資料結構 (Data Structure) 作業四

姓名：謝政彥
學號：109753207

一、 作業目標

Study the Properties of "Small World" and Compare Different Data Structures as follow:

1. Generate a cycle of 1000 nodes. Each edge has length 1.
2. Add x random edges. Each random edge has the same length y.
3. Sample z pairs of source and destination and compute the average shortest distance (d) of these z source-destination pairs.

You need to use 2 different structures of heaps.

二、 任務

1. A picture of the graph where x = 100.
2. Responses to the following questions:
   a. What is the relationship between x and d?
   b. What is the relationship between y and d?
   c. How to choose z properly to reflect the true average distance between all pairs of source and destination?
   d. Which implementation of Dijkstra's Algorithm is the fastest?

You need to support your answers with experimental results.

You also need to explain how you obtain the results.

三、 實驗環境：

(一)電腦：

1. 處理器： Intel(R) Core(TM)i7-4790 CPU @4.00GHz 4.00GHz
2. 記憶體(RAM)： 32.00GB
3. 系統類型： 64 位元作業系統，x64 型處理器
4. Windows 規格
   (1) 版本 Windows 10 家用版
   (2) 版本 20H2

(二)g++版本資訊：

```
Using built-in specs.
COLLECT_GCC=C:\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin\gcc.exe
COLLECT_LTO_WRAPPER=C:/mingw-w64/x86_64-8.1.0-posix-seh-rt_v6-
rev0/mingw64/bin/../libexec/gcc/x86_64-w64-mingw32/8.1.0/lto-wrapper.exe
```

Target: x86_64-w64-mingw32

Configured with: ../../../src/gcc-8.1.0/configure --host=x86_64-w64-mingw32 --build=x86_64-w64-mingw32 --target=x86_64-w64-mingw32 --prefix=/mingw64 --with-sysroot=/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64 --enable-shared --enable-static --disable-multilib --enable-languages=c,c++,fortran,lto --enable-libstdcxx-time=yes --enable-threads=posix --enable-libgomp --enable-libatomic --enable-lto --enable-graphite --enable-checking=release --enable-fully-dynamic-string --enable-version-specific-runtime-libs --disable-libstdcxx-pch --disable-libstdcxx-debug --enable-bootstrap --disable-rpath --disable-win32-registry --disable-nls --disable-werror --disable-symvers --with-gnu-as --with-gnu-ld --with-arch=nocona --with-tune=core2 --with-libiconv --with-system-zlib --with-gmp=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpfr=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpc=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-isl=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-pkgversion='x86_64-posix-seh-rev0, Built by MinGW-W64 project' --with-bugurl=https://sourceforge.net/projects/mingw-w64 CFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CXXFLAGS='-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CPPFLAGS=' -I/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' LDFLAGS='-pipe -fno-ident -L/c/mingw810/x86_64-810-posix-seh-rt_v6-rev0/mingw64/opt/lib -L/c/mingw810/prerequisites/x86_64-zlib-static/lib -L/c/mingw810/prerequisites/x86_64-w64-mingw32-static/lib '

Thread model: posix

gcc version 8.1.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)

(三)在 visual studio code 執行 complie。

四、 演算法程式來源：詳第 12 頁。

五、 實驗程式碼：詳第 13 頁。

六、 實驗過程及結果：

(一)在繪製圖形時，使用 opencv 套件，先運用圓形半徑 r 及 cos(θ)、sin(θ) 求得圓上 1000 個點座標，將點與點以線條連結，再從這 1000 個點隨機不重複產生 100 對點資料，以線條連結成 edges 繪製而成。

(二)有關 Small World 實驗，係使用 Binary Heap 及 Unordered_Map 為資料結構，以 Dijkstra 演算法求最小距離。

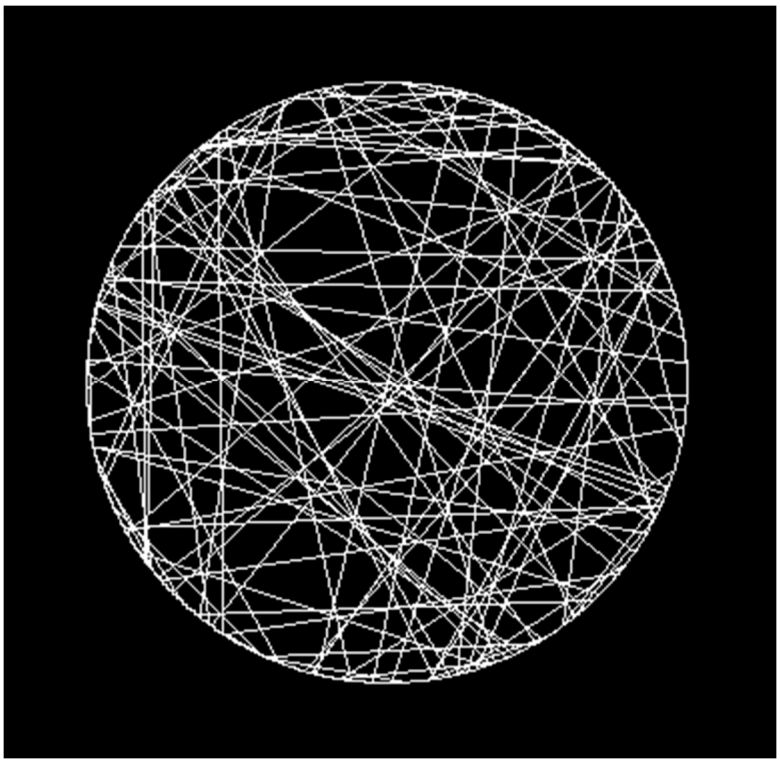
(三)實驗設計係以重複 5 次為最外圍迴圈，接下來隨機新增 edge 之長度(y) 為第 2 層迴圈 【for (int y=1; y<=256; y=2*y)】，再來是隨機新增 edge 數(x) 為第 3 層迴圈【for (int x=0; x<=300; x=x+10)】，最後是起訖點取樣對數(z) 為第 4 層迴圈【for (int z =50; z<=300 ; z=z+50)】，以

