Yuan Qin
yqin7@uci.edu

Part 3

1. Describe the machine you are running your tests on (CPU, OS, Memory, other relevant info)

—OS is MacOs, and the application is implemented under python3.
—Time profiling used cProfile, and memory_profiler packages is imported and used for memory monitoring.
—Starting point (0,0), end point(0,20)

2. Briefly describe the algorithm you are using again in this document

—The profiling in this document is mainly based on nearest neighbor since brute force in my implementation takes more than 20 sec for item number beyond 11.

3. Time spent (in ms, seconds, etc.) processing "warehouse-grid" file to put into memory or other storage (in KB, MB, etc.)

—We can see from the third line of the profile that my function readin() is called in warehouseapp.py module once for 0.174 seconds

```
Total goods num: 25525
Max rack number in row, col 18.0 10.0
         115 function calls in 0.175 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.175    0.175 {built-in method builtins.exec}
        1    0.000    0.000    0.175    0.175 <string>:1(<module>)
        1    0.174    0.174    0.175    0.175 warehouseapp.py:32(readin)
       51    0.000    0.000    0.001    0.000 codecs.py:318(decode)
       51    0.000    0.000    0.000    0.000 {built-in method _codecs.utf_8_decode}
        1    0.000    0.000    0.000    0.000 {built-in method io.open}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 _bootlocale.py:23(getpreferredencoding)
        1    0.000    0.000    0.000    0.000 {built-in method _locale.nl_langinfo}
        1    0.000    0.000    0.000    0.000 codecs.py:308(__init__)
        1    0.000    0.000    0.000    0.000 {built-in method _csv.reader}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 codecs.py:259(__init__)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

4.

Any time spent pre-processing data for order gathering (converting vertices/shelves into points w/ distances to other points, etc).

—From the third line, we can see that 0.003s is spent on creating the path graph including creating nodes (location point) and edges between them.

```
Total goods num: 25525
Max rack number in row, col 18.0 10.0
         2307 function calls in 0.006 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.006    0.006 {built-in method builtins.exec}
        1    0.000    0.000    0.006    0.006 <string>:1(<module>)
        1    0.003    0.003    0.006    0.006 warehouseapp.py:343(createpathg)
      620    0.002    0.000    0.002    0.000 graph.py:823(add_edge)
      441    0.001    0.000    0.001    0.000 graph.py:442(add_node)
      621    0.000    0.000    0.000    0.000 {method 'update' of 'dict' objects}
      620    0.000    0.000    0.000    0.000 {method 'get' of 'dict' objects}
        1    0.000    0.000    0.000    0.000 graph.py:270(__init__)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

5. Average time spent & amount of memory used for calculating the path for a single item (assume start & end point are (0,0) or whatever equivalent you have in your program)

Memory profiling compile use:

python -m memory_profiler warehouseapp.py

Start calculating from pragma *@profile* end at *return,* which is a line by line memory profiling.

- Item #46071

—Original() function cost 0.032 seconds in this scenario while only the original path computing function is profiled.

Time spent:

```
Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.067    0.067 {built-in method builtins.exec}
     1    0.000    0.000    0.067    0.067 <string>:1(<module>)
     1    0.000    0.000    0.067    0.067 warehouseapp.py:324(singleOrder)
     4    0.001    0.000    0.067    0.017 graphtest.py:14(locdistance)
    76    0.000    0.000    0.049    0.001 function.py:663(get_node_attributes)
    76    0.016    0.000    0.048    0.001 function.py:685(<dictcomp>)
     1    0.000    0.000    0.035    0.035 warehouseapp.py:169(optimizeorder)
     2    0.000    0.000    0.034    0.017 warehouseapp.py:67(findpath)
 33592    0.024    0.000    0.032    0.000 _collections_abc.py:742(__iter__)
     1    0.000    0.000    0.032    0.032 warehouseapp.py:305(originalorder)
     8    0.010    0.001    0.017    0.002 weighted.py:729(_dijkstra_multisource)
     4    0.000    0.000    0.010    0.002 weighted.py:90(dijkstra_path)
     4    0.000    0.000    0.010    0.002 weighted.py:360(single_source_dijkstra)
     4    0.000    0.000    0.009    0.002 weighted.py:602(multi_source_dijkstra)
     4    0.000    0.000    0.008    0.002 weighted.py:164(dijkstra_path_length)
     4    0.000    0.000    0.008    0.002 weighted.py:715(_dijkstra)
 33516    0.008    0.000    0.008    0.000 reportviews.py:177(__getitem__)
  5384    0.002    0.000    0.004    0.000 weighted.py:87(<lambda>)
  5386    0.002    0.000    0.002    0.000 {method 'get' of 'dict' objects}
  1862    0.001    0.000    0.001    0.000 {built-in method _heapq.heappop}
  1980    0.001    0.000    0.001    0.000 {built-in method _heapq.heappush}
  1980    0.000    0.000    0.000    0.000 {built-in method builtins.next}
  1854    0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
    76    0.000    0.000    0.000    0.000 graph.py:628(nodes)
     6    0.000    0.000    0.000    0.000 {built-in method builtins.print}
    76    0.000    0.000    0.000    0.000 _collections_abc.py:676(items)
    76    0.000    0.000    0.000    0.000 reportviews.py:174(__iter__)
    76    0.000    0.000    0.000    0.000 reportviews.py:167(__init__)
```

