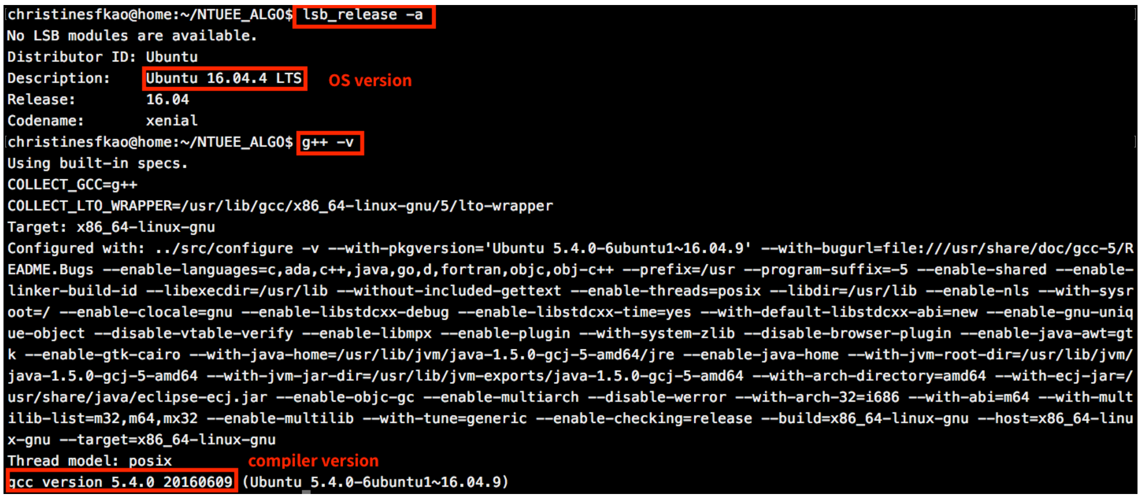
第 12 組 B02901035 電機五 高紹芳

Algorithms
Programming Assignment #2
Worker Ant Path Optimization

1. Fill in the following table.

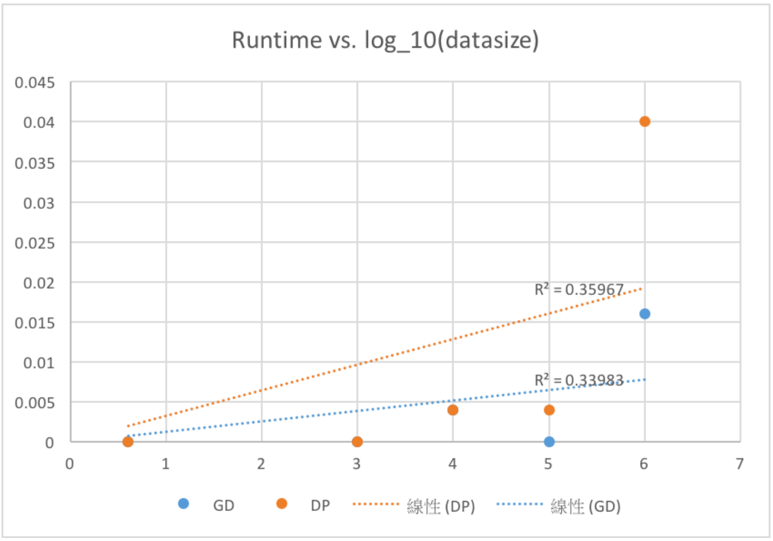
Note: Since all the servers on EDA Union is under maintenance, I used the server home.ntuosc.org under 140.112.202.149 for the following work.