Unordered_Map 為資料結構，運用 Dijkstra 演算法求得平均最短距離 (d)，以 csv 格式匯出。

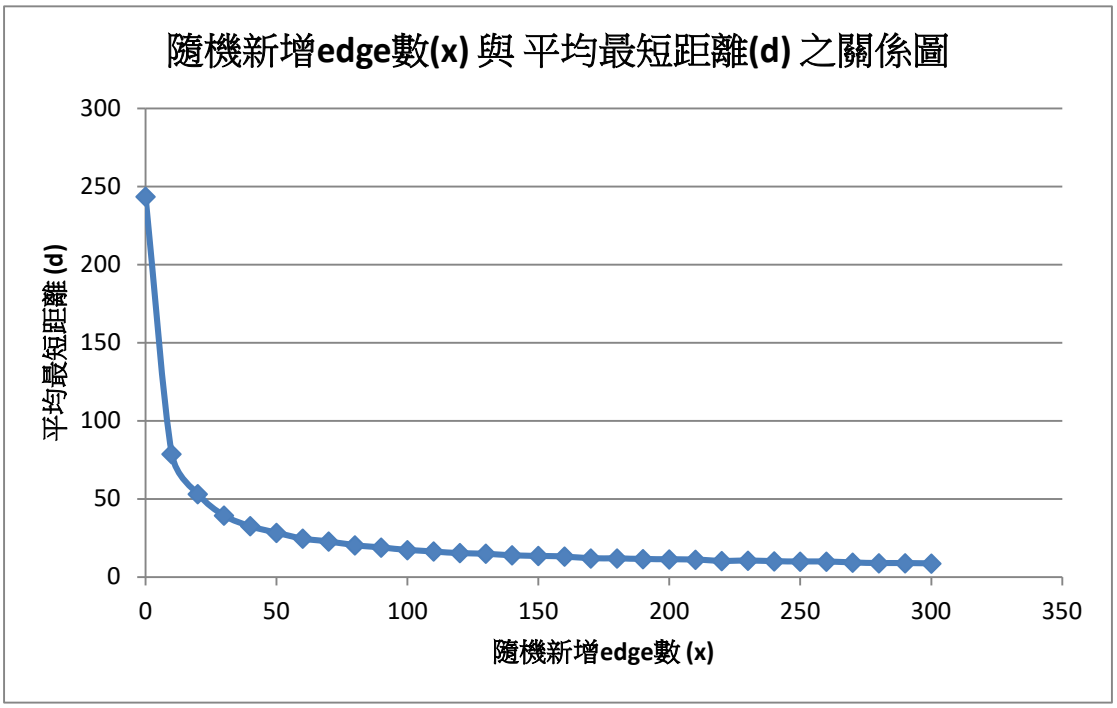
(四)比較 Binary Heap 及 Unordered_Map 為資料結構之演算時間，並參考前面實驗成果，設計如下：隨機新增 edge 數 x=100、隨機新增 edge 之長度 y=1、起訖點取樣對數 z=100 及重複 10 次，並計算所需時間。

(五)謹就所獲得的執行成果，圖 1 以 opencv 繪製，其餘以 Excel 繪製：

1. 圖 1：A picture of the graph where x = 100



2. 圖 2：The relationship between x and d

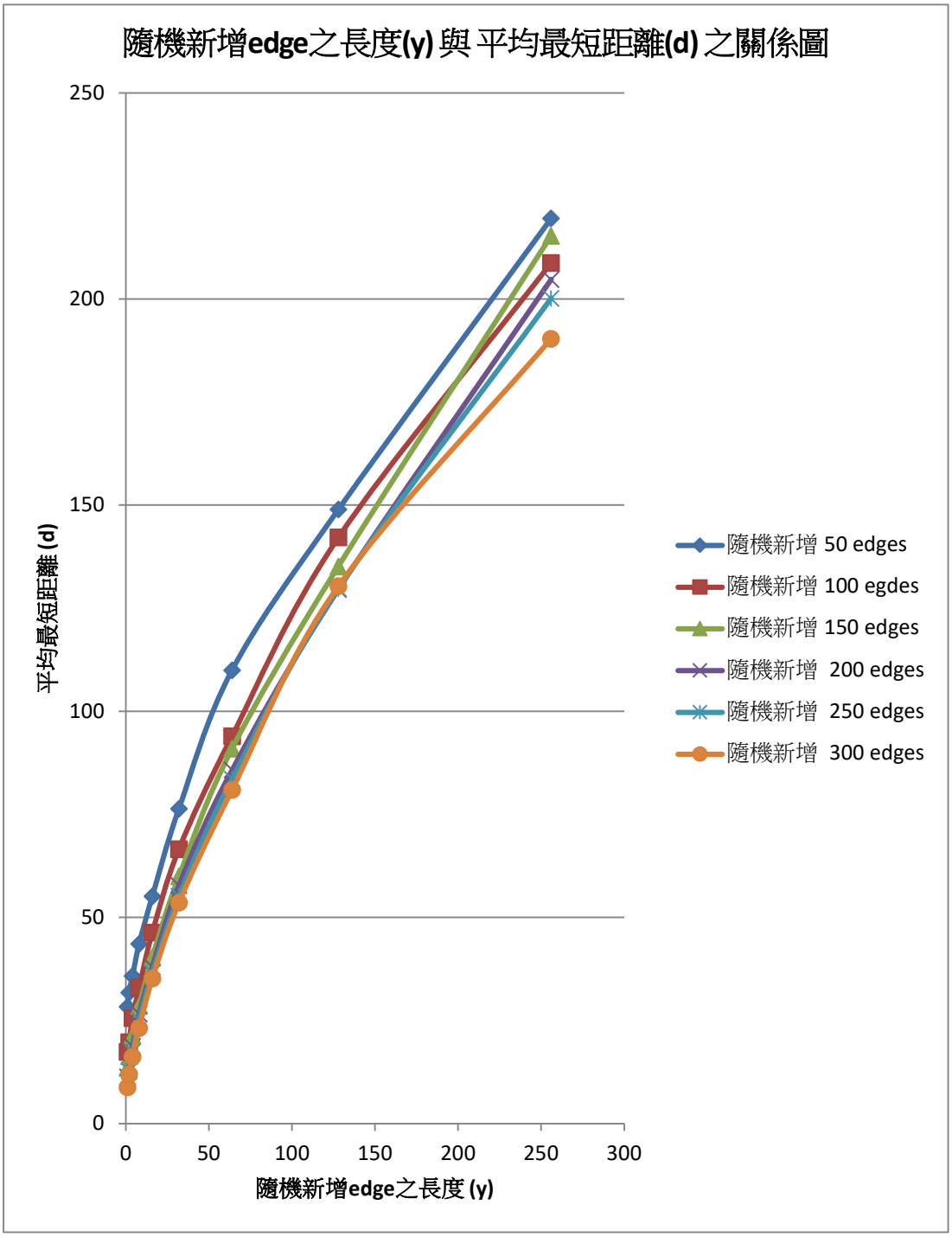


(1) 在固定 y=1、z=100 情況下，隨機新增 edge 數(x)自 0 開始，以 10 為單位，逐步增加到 300。

(2) 從圖 2 得知，平均最短距離(d)隨著新增 edge 數(x)的增加而減少，減少幅度最大在 x 位於 0 到 50 之間，50 到 100 已明顯趨緩，100 以後持續降低。