Memory spent:

```
Line #    Mem usage    Increment   Line Contents
================================================
   306   45.402 MiB   45.402 MiB   @profile
   307                             def originalorder(oneorder,x_init,y_init,x_end,y_end):
   308                                 # original order
   309   45.402 MiB    0.000 MiB       dist_oneorder = 0
   310   45.438 MiB    0.000 MiB       for item in oneorder:
   311   45.438 MiB    0.035 MiB           dist, x_des, y_des = findpath(item,x_init,y_init)
   312   45.438 MiB    0.000 MiB           x_init = x_des
   313   45.438 MiB    0.000 MiB           y_init = y_des
   314   45.438 MiB    0.000 MiB           dist_oneorder = dist_oneorder + dist
   315                                     # print
   316                                 # back to end point
   317
   318                                 # print"returning to end point......"
   319   45.445 MiB    0.008 MiB       backtrip = graphtest.locdistance(pathgraph,x_init,y_init,x_end,y_end)
   320   45.445 MiB    0.000 MiB       dist_oneorder = dist_oneorder + backtrip
   321
   322   45.445 MiB    0.000 MiB       print ('Distance for one order without optimization', dist_oneorder)
   323   45.445 MiB    0.000 MiB       return dist_oneorder
```

- Item # 379019:

ncalls tottime percall cumtime percall filename:lineno(function)
1 0.000 0.000 0.032 0.032 warehouseapp.py:305(originalorder)

```
Line #    Mem usage    Increment   Line Contents
================================================
   306   43.402 MiB   43.402 MiB   @profile
   307                             def originalorder(oneorder,x_init,y_init,x_end,y_end):
   308                                 # original order
   309   43.402 MiB    0.000 MiB       dist_oneorder = 0
   310   43.449 MiB    0.000 MiB       for item in oneorder:
   311   43.449 MiB    0.047 MiB           dist, x_des, y_des = findpath(item,x_init,y_init)
   312   43.449 MiB    0.000 MiB           x_init = x_des
   313   43.449 MiB    0.000 MiB           y_init = y_des
   314   43.449 MiB    0.000 MiB           dist_oneorder = dist_oneorder + dist
   315                                     # print
   316                                 # back to end point
   317
   318                                 # print"returning to end point......"
   319   43.449 MiB    0.000 MiB       backtrip = graphtest.locdistance(pathgraph,x_init,y_init,x_end,y_end)
   320   43.449 MiB    0.000 MiB       dist_oneorder = dist_oneorder + backtrip
   321
   322   43.449 MiB    0.000 MiB       print ('Distance for one order without optimization', dist_oneorder)
   323   43.449 MiB    0.000 MiB       return dist_oneorder
```

- Item #70172:

ncalls tottime percall cumtime percall filename:lineno(function)
1 0.000 0.000 0.029 0.029 warehouseapp.py:305(originalorder)
Memory profile:
171 45.457 MiB 45.457 MiB @profile
306 45.488 MiB 0.000 MiB return optoneorder,min

- Item # 1321:

ncalls tottime percall cumtime percall filename:lineno(function)
1 0.000 0.000 0.027 0.027 warehouseapp.py:305(originalorder)
Memory profile:
171 43.895 MiB 43.895 MiB @profile
306 43.945 MiB 0.000 MiB return optoneorder,min

- Item #2620261:

ncalls tottime percall cumtime percall filename:lineno(function)
1 0.000 0.000 0.026 0.026 warehouseapp.py:305(originalorder)
Memory profile:
171 45.434 MiB 45.434 MiB @profile
306 45.465 MiB 0.000 MiB return optoneorder,min

