Project Report
Group 5

1. Algorithm we choose

   We choose Dynamic Programming, Greedy and Simulated annealing algorithms to solve TSP.

   In dynamic programming, a series of optimal decisions are made by using the principle of optimality. The principle of optimality: if the optimal total solution, then the solution to the k_{th} stage is also optimal. With the principle of optimality is guaranteed that at some stage of decision making is the right decision for the later stages. The essence of dynamic programming is to remove a small part of a problem at every step, and then solve the smaller problems and use the results of the settlement to remedy the solution is added back to the issue in the next step.

   Greedy algorithm is to find the best solution on each step. For tsp problem, greedy can find a shortest tour form start city to visited all cities and back to start city without visited same city twice.

   Simulated annealing is a randomized algorithm which find the optimal answer from a set of solutions in the required time. When simulated annealing algorithm is used to solve TSP, it is different from the greedy algorithm which always take the minimum path and set it with optimal. The annealing algorithm may choose the “worse” solution (longer than the current shortest path) rather than always find the shortest path, since if we always choose the minimum path, we may get the partial optimal solution rather than the best one eventually. The rule of choosing “worse” solution is called annealing.

2. How it work

   DP:

   Denote the cities by 1,...,n, the salesman’s hometown being 1, and let D = (d_{ij}))be the matrix of intercity distances. The goal is to design a tour that starts and ends at 1, includes all other cities exactly once, and has minimum total length. Figure shows an example involving five cities.

   Greedy:

   A greedy algorithm for the traveling salesman problem: Pick an arbitrary city and call it city 1. Find a city with the smallest distance from city 1, and call it city 2. Find a city in the rest of the n − 2 cities with the smallest distance from city 2 . . .

Output the tour: City1→City2→⋯→Cityn→City1.

Simulated annealing:
Let T_min be the minimum temperature, T_int be the initial temperature, and k be the coefficient of temperature decrease. In the theory of thermodynamics, P(dE) = exp(dE/kT), in which dE is the different energy between the current and dropped temperature. From the formula, it is not hard to find that when current temperature is high, the possibility of dropping temperature in energy size of dE (called "P(dE)") is bigger than the lower current temperature. Since this is annealing, dE < 0, dE/kT < 0. Thus, 0 < P(dE) < 1. In TSP, when a length of path incoming, the algorithm always check the temperature, if it is bigger than T_min, and this path is longer than current optimal path, then using the P(dE) formula to find the possibility of using this path or not.

3. Pseudocode
DP:

$$C(S,j) = \min_{i \in S: i \neq j} C(S - \{j\}, i) + d_{ij}$$

The sub-problems are ordered by |S|. Here's the code.

```
C({1},1) = 0
for s = 2 to n:
  for all subsets S ⊆ {1,2,...,n} of size s and containing 1:
    C(S,1) = ∞
    for all j ∈ S, j ≠ 1 :
      C(S, j) = min{C(S − {j}, i) + d_ij : i ∈ S, i ≠ j}
return min_j C({1, ..., n}, j) + d_j1
```

Greedy:

```
def nearestneighbor(cities)
    start_vertex = vertices[0]
    unvisited_vertices = vertices[1:]
    tour = [start_vertex]
    while len(unvisited_vertices) > 0:
        nearest_neighbor = find_nearest_neighbor(tour[-1], unvisited_vertices)
        unvisited_vertices.remove(nearest_neighbor)
        tour.append(nearest_neighbor)
     return tour
```

Simulated annealing:

1) Using the calculated distance matrix of all cities to calculate every solution path.

```
Length_calculate(dist, path)
    Len = 0
    Path is the sequence of the current solution of path
    Len += all length of edges in path
    Len += the length of last vertex to the first vertex in path
```

2) Using two kinds of idea to get the new path. One way is to change the order between two random node in old path, the other is randomly find three node x,y,z and change edge x-z to x-y and y-z. It can also add more method of changing the path such as reverse the old path or move all node to the left by one. More methods can make the algorithm faster and it is better to use ratio to control the possibility of using these methods.

```
Find_new_path_1(old_path)
    Randomly get two nodes x1 and x2.
    If x1 != x2
        Then change the order of x1 and x2, new_path[x1] = old_path[x2]
Find_new_path_2(old_path)
    Randomly get three nodes x1, x2 and x3
    If x1 != x2 != x3
    Then let E(a-c) = E(a - b) + E(b - c)
```

4. Best tours for the three example instances and the time it took to obtain these tours.

	Tour	Time(sec)
Example 1	130921	0.198181
Example 2	2975	8.574694
Example 3	1964948	84.758068

5. Best solutions for the competition test instances. Time limit 3 minutes and unlimited time.

	Tour	Time(sec)
Test-input-1	5911	0.125849
Test-input-2	8011	0.754211

Test-input-3	14826	11.497598
Test-input-4	19711	58.614923
Test-input-5	28685	0.792768
Test-input-6	40933	3.113752
Test-input-7	63780	19.571800