(3) 因 x=100 已有顯著降低平均最短距離(d)之效果，爰在資料結構演算時間測試時，即採用此數值。

3. 圖 3：The relationship between y and d

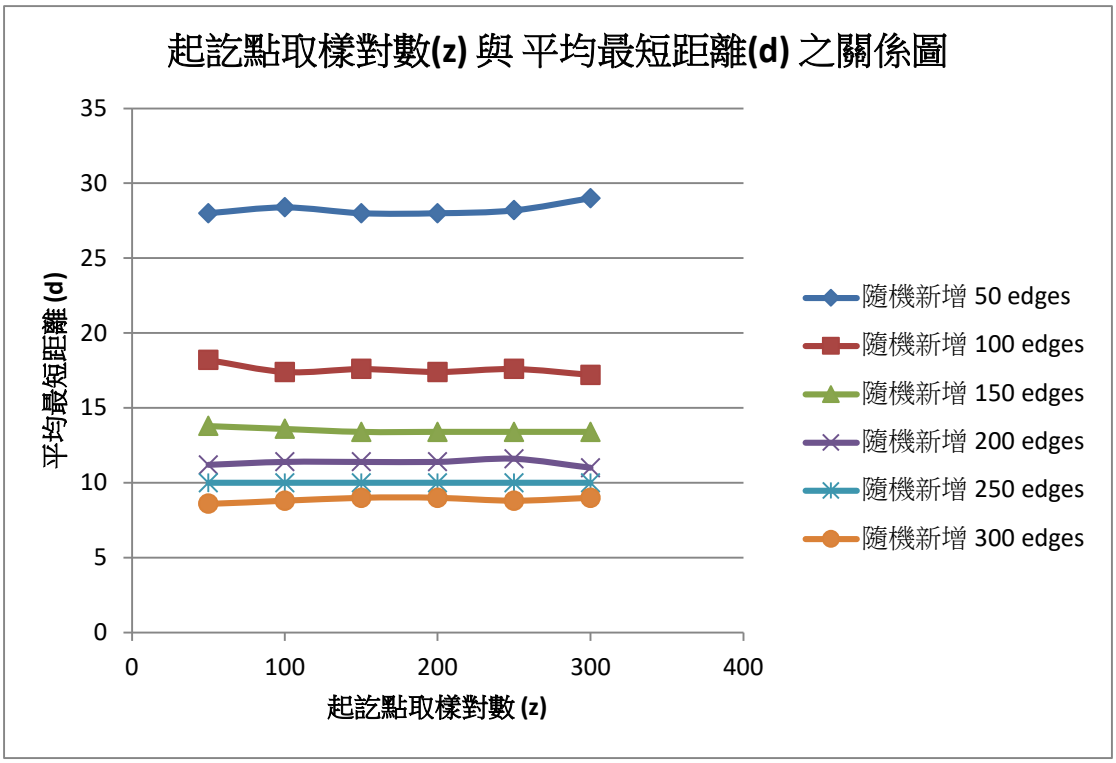


(1) 在固定 z=100，同時比較 edge 數 x= 50, 100, 150, 200, 250, 300 情況下，隨機新增 edge 之長度(y)自 1 開始，以 2 倍方式遞增，逐步增加到 256。

(2) 從圖 3 得知，平均最短距離(d)隨著 edge 之長度(y)的增加而增加，不管 x 為何，均呈現相同趨勢。

(3) 因 y=1 時距離(d)最短，爰在資料結構演算時間測試時，即採用此數值。

4. 圖 4：The relationship between z and d



(1) 在固定 y=1，同時比較 edge 數 x= 50, 100, 150, 200, 250, 300 情況下，起訖點取樣對數 (z)自 50 開始，以增加 50 方式遞增，逐步增加到 300。

(2) 從圖 4 得知，不管 x 為何，平均最短距離(d)的趨勢並無太大起伏，於是進一步計算其平均值及標準偏差如下表。

5. 表 1：起訖點取樣對數(z)與平均最短距離(d)在不同新增 edges 數(x)下之實驗數據及統計資料

隨機新增 edges 數 (x)	起訖點取樣對數 (z)						平均值 (Avg.)	標準偏差 (SD)
	50	100	150	200	250	300		
50	28	28.4	28	28	28.2	29	28.3	0.4
100	18.2	17.4	17.6	17.4	17.6	17.2	17.6	0.3
150	13.8	13.6	13.4	13.4	13.4	13.4	13.5	0.1
200	11.2	11.4	11.4	11.4	11.6	11	11.4	0.1
250	10	10	10	10	10	10	10.0	0.0
300	8.6	8.8	9	9	8.8	9	8.8	0.1

由表 1 資料顯示，不同取樣對數(z)取得之最短距離(d)，其標準偏差差異不大，其中 z=100 時似乎比較接近平均值，爰在資料結構演算時間測試時，即採用此數值。

6. 表 2：不同資料結構之 Dijkstra 演算時間比較表

時間單位：Tick (毫秒)

資料結構	重複實驗之時間平均	起訖取樣對數 (z)	每次演算平均時間
BinaryHeap	724.7	100	7.247
Unordered_Map	355.1	100	3.551

在固定 x=100、y=1、z=100 條件下，由表 2 得知，以 Unordered_Map 為資料結構的 Dijkstra 演算速度較 Binary Heap 快。

七、 心得、疑問及遇到困難

(一)心得

這份作業學到如何運用 opencv 套件來繪製圖形，特別是運用三角函數，及其與圓形的關係公式，算出 1000 個點的座標，再隨機取樣 100 對不重複之點資料，來繪製 edges。圖形繪製完成後，發現只出現 1 個象限的圖形，經網路查詢後才知道左上角的座標為(0,0)，且 x、y 均為正數，於是將前述 1000 個計算出來的座標進行平移，才順利完成。

