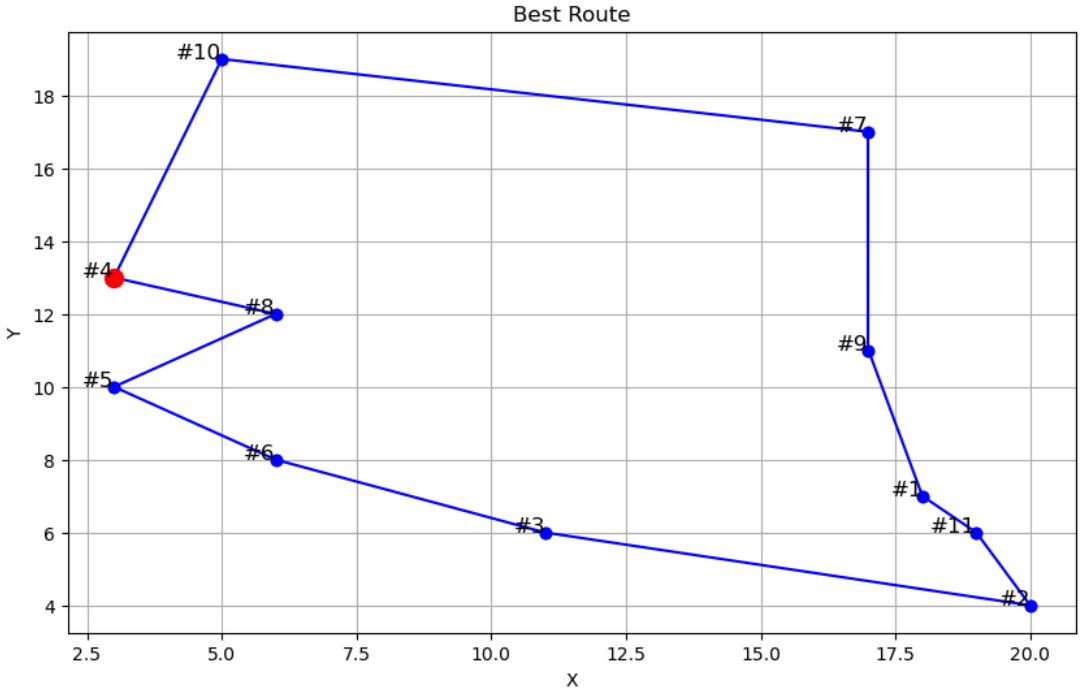
In [2]:
```python
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt

data = {'Cities': ['#1', '#2', '#3', '#4', '#5', '#6', '#7', '#8', '#9', '#10', '#11'],
    'X': [18, 20, 11, 3, 3, 6, 17, 6, 17, 5, 19],
    'Y': [7, 4, 6, 13, 10, 8, 17, 12, 11, 19, 6]}

dataframe = pd.DataFrame(data)

def calculate(p1, j1, p2, j2):  # 두 지점 거리 계산 함수
    return np.sqrt((p2 - p1) ** 2 + (j2 - j1) ** 2)

lencity = len(dataframe)  # 도시의 개수(길이)를 변수로 저장
distance = np.zeros((lencity, lencity))  # 도시 간 거리 행렬을 0으로 초기화

for i in range(lencity):  # 모든 도시 간 거리 계산
    for j in range(lencity):
        if i != j:  # 같은 도시 간의 거리는 계산하지 않음
            distance[i][j] = calculate(dataframe.loc[i, 'X'], dataframe.loc[i, 'Y'], dataframe.loc[j, 'X'], dataframe.loc[j, 'Y'])

def firstsun():
    route = random.sample(range(lencity), lencity)  # 도시의 인덱스를 임의로 선택
    return route

def total_distance(route):  # 주어진 경로의 총 거리를 계산하는 함수
    total_distance = 0
    for i in range(lencity - 1):  # 경로 상의 모든 연속된 도시 간 거리 합산
        total_distance += distance[route[i]][route[i + 1]]
    total_distance += distance[route[-1]][route[0]]  # 마지막 도시에서 첫 도시로 돌아오는 거리 추가
    return total_distance

def neighborsun(route):  # 주어진 경로에서 가능한 모든 이웃 경로를 생성하는 함수
    neighbors = []
    for i in range(lencity):  # 경로 상의 두 도시 위치를 바꿔서 이웃 생성
        for k in range(i + 1, lencity):
            copyroute = route[:]  # 경로를 복사
            copyroute[i], copyroute[k] = copyroute[k], copyroute[i]  # 두 도시 위치 교환
            neighbors.append(copyroute)  # 이웃 경로를 리스트에 추가
    return neighbors

def tabu_search(banbok, rangenumber):  # 타부 서치 알고리즘
    current_route = firstsun()  # 초기 경로 생성
    top_route = current_route[:]  # 초기 경로를 최적 경로로 설정
    top_distance = total_distance(current_route)  # 초기 경로의 거리를 최적 거리로 설정

    tabu_list = []  # 타부 리스트 초기화
    distance_record = []  # 거리 기록을 위한 리스트

    for k in range(banbok):  # 지정된 횟수만큼 반복
        neighbors = neighborsun(current_route)  # 이웃 경로 생성
        better_neighbor = None
        better_neighbor_distance = float('inf')  # 이웃 경로의 최적 거리를 무한대로 초기화(그러면 뒤에 계산할 이웃해는 무조건 더작으니까 초

        for x in neighbors:  # 각 이웃 경로에 대해 반복
            neighbor_distance = total_distance(x)  # 이웃 경로의 총 거리 계산
            if x not in tabu_list or neighbor_distance < top_distance:  # 이웃 경로가 타부 리스트에 없거나, 열망 기준을 만족하는 경우
                if neighbor_distance < better_neighbor_distance:  # 더 좋은 이웃 경로가 발견되면 갱신
                    better_neighbor = x
                    better_neighbor_distance = neighbor_distance

        current_route = better_neighbor  # 최적의 이웃 경로로 현재 경로 갱신

        if better_neighbor_distance < top_distance:    # 최적의 이웃 경로가 현재까지의 최적 경로보다 나으면 갱신
            top_route = better_neighbor
            top_distance = better_neighbor_distance

        tabu_list.append(current_route)  # 타부 리스트에 현재 경로 추가
        if len(tabu_list) > rangenumber:  # 타부 리스트가 일정 크기를 초과하면 오래된 경로 삭제
            tabu_list.pop(0) # pop은 리스트의 첫번째 요소를 제거하고 반환하는 역할을 함

        # 거리 기록
        distance_record.append((k, top_distance))