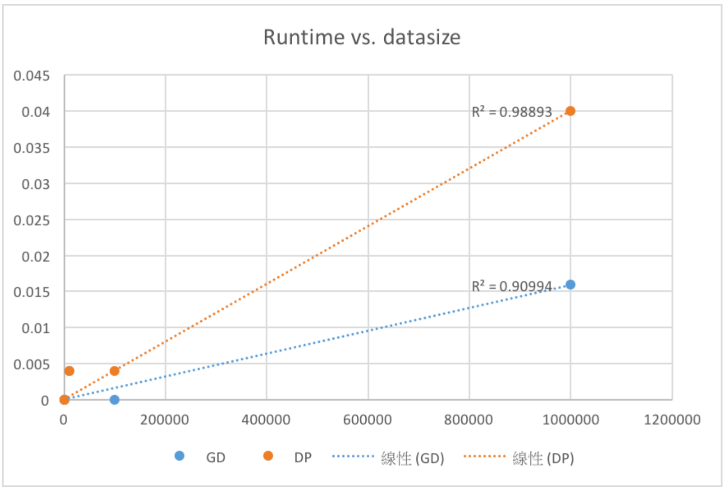


```
christinesfkao@home:~/NTUEE_ALGO$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.4 LTS  OS version
Release:        16.04
Codename:       xenial
christinesfkao@home:~/NTUEE_ALGO$ g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 5.4.0-6ubuntu1~16.04.9' --with-bugurl=file:///usr/share/doc/gcc-5/R
EADME.Bugs --enable-languages=c,ada,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-5 --enable-shared --enable-
linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --with-sysr
oot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-uniq
ue-object --disable-vtable-verify --enable-libmpx --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gt
k --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-5-amd64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/
java-1.5.0-gcj-5-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-5-amd64 --with-arch-directory=amd64 --with-ecj-jar=/
usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-mult
ilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linu
x-gnu --target=x86_64-linux-gnu
Thread model: posix          compiler version
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.9)
```

Case	# of food	GD			DP		
		Total distance	CPU time (s)	Memory (MB)	*Total distance*	CPU time (s)	Memory (MB)
case1	4	20	0	13.292	14	0	13.292
case2	1000	154742	0	13.292	150426	0	13.292
case3	10000	1550932	0.004	13.688	1512940	0.004	13.7
case4	100000	30931698	0	16.376	30092520	0.004	17.156
case5	1000000	1551949160	0.016	37.88	1510902772	0.04	44.036

2. Draw a figure to show the runtime of both GD and DP. Please draw all runtime curves on the same figure so that we can see difference.





3. Prove this worker ants problem has optimal substructure.

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:

$$d[i] = \min_{0 \le j < i} \{d[j] + dist2origin(j+1) + dist(j+1, i) + dist2origin(i)\}$$

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 1st 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:

$$\text{subject to constriants:}$$
$$j < i,$$
$$\sum_{k=j+1}^{i} weight(k) \le C$$

4. Analyze the time complexity of the dynamic programming.

init() => O(n) (two for loops inside)