此外，在進行實驗時，雖然是參採網路搜尋到的程式，但要實際運用在作業上時，還是需要透過上課所學，修改程式，包括計算出起訖點時即終止運算，每迴圈開始時清除資料，重新 setup 圖形基本資料等。當然，最重要的是，透過實驗，發覺 Small World 中 x、y、z 與 d 的關係，以及在不同需求情況下，應使用不同資料結構。

(二)疑問及遇到困難：原本設計 2 個資料結構一起執行，但在執行迴圈時，發現 Binary Heap 資料結構所使用的 Graph (儲存在 vector<list<pair<int,int>>>)資料在第 2 回運算時雖然有 clear 掉，但在重新加入資料時，在新增第 17 筆資料時，即中止執行，原因不明，只好先放棄，並以 Unordered_Map 來完成實驗。有關存在 vector 的資料 clear 掉後，無法順利新增，不知這要如何克服？

✧ 原始數據

表 3：The relationship between x and d

x	y	z	d
0	1	100	252
0	1	100	249
0	1	100	244
0	1	100	229
0	1	100	244
10	1	100	87
10	1	100	72
10	1	100	79
10	1	100	84
10	1	100	72
20	1	100	47
20	1	100	52
20	1	100	58
20	1	100	54
20	1	100	55
30	1	100	41
30	1	100	40
30	1	100	36
30	1	100	40
30	1	100	40
40	1	100	30
40	1	100	31
40	1	100	35
40	1	100	34
40	1	100	33
50	1	100	27
50	1	100	30
50	1	100	26
50	1	100	29
50	1	100	30
60	1	100	26
60	1	100	24
60	1	100	25
60	1	100	23
60	1	100	25
70	1	100	24
70	1	100	21
70	1	100	23
70	1	100	23
70	1	100	23

x	y	z	d
80	1	100	23
80	1	100	18
80	1	100	20
80	1	100	21
80	1	100	20
90	1	100	20
90	1	100	19
90	1	100	20
90	1	100	17
90	1	100	19
100	1	100	19
100	1	100	16
100	1	100	17
100	1	100	18
100	1	100	17
110	1	100	17
110	1	100	16
110	1	100	16
110	1	100	16
110	1	100	17
120	1	100	15
120	1	100	15
120	1	100	15
120	1	100	17
120	1	100	15
130	1	100	15
130	1	100	14
130	1	100	14
130	1	100	17
130	1	100	15
140	1	100	13
140	1	100	14
140	1	100	14
140	1	100	15
140	1	100	14
150	1	100	14
150	1	100	13
150	1	100	14
150	1	100	14
150	1	100	13

x	y	z	d
160	1	100	13
160	1	100	13
160	1	100	13
160	1	100	13
160	1	100	14
170	1	100	12
170	1	100	12
170	1	100	12
170	1	100	12
170	1	100	12
180	1	100	12
180	1	100	12
180	1	100	12
180	1	100	12
180	1	100	12
190	1	100	11
190	1	100	12
190	1	100	11
190	1	100	12
190	1	100	12
200	1	100	11
200	1	100	12
200	1	100	11
200	1	100	12
200	1	100	11
210	1	100	11
210	1	100	11
210	1	100	11
210	1	100	12
210	1	100	11
220	1	100	11
220	1	100	11
220	1	100	10
220	1	100	10
220	1	100	10
230	1	100	11
230	1	100	11
230	1	100	11
230	1	100	10
230	1	100	10

x	y	z	d
240	1	100	10
240	1	100	11
240	1	100	10
240	1	100	10
240	1	100	10
250	1	100	10
250	1	100	10
250	1	100	10
250	1	100	10
250	1	100	10
260	1	100	10
260	1	100	10
260	1	100	10
260	1	100	10
260	1	100	10
270	1	100	9
270	1	100	9
270	1	100	10
270	1	100	9
270	1	100	10
280	1	100	9
280	1	100	9
280	1	100	9
280	1	100	9
280	1	100	9
290	1	100	9
290	1	100	9
290	1	100	9
290	1	100	9
290	1	100	9
300	1	100	8
300	1	100	9
300	1	100	9
300	1	100	9
300	1	100	9

表 4：The relationship between y and d

x	y	z	d
50	1	100	27
50	1	100	30
50	1	100	26
50	1	100	29
50	1	100	30
50	2	100	32
50	2	100	32
50	2	100	32
50	2	100	32
50	2	100	31
50	4	100	36
50	4	100	35
50	4	100	36
50	4	100	36
50	4	100	36
50	8	100	47
50	8	100	42
50	8	100	42
50	8	100	43
50	8	100	44
50	16	100	53
50	16	100	58
50	16	100	55
50	16	100	55
50	16	100	55
50	32	100	73
50	32	100	77
50	32	100	75
50	32	100	79
50	32	100	78
50	64	100	117
50	64	100	104
50	64	100	112
50	64	100	109
50	64	100	108
50	128	100	156
50	128	100	147
50	128	100	148
50	128	100	150
50	128	100	144
50	256	100	218
50	256	100	191
50	256	100	222
50	256	100	230
50	256	100	237

x	y	z	d
100	1	100	19
100	1	100	16
100	1	100	17
100	1	100	18
100	1	100	17
100	2	100	19
100	2	100	20
100	2	100	20
100	2	100	20
100	2	100	20
100	4	100	25
100	4	100	24
100	4	100	28
100	4	100	26
100	4	100	25
100	8	100	32
100	8	100	34
100	8	100	33
100	8	100	33
100	8	100	33
100	16	100	44
100	16	100	46
100	16	100	48
100	16	100	48
100	16	100	46
100	32	100	69
100	32	100	67
100	32	100	65
100	32	100	68
100	32	100	64
100	64	100	95
100	64	100	93
100	64	100	91
100	64	100	94
100	64	100	97
100	128	100	145
100	128	100	146
100	128	100	134
100	128	100	140
100	128	100	146
100	256	100	198
100	256	100	204
100	256	100	217
100	256	100	214
100	256	100	211

x	y	z	d
150	1	100	14
150	1	100	13
150	1	100	14
150	1	100	14
150	1	100	13
150	2	100	17
150	2	100	16
150	2	100	16
150	2	100	17
150	2	100	16
150	4	100	22
150	4	100	21
150	4	100	20
150	4	100	21
150	4	100	23
150	8	100	30
150	8	100	28
150	8	100	27
150	8	100	29
150	8	100	29
150	16	100	42
150	16	100	38
150	16	100	41
150	16	100	40
150	16	100	41
150	32	100	57
150	32	100	58
150	32	100	60
150	32	100	59
150	32	100	66
150	64	100	94
150	64	100	91
150	64	100	93
150	64	100	90
150	64	100	87
150	128	100	127
150	128	100	137
150	128	100	132
150	128	100	142
150	128	100	138
150	256	100	212
150	256	100	215
150	256	100	216
150	256	100	213
150	256	100	221