Average time cost=(0.032+0.032+0.029+0.027+0.026)/5=0.0292s
Average memory cost =
(45.445-45.402+43.449-43.402+45.488-45.437+43.945-43.895+45.465-45.434)/5=0.0444MiB

6. Average time spent & amount of memory used for calculating orders of various sizes from our "warehouse-orders" file (assume start & end point are (0,0) or whatever equivalent you have in your program)

—Memory profiled for the optimizedorder function, which used nearest-neighbor algorithm.
—Time profiled within the call of optimizedorder function.

- 3 items (orders #2, 3, 15, 25)

#2

1 0.000 0.000 0.105 0.105 warehouseapp.py:169(optimizeorder)

171 45.469 MiB 45.469 MiB @profile
306 45.527 MiB 0.000 MiB return optoneorder,min
#3
1 0.000 0.000 0.111 0.111 warehouseapp.py:169(optimizeorder)
171 45.516 MiB 45.516 MiB @profile
306 45.566 MiB 0.000 MiB return optoneorder,min

#15
1 0.000 0.000 0.074 0.074 warehouseapp.py:169(optimizeorder)
171 45.516 MiB 45.516 MiB @profile
306 45.551 MiB 0.000 MiB return optoneorder,min
#25
1 0.000 0.000 0.186 0.186 warehouseapp.py:169(optimizeorder)
171 45.426 MiB 45.426 MiB @profile
306 45.461 MiB 0.000 MiB return optoneorder,min
Average time cost=(0.105+0.111+0.074+0.186)/4=0.119s
Average memory cost = (45.527-45.469+45.566-45.516+45.551-45.516+45.461-45.426)/4=0.0445MiB

- 5 items (orders #1, 4, 17, 26)

#1
1 0.000 0.000 0.252 0.252 warehouseapp.py:172(optimizeorder)
171 45.504 MiB 45.504 MiB @profile
306 45.551 MiB 0.000 MiB return optoneorder,min
#4
1 0.000 0.000 0.236 0.236 warehouseapp.py:172(optimizeorder)
171 45.578 MiB 45.578 MiB @profile
306 45.637 MiB 0.000 MiB return optoneorder,min
#17
1 0.000 0.000 0.219 0.219 warehouseapp.py:172(optimizeorder)
171 43.680 MiB 43.680 MiB @profile
306 43.750 MiB 0.000 MiB return optoneorder,min
#26
1 0.000 0.000 0.318 0.318 warehouseapp.py:172(optimizeorder)
171 45.543 MiB 45.543 MiB @profile
306 45.590 MiB 0.000 MiB return optoneorder,min

Average time cost=(0.252+0.236+0.219+0.318)/4=0.256s
Average memory cost = (45.541-45.504+45.637-45.578+43.750-43.680+45.590-45.543)/4=0.0533MiB

- 10 items (orders #5, 43)

#5
1 0.001 0.001 0.795 0.795 warehouseapp.py:172(optimizeorder)
171 45.434 MiB 45.434 MiB @profile
306 45.508 MiB 0.000 MiB return optoneorder,min
#43

1 0.000 0.000 0.606 0.606 warehouseapp.py:172(optimizeorder)
171 45.496 MiB 45.496 MiB @profile
306 45.578 MiB 0.000 MiB return optoneorder,min

Average time cost=(0.795+0.606)/2=0.7s
Average memory cost = (45.508-45.434+45.578-45.496)/2=0.078MiB

- 21 items (orders #10, 12, 69)

#10
1 0.002 0.002 3.197 3.197 warehouseapp.py:172(optimizeorder)
171 45.621 MiB 45.621 MiB @profile
306 45.785 MiB 0.000 MiB return optoneorder,min
#12
1 0.002 0.002 3.811 3.811 warehouseapp.py:172(optimizeorder)
171 45.785 MiB 45.785 MiB @profile
306 45.910 MiB 0.000 MiB return optoneorder,min
#69
1 0.002 0.002 3.345 3.345 warehouseapp.py:172(optimizeorder)
171 45.570 MiB 45.570 MiB @profile
306 45.699 MiB 0.000 MiB return optoneorder,min

Average time cost=(3.197+3.811+3.345)/3=3.451sec
Average memory cost = (45.785-45.621+45.910-45.785+45.699-45.621)/3=0.123MiB