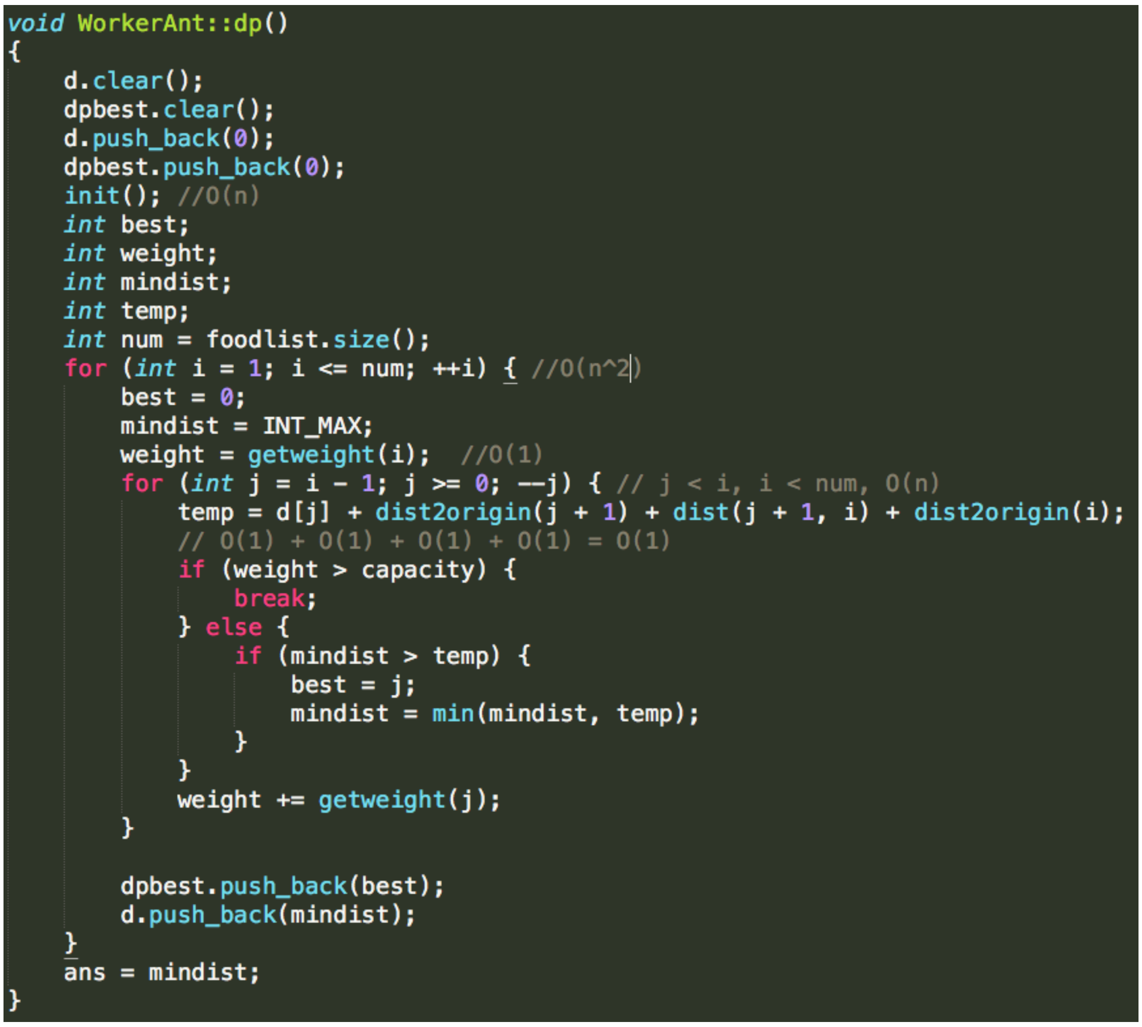
getweight() => O(1)

dist2origin() => O(1)

dist() => O(1)

two nested for loops => O(n^2)

⇨ O(n^2)



```
void WorkerAnt::dp()
{
    d.clear();
    dpbest.clear();
    d.push_back(0);
    dpbest.push_back(0);
    init(); //O(n)
    int best;
    int weight;
    int mindist;
    int temp;
    int num = foodlist.size();
    for (int i = 1; i <= num; ++i) { //O(n^2)
        best = 0;
        mindist = INT_MAX;
        weight = getweight(i);  //O(1)
        for (int j = i - 1; j >= 0; --j) { // j < i, i < num, O(n)
            temp = d[j] + dist2origin(j + 1) + dist(j + 1, i) + dist2origin(i);
            // O(1) + O(1) + O(1) + O(1) = O(1)
            if (weight > capacity) {
                break;
            } else {
                if (mindist > temp) {
                    best = j;
                    mindist = min(mindist, temp);
                }
            }
            weight += getweight(j);
        }

        dpbest.push_back(best);
        d.push_back(mindist);
    }
    ans = mindist;
}
```

5. **Prove that this problem does NOT have greedy choice property.**

Greedy choice property assures that making locally optimal choices will eventually lead to a globally optimal solution. However, in this problem, since the order of the food ID is fixed beforehand, we cannot guarantee that the greedy choice brings the optimal solution in the end. Moreover, from the form we filled in Q1, we see that some total distances are larger while others are smaller than the DP version, which provides contradiction for greedy choice property to hold.