x	y	z	d
200	1	100	11
200	1	100	12
200	1	100	11
200	1	100	12
200	1	100	11
200	2	100	14
200	2	100	15
200	2	100	14
200	2	100	14
200	2	100	14
200	4	100	19
200	4	100	19
200	4	100	19
200	4	100	19
200	4	100	18
200	8	100	26
200	8	100	27
200	8	100	27
200	8	100	27
200	8	100	26
200	16	100	38
200	16	100	38
200	16	100	37
200	16	100	38
200	16	100	39
200	32	100	60
200	32	100	55
200	32	100	57
200	32	100	57
200	32	100	60
200	64	100	84
200	64	100	88
200	64	100	87
200	64	100	87
200	64	100	83
200	128	100	136
200	128	100	130
200	128	100	132
200	128	100	122
200	128	100	128
200	256	100	211
200	256	100	207
200	256	100	197
200	256	100	208
200	256	100	201

x	y	z	d
250	1	100	10
250	1	100	10
250	1	100	10
250	1	100	10
250	1	100	10
250	2	100	13
250	2	100	13
250	2	100	13
250	2	100	13
250	2	100	12
250	4	100	18
250	4	100	18
250	4	100	17
250	4	100	17
250	4	100	17
250	8	100	25
250	8	100	26
250	8	100	26
250	8	100	25
250	8	100	24
250	16	100	36
250	16	100	37
250	16	100	37
250	16	100	38
250	16	100	35
250	32	100	56
250	32	100	54
250	32	100	56
250	32	100	53
250	32	100	57
250	64	100	86
250	64	100	82
250	64	100	82
250	64	100	86
250	64	100	82
250	128	100	135
250	128	100	129
250	128	100	132
250	128	100	125
250	128	100	128
250	256	100	223
250	256	100	195
250	256	100	189
250	256	100	189
250	256	100	205

x	y	z	d
300	1	100	8
300	1	100	9
300	1	100	9
300	1	100	9
300	1	100	9
300	2	100	12
300	2	100	12
300	2	100	12
300	2	100	12
300	2	100	12
300	4	100	16
300	4	100	16
300	4	100	16
300	4	100	17
300	4	100	16
300	8	100	24
300	8	100	22
300	8	100	23
300	8	100	24
300	8	100	23
300	16	100	34
300	16	100	35
300	16	100	35
300	16	100	36
300	16	100	36
300	32	100	54
300	32	100	53
300	32	100	53
300	32	100	54
300	32	100	54
300	64	100	80
300	64	100	86
300	64	100	79
300	64	100	78
300	64	100	82
300	128	100	133
300	128	100	136
300	128	100	133
300	128	100	125
300	128	100	125
300	256	100	179
300	256	100	197
300	256	100	194
300	256	100	197
300	256	100	185

表 5：The relationship between z and d

x	y	z	d
50	1	50	28
50	1	100	27
50	1	150	27
50	1	200	27
50	1	250	27
50	1	300	27
50	1	50	26
50	1	100	30
50	1	150	28
50	1	200	28
50	1	250	28
50	1	300	29
50	1	50	28
50	1	100	26
50	1	150	27
50	1	200	27
50	1	250	28
50	1	300	28
50	1	50	30
50	1	100	29
50	1	150	28
50	1	200	29
50	1	250	28
50	1	300	30
50	1	50	28
50	1	100	30
50	1	150	30
50	1	200	29
50	1	250	30
50	1	300	31

x	y	z	d
100	1	50	19
100	1	100	19
100	1	150	18
100	1	200	19
100	1	250	18
100	1	300	18
100	1	50	19
100	1	100	16
100	1	150	18
100	1	200	17
100	1	250	19
100	1	300	18
100	1	50	18
100	1	100	17
100	1	150	17
100	1	200	17
100	1	250	17
100	1	300	17
100	1	50	18
100	1	100	18
100	1	150	17
100	1	200	18
100	1	250	17
100	1	300	17
100	1	50	17
100	1	100	17
100	1	150	18
100	1	200	16
100	1	250	17
100	1	300	16

x	y	z	d
150	1	50	14
150	1	100	14
150	1	150	13
150	1	200	13
150	1	250	13
150	1	300	13
150	1	50	14
150	1	100	13
150	1	150	13
150	1	200	13
150	1	250	14
150	1	300	13
150	1	50	14
150	1	100	14
150	1	150	14
150	1	200	14
150	1	250	14
150	1	300	14
150	1	50	13
150	1	100	14
150	1	150	14
150	1	200	14
150	1	250	13
150	1	300	14
150	1	50	14
150	1	100	13
150	1	150	13
150	1	200	13
150	1	250	13
150	1	300	13

x	y	z	d
200	1	50	12
200	1	100	11
200	1	150	11
200	1	200	11
200	1	250	11
200	1	300	11
200	1	50	11
200	1	100	12
200	1	150	11
200	1	200	11
200	1	250	11
200	1	300	11
200	1	50	11
200	1	100	11
200	1	150	12
200	1	200	12
200	1	250	12
200	1	300	11
200	1	50	11
200	1	100	12
200	1	150	12
200	1	200	11
200	1	250	12
200	1	300	11
200	1	50	11
200	1	100	11
200	1	150	11
200	1	200	12
200	1	250	12
200	1	300	11

x	y	z	d
250	1	50	10
250	1	100	10
250	1	150	10
250	1	200	10
250	1	250	10
250	1	300	10
250	1	50	10
250	1	100	10
250	1	150	10
250	1	200	10
250	1	250	10
250	1	300	10
250	1	50	10
250	1	100	10
250	1	150	10
250	1	200	10
250	1	250	10
250	1	300	10
250	1	50	10
250	1	100	10
250	1	150	10
250	1	200	10
250	1	250	10
250	1	300	10
250	1	50	10
250	1	100	10
250	1	150	10
250	1	200	10
250	1	250	10
250	1	300	10

x	y	z	d
300	1	50	9
300	1	100	8
300	1	150	9
300	1	200	9
300	1	250	9
300	1	300	9
300	1	50	8
300	1	100	9
300	1	150	9
300	1	200	9
300	1	250	9
300	1	300	9
300	1	50	9
300	1	100	9
300	1	150	9
300	1	200	9
300	1	250	9
300	1	300	9
300	1	50	9
300	1	100	9
300	1	150	9
300	1	200	9
300	1	250	9
300	1	300	9
300	1	50	8
300	1	100	9
300	1	150	9
300	1	200	9
300	1	250	8
300	1	300	9