    # 기록된 거리 데이터를 데이터프레임으로 변환
    distance_df = pd.DataFrame(distance_record, columns=['banbok', 'Best Distance'])

    return top_route, top_distance, distance_df  # 최적 경로와 그 경로의 거리, 기록된 거리 반환

best_route, top_distance, distance_df  = tabu_search(50, 10)  # 타부 서치 알고리즘 실행, 반복 횟수 500, 타부 리스트 크기 50
best_route_cities = [dataframe['Cities'][i] for i in best_route]  # 최적 경로의 도시 인덱스를 도시 이름으로 변환

print("Best Route:", best_route_cities)  # 최적 경로 출력
```

```
print("Best Distance:", top_distance)  # 최적 거리 출력
print(distance_df) # 이동 거리 기록 데이터프레임 출력
```

```
Best Route: ['#4', '#10', '#7', '#9', '#1', '#11', '#2', '#3', '#6', '#5', '#8']
Best Distance: 57.24155702194748
    banbok  Best Distance
0        0      87.879804
1        1      70.816900
2        2      58.546477
3        3      58.213350
4        4      57.241557
5        5      57.241557
6        6      57.241557
7        7      57.241557
8        8      57.241557
9        9      57.241557
10      10      57.241557
11      11      57.241557
12      12      57.241557
13      13      57.241557
14      14      57.241557
15      15      57.241557
16      16      57.241557
17      17      57.241557
18      18      57.241557
19      19      57.241557
20      20      57.241557
21      21      57.241557
22      22      57.241557
23      23      57.241557
24      24      57.241557
25      25      57.241557
26      26      57.241557
27      27      57.241557
28      28      57.241557
29      29      57.241557
30      30      57.241557
31      31      57.241557
32      32      57.241557
33      33      57.241557
34      34      57.241557
35      35      57.241557
36      36      57.241557
37      37      57.241557
38      38      57.241557
39      39      57.241557
40      40      57.241557
41      41      57.241557
42      42      57.241557
43      43      57.241557
44      44      57.241557
45      45      57.241557
46      46      57.241557
47      47      57.241557
48      48      57.241557
49      49      57.241557
```

In [3]:

```
# 최적 경로 시각화 함수
def plot_best_route(best_route, df, top_distance):
    plt.figure(figsize=(10, 6))

    # 도시들의 X, Y 좌표
    X_axis = df['X'].values
    Y_axis = df['Y'].values

    # 최적 경로 따라 도시끼리 선 만들기
    for i in range(len(best_route)):
        start_city = best_route[i]
        end_city = best_route[(i + 1) % len(best_route)]  # 마지막 도시에서 첫 도시로 돌아오는 경로도 포함
        plt.plot([X_axis[start_city], X_axis[end_city]], [Y_axis[start_city], Y_axis[end_city]], 'bo-')
        plt.text(X_axis[start_city], Y_axis[start_city], df['Cities'][start_city], fontsize=12, ha='right')

    # 시작도시 표시
    start_city = best_route[0]
    plt.plot(X_axis[start_city], Y_axis[start_city], "ro", markersize=10)  # 첫 도시를 빨간색으로 표시

    plt.title(f'Best Route')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)
    plt.show()

# 최적 경로 시각화 함수 호출
plot_best_route(best_route, dataframe, top_distance)
```


Best Route
#10
#7
#4
#8
#9
#5
#6
#1
#11
#3
#2
Y
X
18
16
14
12
10
8
6
4
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0