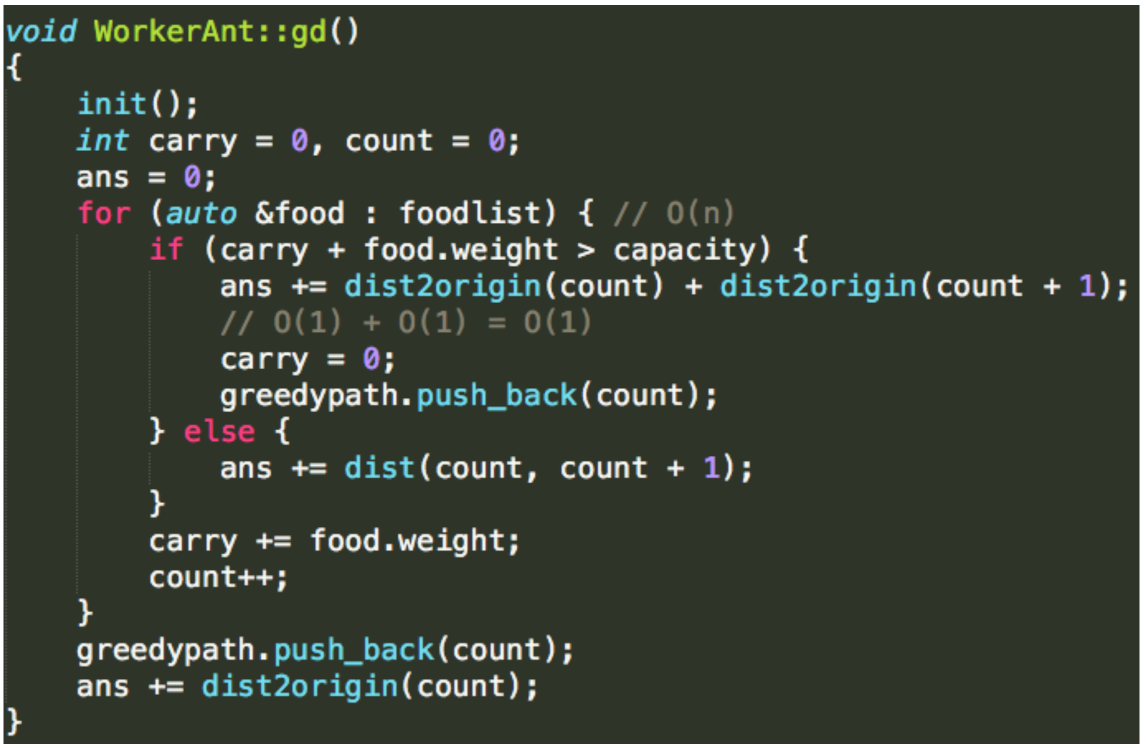
6. **Please analyze the complexity of your greedy algorithms.**

init() => O(n) (two for loops inside)

dist2origin() => O(1)

dist() => O(1)

⇨ O(n)



```cpp
void WorkerAnt::gd()
{
    init();
    int carry = 0, count = 0;
    ans = 0;
    for (auto &food : foodlist) { // O(n)
        if (carry + food.weight > capacity) {
            ans += dist2origin(count) + dist2origin(count + 1);
            // O(1) + O(1) = O(1)
            carry = 0;
            greedypath.push_back(count);
        } else {
            ans += dist(count, count + 1);
        }
        carry += food.weight;
        count++;
    }
    greedypath.push_back(count);
    ans += dist2origin(count);
}
```