表 6：不同資料結構之 Dijkstra 演算時間表

時間單位：Tick (毫秒)

Data Structure	x	y	z	d	Time
BinaryHeap	100	1	100	17	626
BinaryHeap	100	1	100	18	687
BinaryHeap	100	1	100	18	792
BinaryHeap	100	1	100	18	744
BinaryHeap	100	1	100	17	752
BinaryHeap	100	1	100	18	650
BinaryHeap	100	1	100	19	764
BinaryHeap	100	1	100	19	730
BinaryHeap	100	1	100	17	747
BinaryHeap	100	1	100	17	755
Unordered_Map	100	1	100	17	275
Unordered_Map	100	1	100	18	344
Unordered_Map	100	1	100	18	368
Unordered_Map	100	1	100	18	303
Unordered_Map	100	1	100	17	364
Unordered_Map	100	1	100	18	368
Unordered_Map	100	1	100	19	403
Unordered_Map	100	1	100	19	394
Unordered_Map	100	1	100	17	394
Unordered_Map	100	1	100	17	338

✧ 繪圖套件/ 不同資料結構 Dijkstra 演算法程式來源

序號	繪圖套件/ 資料結構	程式來源
1	opencv	https://docs.opencv.org/master/d3/d96/tutorial_basic_geometric_drawing.html
2	Bibary Heap	http://alrightchiu.github.io/SecondRound/single-source-shortest-pathdijkstras-algorithm.html
3	Unordered_Map	https://codingblocks.com/resources/dijkstra/

