best_neighbor_cost = best_move
    current_solution = best_neighbor
    # 단계 3.2: 현재 해가 최선 해보다 우수하면 최선 해를 현재 해로 갱신한다.
    if best_neighbor_cost < best_cost:
        best_solution = best_neighbor
        best_cost = best_neighbor_cost
    return current_solution, best_solution, best_cost
```

```
# 타부 목록 갱신 함수
def update_tabu_list(tabu_list, current_solution, tabu_list_size):
    # 단계 4: 타부 목록 갱신
    # 새로운 현재 해의 이동 속성을 타부 목록에 기록하고, 타부 목록에 저장된 타부
    tabu_list.append(current_solution)
    if len(tabu_list) > tabu_list_size:
        tabu_list.pop(0)
    return tabu_list

# 타부 탐색 알고리즘
def tabu_search(graph, start_node, num_nodes, max_iter, tabu_size, aspiration_le
    # 단계 1: 초기화
    current_solution, tabu_list, best_cost, aspiration_level, tabu_list_size = i
    best_solution = current_solution[:]

    # 단계 2: 이웃 해 생성과 적합도 평가
    for _ in range(max_iter):
        # 2.1: 모든 이웃 해의 적합도를 평가하고, 모든 이웃 해를 허용 이동 집합으
        neighbors = get_neighbors(current_solution, num_nodes)
        allowed_moves = filter_neighbors(neighbors, tabu_list, aspiration_level,

        if not allowed_moves:
            break

        # 단계 3: 현재 해와 최선 해 갱신
        current_solution, best_solution, best_cost = update_current_and_best_sol
        # 단계 4: 타부 목록 갱신
        tabu_list = update_tabu_list(tabu_list, current_solution, tabu_list_size
        # 단계 5: 종료 조건을 검사하여 종료 조건이 만족되면 종료, 그렇지 않으면 단
        if not allowed_moves:
            break

    return best_solution, best_cost

# 도시 위치를 기반으로 거리 그래프 생성
def calculate_distance_graph(locations, num_nodes):
    graph = [[0] * num_nodes for _ in range(num_nodes)]
    for i in range(num_nodes):
        for j in range(num_nodes):
            if i != j:
                graph[i][j] = math.sqrt((locations[i][0] - locations[j][0]) ** 2
    return graph

# 경로 시각화
def plot_path(solution, cities):
    x = [cities[i][0] for i in solution] + [cities[solution[0]][0]]
    y = [cities[i][1] for i in solution] + [cities[solution[0]][1]]

    plt.figure(figsize=(10, 5))
    plt.plot(x, y, 'o-', color='blue', markerfacecolor='red', markersize=10)
    for i, (cx, cy) in enumerate(cities):
        plt.text(cx, cy, f'{i+1}', fontsize=12, ha='right')
    plt.title('최적 경로')
    plt.xlabel('X 좌표')
    plt.ylabel('Y 좌표')
    plt.grid(True)
    plt.show()

# 주어진 좌표 값
```

```
data_string = """Cities\t#1\t#2\t#3\t#4\t#5\t#6\t#7\t#8\t#9\t#10\t#11
X-axis\t18\t20\t11\t3\t3\t6\t17\t6\t17\t5\t19
Y-axis\t7\t4\t6\t13\t10\t8\t17\t12\t11\t19\t6
"""

# 주어진 형식의 문자열에서 도시 좌표 읽기
def read_cities_from_custom_string(data_string):
    lines = data_string.strip().split('\n')
    x_coords = list(map(float, lines[1].split()[1:]))
    y_coords = list(map(float, lines[2].split()[1:]))
    cities = [(x, y) for x, y in zip(x_coords, y_coords)]
    return cities

# 실행 코드
def main():
    cities = read_cities_from_custom_string(data_string)
    num_nodes = len(cities)
    graph = calculate_distance_graph(cities, num_nodes)
    start_node = 0  # 시작 노드를 설정
    tabu_size = 7  # 타부 목록 크기 증가
    max_iter = 300  # 반복 횟수 조정
    aspiration_level = 15  # 열망 기준 설정

    # 타부 탐색 알고리즘 실행
    best_solution, best_cost = tabu_search(graph, start_node, num_nodes, max_ite

    # 단계 4: 결과 출력
    # 4.1 최적 솔루션 출력
    print(f"최적의 해: {best_solution}")
    print(f"최적의 해의 비용: {best_cost}")

    # 4.2 최적 경로 시각화
    plot_path(best_solution, cities)

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

최적의 해: [0, 10, 1, 2, 5, 4, 7, 3, 9, 6, 8]
최적의 해의 비용: 57.241557021947486



In [ ]: