Algorithms

Programming Assignment #2

Worker Ant Path Optimization

Introduction:

Worker ants are usually assigned to find food and carry it back to the formicary. In this PA, you are a smart worker ant that has learnt optimization in the NTUEE algorithm class. Your companions have already found a lot of food outside and tell you their location and weight. Your job is to carry all of them back to the formicary in order, and you can only carry at most a certain weight (*i.e.* capacity) every time. Please notice that each food is too hard to separate so you can only choose to pick it all up or not (*i.e.* you cannot pick a portion of it). You can go back to the formicary and put down the food you picked whenever you want. Please implement an optimization algorithm to minimize the total distance you travel to pick all of food back to the formicary. We use the *Manhattan distance* to measure the distance between two points. For example, the distance between (3, 1) and (1, 2) is $|3 - 1| + |1 - 2| = 3$.

Input/output Files:

In the input file (`*.in`), the first line shows the capacity you can take. The second line means the number of food outside. Starting from the third line, every line shows the location and weight of every food, starting from the 1st food. The first two column means x-coordinate and y-coordinate (assume $0 \leq x < 1000$, $0 \leq y < 1000$). The third column means the weight of food ($0 < weight \leq capacity$). Please notice that the location of the formicary is (0, 0).

```
10
4
1 2 3
1 0 3
3 1 4
3 1 4
```

In the output file (`*.out`), the last lines shows the total shortest distance you take after picking all of food. Starting from first line, every line shows the food ID after which you should go back to the formicary after picking this food. The following example shows a result using the greedy algorithm. The worker ant tries to pick up as much food as possible before going back to the formicary. The result shows that it will go back to the formicary after picking up the 3rd and the 4th food. The total distance is *(Origin to 1st food) + (1st food to 2nd food) + (2nd food to 3rd food) + (3rd food to origin) + (origin to 4th food) + (4th food to origin) = (1 + 2) + (0 + 2) + (2 + 1) + (3 + 1) + (3 + 1) + (3 + 1) = 20.*

```
3
4
20
```

The following example shows the optimized results after using the dynamic

programming. You should go back after picking up the 2nd food and 4th food. The optimized total distance is shorter than the result obtained by the greedy algorithm. The total distance is

(Origin to 1st food) + (1st food to 2nd food) + (2nd food to origin) + (origin to 3rd and 4th food) + (3rd and 4th food to origin) = (1 + 2) + (0 + 2) + (1 + 0) + (3 + 1) + (3 + 1) = 14.

```
2
4
14
```

Jobs:

1. **Prove the Optimal Substructure:** Write a recursive function for this problem. Prove that this problem has *optimal substructure* so that it can be solved by dynamic programming.

Hints: (it is NOT required to use this model)
Suppose *d[i]* means the shortest distance after picking up the i^{th} food, and the maximum capacity is *C*. *dist2origin(i)* means the distance between the i^{th} food and the origin. *dist(a, b)* means the distance you need to travel between the a^{th} food and the b^{th} food. We can find the formula (please complete it):

$$d[i] = \min_{0 \le j < i} \{ d[j] + dist2origin(\_\_\_\_) + dist(\_\_\_\_, i) + dist2origin(i) \}$$

Assume that *j* was the **second-last** food ID (the last food ID is *i*) after which you return to the formicary so *j* is smaller than *i*. We also want to make sure that the sum of weight from *j+1* to *i* no larger than *C*. Please notice that if the minimum value of *d[i]* is found when *j = 0*, it means the ant should pick up the food from the 1^{st} to the i^{th} and then go back to the formicary only one time. *d[0]* should be zero to make the formula reasonable under this condition.
The following are the constraints corresponding to above formula:

$$\text{subject to constriants:}$$

$$j < i,$$

$$\sum_{k=j+1}^{i} weight(k) \le C$$

2. **Dynamic Programming:** Use dynamic programming to solve this problem. Please analyze the time complexity in your report.
3. **Greedy Choice Property:** Prove that his problem does NOT have greedy choice property so the greedy algorithm does NOT always produce the optimal solution.
4. **Greedy Implementation**: Write a greedy algorithm to implement this worker ant optimization tool. Please analyze the complexity in your report.

Command line parameters:

In the command line, you are required to follow this format

```
./WorkerAnt -[GD|DP] <input_file_name> <output_file_name>
```

where GD and DP represent the greedy algorithm, dynamic programming, respectively.

For example, the following command invokes the tool and run the greedy algorithm.

```
./bin/WorkerAnt -GD ./inputs/case1.in ./outputs/case1.out
```

Requirements:

1. Your source code must be written in C or C++. The code must be executable on EDA union lab machines.
2. In your report, please fill in the following table and also plot a figure showing the memory and running time. Please use –O2 optimization and turn off all debugging message.

case	# of food	GD			DP		
		Total distance	CPU time (s)	Memory (KB)	*Total distance*	CPU time (s)	Memory (KB)
case1	4						
case2							
case3							
case4							

3. Your binary code should be compiled by the following commands under <student_id>_pa2 directory. The binary code is located in the bin directory.

```
make
cd bin
./WorkerAnt
```

Submission:

Please submit a hardcopy of your report in class. Also, please submit a single *.*tgz* file to CEIBA system. Your submission must contain:

1. <student_id>_pa2 directory contains your source code in *src* directory. By simply typing "make" can compile.
2. A report in the *doc* directory. <student_id>_pa2_report.doc. Please also submit a printed report in class.
3. A README file that explains your files.
4. We will use our own test cases so do NOT include the input files.

The submission filename should be compressed in a single file <student_id>_pa2.tgz. (e.g. b90901000_pa2.tgz). If you have a modified version, please add _v2 as a postfix to the filename and resubmit it (e.g. b90901000_pa2_v2.tgz). You can use the following command to compress a whole directory:

```
tar -zcvf <filename>.tgz <dir>
```

You are required to run the checksubmitPA2 script to check if your .tgz submission file is correct. To use this script, simply type

```
./checkSubmitPA2.sh  b99901000_pa2.tgz
```

We have so many students in the class so we need automatic grading. Any mistake in the submission will result in cost 20% off your score. Please be very careful in your

submission.

Grading:

60% correctness (including submission correctness)
20% file format and location
20% report

Bonus:

10% bonus will be given to any extra features that you add to make your program faster or require less memory. Please write clearly in your report.

NOTE:

Copying other source code can result in zero grade for all students involved.