✧ 實驗程式碼

```
//Reference
//http://alrightchiu.github.io/SecondRound/single-source-shortest-pathdijkstras-
algorithm.html
//https://codingblocks.com/resources/dijkstra/
//https://docs.opencv.org/master/d3/d96/tutorial basic geometric drawing.html

#include <iostream>
#include <vector>
#include <list>
#include <utility>
#include <iomanip>
#include <cmath>
#include <random>
#include <ctime>
#include <iostream>
#include <time.h>
#include <string>
#include <string.h>
#include <fstream>
#include <array>
#include <cstdlib>
#include <assert.h>
#include <stdio.h>
#include <chrono>
#include <thread>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <bits/stdc++.h>

using namespace std;
using namespace cv;
#define PI 3.1415926
#define w 400
#define CV 8UC3 CV MAKETYPE(CV 8U,3)

//************* Draw Circle ****************//
void myEdge( Mat img ){
  srand(time(NULL));
  int thickness = 1;
  int lineType = LINE 8;
  int shift=1;

  Point myPoints[1][1000];

  for (int i=1; i<=1000; i++) {
      float x = 400 + 300 * sin(i*2*PI/1000);
      float y = 400 + 300 * cos(i*2*PI/1000);
      myPoints[0][i]=Point(x, y);
  }

  for (int j=1; j<=999; j++){
    line( img, myPoints[0][j], myPoints[0][j+1],
    Scalar( 255, 255, 255 ), thickness, lineType, shift );
  }

  for (int i=1; i<=100; i++){
    int A = 1 + rand()%(1000);
    int B = 1 + rand()%(1000);
    if (A==B){
      B = rand()%(1000);
    }

    line( img, myPoints[0][A], myPoints[0][B],
    Scalar( 255, 255, 255 ), thickness, lineType, shift );
  }
}
//------------- Draw Circle ----------------//
```

```
//************** Binary Heap ****************//
struct HeapNode{
    int element, key;
    HeapNode():element(0),key(0){};
    HeapNode(int node, int key):element(node), key(key){};
};

class BinaryHeap{
private:
    std::vector<HeapNode> heap;        // 存放 HeapNode 資料的矩陣
    void swap(struct HeapNode &p1, struct HeapNode &p2);
    int FindPosition(int node);
    int GetParentNode(int node){return std::floor(node/2);};
public:
    BinaryHeap(){              // default constructor 會把 heap[0]給預留
        heap.resize(1);        // 之後若新增 HeapNode, 會從 heap[1]開始新增
    }
    BinaryHeap(int n){
        heap.resize(n + 1);
    }
    bool IsHeapEmpty(){return (heap.size()<1);};

    // Min-Priority Queue
    void MinHeapify(int node, int length);
    void BuildMinHeap(std::vector<int> array);
    void DecreaseKey(int node, int newKey);
    void MinHeapInsert(int node, int key);
    int Minimum();                // 回傳 vertex 的位置 index
    int ExtractMin();             // 回傳 vertex 的位置 index

    //bool IsHeapEmpty(){return (heap.size()<1);};
    //int GetParentNode(int node){return std::floor(node/2);};

    // void HeapSort();
    // Max-Priority Queue
};

const int Max Distance = 10000;
class Graph SP{           // SP serves as Shortest Path
private:
    int num vertex;
    std::vector<std::list<std::pair<int,int>>> AdjList;
    std::vector<int> predecessor, distance;
    std::vector<bool> visited;
public:
    Graph SP():num vertex(0){};
    Graph SP(int n):num vertex(n){
        AdjList.resize(num vertex);
    }
    void AddEdge(int from, int to, int weight);
    void AdjListClear();
    int AdjListSize();
    void PrintDataArray(std::vector<int> array);
    void PrintIntArray(int *array);

    void InitializeSingleSource(int Start);     // 以 Start 作為起點
    void Relax(int X, int Y, int weight);       // edge 方向：from X to Y

    int Dijkstra(int Start = 0, int myEnd=999);       // 需要 Min-Priority Queue
    friend class BinaryHeap;           // 以 Binary Heap 實現 Min-Priority Queue
};
int Graph SP::Dijkstra(int Start, int myEnd){

    InitializeSingleSource(Start);

    BinaryHeap minQueue(num vertex);   // object of min queue
    minQueue.BuildMinHeap(distance);

    visited.resize(num vertex, false);   // initializa visited[] as {0,0,0,...,0}
```

```
    while (!minQueue.IsHeapEmpty()) {
        int u = minQueue.ExtractMin();
        if (u==myEnd){
            //std::list<std::pair<int, int>>::iterator itr = AdjList[u].begin();
            //std::cout<< distance[u]<< std::endl;
            return distance[u];
        }
        else {
        for (std::list<std::pair<int, int>>::iterator itr = AdjList[u].begin();
             itr != AdjList[u].end(); itr++) {

            Relax(u, (*itr).first, (*itr).second);
            minQueue.DecreaseKey((*itr).first, distance[(*itr).first]);
            }
        }
    }
    //std::cout << "\nprint predecessor:\n";
    //PrintDataArray(predecessor);
    //std::cout << "\nprint distance:\n";
    //PrintDataArray(distance);

    return 0;
}
void Graph SP::InitializeSingleSource(int Start){

    distance.resize(num vertex);
    predecessor.resize(num vertex);

    for (int i = 0; i < num vertex; i++) {
        distance[i] = Max Distance;
        predecessor[i] = -1;
    }
    distance[Start] = 0;
}
void Graph SP::Relax(int from, int to, int weight){

    if (distance[to] > distance[from] + weight) {
        distance[to] = distance[from] + weight;
        predecessor[to] = from;
    }
}
void Graph SP::AddEdge(int from, int to, int weight){

    AdjList[from].push back(std::make pair(to,weight));
}
void Graph SP::AdjListClear(){
    AdjList.clear();
}

int Graph SP::AdjListSize(){
    return AdjList.size();
}

//void AdjListClear();
void Graph SP::PrintDataArray(std::vector<int> array){
    for (int i = 0; i < num vertex; i++)
        std::cout << std::setw(5) << i;
    std::cout << std::endl;
    for (int i = 0; i < num vertex; i++)
        std::cout << std::setw(5) << array[i];
    std::cout << std::endl;
}

void BinaryHeap::MinHeapify(int node, int length){

    int left = 2*node,          // 取得 left child
        right = 2*node + 1,     // 取得 right child
        smallest;              // smallest 用來記錄包含 root 與 child, 三者之中 Key 最小的
node
```

```
    if (left <= length && heap[left].key < heap[node].key)
        smallest = left;
    else
        smallest = node;

    if (right <= length && heap[right].key < heap[smallest].key)
        smallest = right;

    if (smallest != node) {                // 如果目前 node 的 Key 不是三者中的最小
        swap(heap[smallest], heap[node]);   // 就調換 node 與三者中 Key 最小的 node 之位置
        MinHeapify(smallest, length);       // 調整新的 subtree 成 Min Heap
    }
}

void BinaryHeap::BuildMinHeap(std::vector<int> array){

    // 將 array[]的資料放進 heap 之矩陣中, 並預留 heap[0] 不做使用
    for (int i = 0; i < array.size(); i++) {
        heap[i + 1].element = i;               // 把 array[]的 idx 視為 element
        heap[i + 1].key = array[i];             // 把 array[]的數值視為 key
    }
    for (int i = (int)heap.size()/2; i >= 1 ; i--) {
        MinHeapify(i, (int)heap.size()-1);     // length 要減一, 因為 heap 從從 1 開始存
放資料
    }
}

void BinaryHeap::swap(struct HeapNode &p1, struct HeapNode &p2){

    struct HeapNode temp = p1;
    p1 = p2;
    p2 = temp;
}

int BinaryHeap::FindPosition(int node){

    int idx = 0;
    for (int i = 1; i < heap.size(); i++) {
        if (heap[i].element == node) {
            idx = i;
        }
    }
    return idx;
}

//class BinaryHeap{
//    bool IsHeapEmpty(){return (heap.size()<1);};
//    int GetParentNode(int node){return std::floor(node/2);};
//};

int BinaryHeap::Minimum(){
    return heap[1].element;
}

int BinaryHeap::ExtractMin(){

    if (IsHeapEmpty()) {
        std::cout << "error: heap is empty\n";
        exit(-1);
    }
    int min = heap[1].element;    // 此時 heap 的第一個 node 具有最小 key 值
                                  // 便以 min 記錄其 element, 最後回傳 min
    // delete the first element/vertex
    heap[1] = heap[heap.size()-1];           // 把最後一個 element 放到第一個位置,
    heap.erase(heap.begin()+heap.size()-1);   // 再刪除最後一個 element
    MinHeapify(1, (int)heap.size());         // 目前, heap[1]具有最大 Key, 需要進行調整

    return min;       // 回傳 heap 中具有最小 key 的 element
}
```

```
void BinaryHeap::DecreaseKey(int node, int newKey){

    int index node = FindPosition(node);      // 找到 node 所在的位置 index

    if (newKey > heap[index node].key) {      // 如果不是把 node 的 Key 下修，便終止此函
式
        //std::cout << "new key is larger than current key\n";
        return;
    }
    heap[index node].key = newKey;           // 更新 node 之 Key 後，需要檢查是否新的
subtree 滿足 Min Heap
    while (index node > 1 && heap[GetParentNode(index node)].key >
