CS577 Homework 6

William Shu, Haoyi Lu

1. We are designing an algorithm to find to the shortest path from weighted graph. We want to find the minimum length that reaches place i with j stops by adding the minimum length of the previous places with j – 1 stops with the edge between the previous place and this place: we have this math function: f(I, j) =min{f(k, j – 1) + e(k, j)}. And we use an array to record them. Then we just need to find the max j such that f(n, j) <= x

Our Algorithm:

maxCityNum(e, V, x){

Initialize a 2d array f to store the min length (size of n \* n)

Initialize a list p to store the corresponding path

f[1][1] = 0;

path.add node Madison

for i = 1 to n: -----------n

for j = 1 to i: -----------1, 2, 3 … n

int min = infinite;

int recordPlaceNum;

for k = 1 to i – 1: -----------1, 2, 3 … n - 1

if f[k][j – 1] + e[k][i] < min && e[k][i] exists:

min = f[k][j – 1] + e[k][i];

recordPlaceNum = I;

f[i][j] = min;

path.add(p(recordPlaceNum))

in answer;

for j = 1 to n: -----------n

if f[n][j] <= x:

answer = j;

for k = j to 1: -----------j

return path.get(k);

return NoPath

Running Time:

for the three loops, 1\*1 + 2\*2 + 3\*3 + … + n\*n , we got O(n3).

Since finding the answer part, it is just O(n2) so it will not affect the overall running time.

Then the overall running time is O(n3).

Correctness:

Base case: We just return the Madison, our algorithm correct

General case: If the algorithm holds for n, then we have minimum length and path for visiting city n. Then if we are going to visit city n + 1, we need to traverse all the edges to n + 1. If the f(n) + e <= x, then we can add this edge to current minimum. If we find the minimum path, then we record it and add the corresponding path to the list. Then our algorithm holds.

CS577 Homework 6

William Shu, Haoyi Lu

1. We find that the carrying capacity is the dominating element in this problem, which means the one with higher carrying capacity should be under the ones with lower carrying capacity. We can prove this by exchange argument:

For any neighbor turtles, if the upper one (A) ‘s carrying capacity is larger than lower one (B). B has to be able to carry all the weight includes A and all the turtles above A, so if we change B to the top of A, it still works. Similarly, A has larger carrying capacity than B, B can carry all the weight includes A and all the turtles above A, then A can definitely be bottom of B. Then we still got A’s carrying capacity is larger than B and it is at bottom of B.

Then if there are n turtles, we just keep doing this process until all the turtle’s carrying capacity is decreasing from top to bottom.

Then we can know that there is a possible combination that holds in this situation.

Then we sort the turtle by their carrying capacity.

We want to record the weight of total of j turtles traversed and i of them are in the stack, fi, j. fi, j = min (fi-1, j, fi-1, j-1 + Wi) if (fi-1, j-1 + Wi) <= Si (the carrying capacity of ith turtle). Then we just need to find max number of j.

Our Algorithm: S for strength of turtle (carrying capacity), W for weight of the turtle

maxNum(Turtles: S, W):  
 initialize an 2-d n\*n array stands for the total weight of stacked turtles

f[1][0] = 0

for I = 1 to n:

for j = 1 to i:

f[i][j] = f[i – 1][j];

if f[i – 1][j - 1] + Wi <= Si && f[i – 1][j - 1] + Wi < f[i][j]:

f[i][j] = f[i – 1][j - 1] + Wi

in answer = 0;

for x = 1 to n:

for y = 1 to n:

if f[x][y] <= Sx && y > answer

answer = y;

return answer;

Running Time:

For the outer loop is 1 – n, the inner loop is 1 to i, which is 1 – 1 to 1 – n, then total is 1 + 2 + 3 + … + n = O(n2)

Correctness:

Base case:

There is only one turtle, then if the turtle’s weight is smaller than its strength then returns 1. This algorithm holds.

General Case:

First, we need to check if the array returns the minimum weight of the total turtles.

We assume the array holds for n – 1 turtle stacks.

For weight of i stacked turtles chosen from total of j turtles, we need to consider two situations, 1. The turtle below it counts 2. The Turtle below it not counts. If the turtle below it not counts, then the bottom of the stack is still the last one, which holds based on our assumption. If the turtle n counts in, we have to check if the turtle n has larger strength in order to carry its own weight and the weight above is, then if it can do that, then we need to take the minimum value of these two situations. Then our algorithm holds.

Then in general, our algorithm holds.