heap[index node].key) {
        swap(heap[index node], heap[GetParentNode(index node)]);
        index node = GetParentNode(index node);
    }
}

void BinaryHeap::MinHeapInsert(int node, int key){

    heap.push back(HeapNode(node,key));    // 在 heap[]尾巴新增一個 node
    DecreaseKey(node, key);
}
//------------- Binary Heap   ---------------//


//************* Unordered Map ****************//
template<typename T>
class Graph{
    unordered map<T, list<pair<T,int> > > m;

public:

    void addEdge(T u,T v,int dist,bool bidir=true){
        m[u].push back(make pair(v,dist));
        if(bidir){
            m[v].push back(make pair(u,dist));
        }
    }

    void EdgeClear(){
        m.clear();
    }

    int EdgeSize(){
        return m.size();
    }

    void printAdj(){
        //Let try to print the adj list
        //Iterate over all the key value pairs in the map
        for(auto j:m){

            cout<<j.first<<"->";

            //Iterater over the list of cities
            for(auto l: j.second){
                   cout<<"("<<l.first<<","<<l.second<<")";

            }
            cout<<endl;
        }
    }
    int dijsktraSSSP(T src, T myEnd){

        unordered map<T,int> dist;

        //Set all distance to infinity
        for(auto j:m){
            dist[j.first] = INT_MAX;
```

```
        }

        //Make a set to find a out node with the minimum distance
        set<pair<int, T> > s;

        dist[src] = 0;
        s.insert(make pair(0,src));

        while(!s.empty()){

            //Find the pair at the front.
            auto p =  *(s.begin());
            T node = p.second;

            if (p.second==myEnd){
                //cout << p.first << endl;
                return p.first;
            }
            int nodeDist = p.first;
            s.erase(s.begin());


            //Iterate over neighbours/children of the current node
            for(auto childPair: m[node]){

                if(nodeDist + childPair.second < dist[childPair.first]){


                    //In the set updation of a particular is not possible
                    // we have to remove the old pair, and insert the new pair to
simulation updation
                    T dest = childPair.first;
                    auto f = s.find( make pair(dist[dest],dest));
                    if(f!=s.end()){
                        s.erase(f);
                    }

                    //Insert the new pair
                    dist[dest] = nodeDist + childPair.second;
                    s.insert(make pair(dist[dest],dest));

                }
            }
        }
        //Lets print distance to all other node from src
        //for(auto d:dist){

        //    cout<<d.first<<" is located at distance of  "<<d.second<<endl;
        //}
        return 0;
    }

};
//------------- Unordered Map -------------//

static double sys time()
{
    return static cast<double>(clock()) / 1000;
}

void ExcelOutput(float d, int x, int y, int z, int zCount, int myTime, string
DS) {
    ofstream myfile;
    string filename = DS + to string(sys time()) + ".csv";
    myfile.open(filename);
    myfile << "x, y, z, zCount, d, myTime\n";
    myfile << x << ",";
    myfile << y << ",";
    myfile << z << ",";
    myfile << zCount << ",";
```

```
    myfile << d << ",";
    myfile << myTime;
    myfile << "\n";
    myfile.close();
}

int main(){

    // ****** Draw Circle ******
    char rook window[] = "Drawing Small World";
    Mat rook image = Mat::zeros( w, w, CV 8UC3 );
    myEdge( rook image );
    imshow( rook window, rook image );
    moveWindow( rook window, w, 200 );
    waitKey( 0 );
    // ------ Draw Circle ------

    // ****** x: # of edges, y: weight of edge, z: # pairs, myRepeat: # repeat
    // ****** < x vs d (y=1,z=100)>, < y vs d (z=100)>, < z vs d (y=1)> to test
Relation
    // ****** <set myRepeat = 10, x=100, y=1, z=100> to test Data Structure
    random device rd;                              //梅森旋轉演算法
    mt19937 generator(rd());
    uniform int distribution<int> unif(0, 999);            //隨機取號
    srand(time(NULL));                             //隨機取號

    int xCount, yCount, zCount;
    int myTotalDistance[2];
    int myTime 1, myTime 2;
    clock t start , end ;

    int myRepeat = 5;                              //重複次數
    for (int i=1; i<= myRepeat; i++){
       //Graph SP gWorld(1000);                     //BinaryHeap
       Graph<int> gWorld map;                      //Unordered Map

       yCount=1;
       for (int y=1; y<=256; y=2*y){

          xCount=1;
          for (int x=0; x<=300; x=x+10){

             if (x==0){
                //gWorld.AdjListClear();
                gWorld map.EdgeClear();

                for (int m=0; m<1000; m++){

                   if (m==0){
                      //gWorld.AddEdge(0, 999, 1);gWorld.AddEdge(0, 1, 1);
                      gWorld map.addEdge(0, 999, 1);gWorld map.addEdge(0, 1,
1);
                   }
                   else if (m==999){
                      //gWorld.AddEdge(999, 998, 1);gWorld.AddEdge(999, 0, 1);
                      gWorld map.addEdge(999, 998, 1);gWorld map.addEdge(999,
0, 1);
                   }
                   else{
                      //gWorld.AddEdge(m, m-1, 1);gWorld.AddEdge(m, m+1, 1);
                      gWorld map.addEdge(m, m-1, 1);gWorld map.addEdge(m, m+1,
1);
                   }
                }
             }
             else {
                gWorld map.EdgeClear();
                for (int m=0; m<1000; m++){
                   if (m==0){
                      //gWorld.AddEdge(0, 999, 1);gWorld.AddEdge(0, 1, 1);
```

```
                        gWorld map.addEdge(0, 999, 1);gWorld map.addEdge(0, 1,
1);
                    }
                    else if (m==999){
                        //gWorld.AddEdge(999, 998, 1);gWorld.AddEdge(999, 0, 1);
                        gWorld map.addEdge(999, 998, 1);gWorld map.addEdge(999,
0, 1);
                    }
                    else{
                        //gWorld.AddEdge(m, m-1, 1);gWorld.AddEdge(m, m+1, 1);
                        gWorld map.addEdge(m, m-1, 1);gWorld map.addEdge(m, m+1,
1);
                    }
                }

                for (int i=1; i<=x; i++){
                    int A = rand()%(1000);
                    int B = rand()%(1000);
                    if (A==B){
                    B = (int)(unif(generator));
                    }
                    //gWorld.AddEdge(A, B, y);gWorld.AddEdge(B, A, y);
                    gWorld map.addEdge(A, B, y);gWorld map.addEdge(B, A, y);
                }
            }

            zCount=1;
            for (int z =50; z<=300 ; z=z+50){
                myTotalDistance[1]=0;
                myTotalDistance[2]=0;
                myTime 1=0;
                myTime 2=0;

                for (int i=1; i<=z; i++){

                    int A = rand()%(1000);
                    int B = rand()%(1000);
                    if (A==B){
                        B = rand()%(1000);
                    }
                    if (A==B){
                        B = (int)(unif(generator));
                    }

                    //start  = clock();
                    //myTotalDistance[1] += gWorld.Dijkstra(A,B);
                    //end  = clock();
                    //myTime 1 =  myTime 1 + (end -start );
                    //cout << "A= " << A << "  " << "B= " << B << endl;
                    //cout << "gWorld distance= " << gWorld.Dijkstra(A,B)  <<
endl;

                    start  = clock();
                    myTotalDistance[2] += gWorld map.dijsktraSSSP(A,B);
                    end  = clock();
                    myTime 2 =  myTime 2 + (end -start );
                }
                //ExcelOutput(static cast<float>(myTotalDistance[1]/z), x, y, z,
zCount, myTime 1, "BinaryHeap");
                ExcelOutput(static cast<float>(myTotalDistance[2]/z), x, y, z,
zCount, myTime 2, "Map");

                std::cout << "zCount= " << zCount << endl;
                zCount += 1;
            }
            std::cout << "xCount= " << xCount << endl;
            xCount += 1;
        }
        std::cout << "yCount= " << yCount << endl;
        yCount += 1;
```

```
        }
    }

    return 0;
}
```