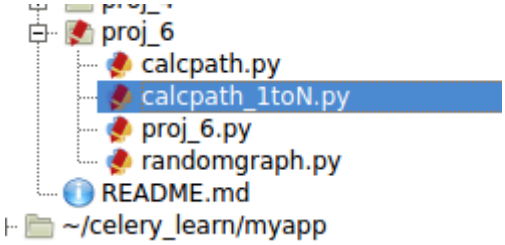
Project 6 报告：

自己设计算法与规范 Dijkstra 算法计算单源多宿网络单纯算法比较：
自己的算法由以前（proj3-proj4）的算法改造而成，规范 Dijkstra 算法由 networkx 模块提供。
算法代码见文件夹中附件：



高亮部分为自己的算法设计模块。

展示代码：

```
#encoding:utf-8
from calcpath_1toN import *

if __name__ == '__main__':
    """
    Cmp All-Pairs!
    """

    N = 8
    M = 3
    calc_buffer = []
    G = randomgraph.get_barabasi_albert_graph(n=N, m=M)
    print '-----------------Graph------------------------'
    for _ in G.nodes():
        print _,':',G[_]
    print '----------------------------------------------'
    for _ in range(N):
        ret = CalcPath_1toN(G=G, start=_, end=[0,1,2,3,4,5,6])
        calc_buffer.append(ret)
    """
    Clac the period 100 times
    """
    start = int(time.time()*1000)
    for _ in range(100):
```

```
        for i in calc_buffer:
            i.calc_path()
    stop = int(time.time()*1000)
    period = stop - start
    print 'Designed by my Alg! cost %d ms per 100 times' % period

    #show result
    for ret in calc_buffer:
        r = ret.show_result()
        for i,k in r.items():
            print ret.start,'--->',i,'PATH:',str(k),'WEIGHT',ret.route[i][0]



    print '------------------DEFAULT ALG------------------'
    start = int(time.time()*1000)
    for _ in range(100):
        ret1 = nx.all_pairs_dijkstra_path(G)
    stop = int(time.time()*1000)
    period = stop - start
    print 'Dijkstra Alg! cost %d ms per 100 times' % period
    for ret in calc_buffer:
        for i in ret.end:
            route = ret1[ret.start][i]
            print ret.start,'--->',i,'PATH:',str(route)
```

结果由于每次生成随机的图，在这里展示两次的结果：

1．取 N=8 时，对于某个随机图

----------Graph----------
0 : {3: {'capability': 11, 'passing_rate': 0.9, 'weight': 43}, 5: {'capability': 4, 'passing_rate': 0.27, 'weight': 47}}
1 : {3: {'capability': 10, 'passing_rate': 0.86, 'weight': 85}, 4: {'capability': 23, 'passing_rate': 0.59, 'weight': 34}, 6: {'capability': 10, 'passing_rate': 0.56, 'weight': 59}}
2 : {3: {'capability': 16, 'passing_rate': 0.5, 'weight': 33}, 4: {'capability': 4, 'passing_rate': 0.71, 'weight': 36}, 7: {'capability': 40, 'passing_rate': 0.23, 'weight': 11}}
3 : {0: {'capability': 11, 'passing_rate': 0.9, 'weight': 43}, 1: {'capability': 10, 'passing_rate': 0.86, 'weight': 85}, 2: {'capability': 16, 'passing_rate': 0.5, 'weight': 33}, 4: {'capability': 28, 'passing_rate': 0.66, 'weight': 78}, 5: {'capability': 30, 'passing_rate': 0.36, 'weight': 43}, 6: {'capability': 22, 'passing_rate': 0.77, 'weight': 36}}
4 : {1: {'capability': 23, 'passing_rate': 0.59, 'weight': 34}, 2: {'capability': 4, 'passing_rate': 0.71, 'weight': 36}, 3: {'capability': 28, 'passing_rate': 0.66, 'weight': 78}, 5: {'capability': 5,

'passing_rate': 0.09, 'weight': 88}}
5 : {0: {'capability': 4, 'passing_rate': 0.27, 'weight': 47}, 3: {'capability': 30, 'passing_rate': 0.36, 'weight': 43}, 4: {'capability': 5, 'passing_rate': 0.09, 'weight': 88}, 6: {'capability': 40, 'passing_rate': 0.24, 'weight': 25}, 7: {'capability': 26, 'passing_rate': 0.94, 'weight': 21}}
6 : {1: {'capability': 10, 'passing_rate': 0.56, 'weight': 59}, 3: {'capability': 22, 'passing_rate': 0.77, 'weight': 36}, 5: {'capability': 40, 'passing_rate': 0.24, 'weight': 25}, 7: {'capability': 19, 'passing_rate': 0.77, 'weight': 18}}
7 : {2: {'capability': 40, 'passing_rate': 0.23, 'weight': 11}, 5: {'capability': 26, 'passing_rate': 0.94, 'weight': 21}, 6: {'capability': 19, 'passing_rate': 0.77, 'weight': 18}}

Designed by my Alg! cost 639 ms per 100 times
0 ---> 0 PATH: [0, 0] WEIGHT 0
0 ---> 1 PATH: [0, 3, 1] WEIGHT 128
0 ---> 2 PATH: [0, 3, 2] WEIGHT 76
0 ---> 3 PATH: [0, 3] WEIGHT 43
0 ---> 4 PATH: [0, 3, 2, 4] WEIGHT 112
0 ---> 5 PATH: [0, 5] WEIGHT 47
0 ---> 6 PATH: [0, 5, 6] WEIGHT 72
1 ---> 0 PATH: [1, 3, 0] WEIGHT 128
1 ---> 1 PATH: [1, 1] WEIGHT 0
1 ---> 2 PATH: [1, 4, 2] WEIGHT 70
1 ---> 3 PATH: [1, 3] WEIGHT 85
1 ---> 4 PATH: [1, 4] WEIGHT 34
1 ---> 5 PATH: [1, 6, 5] WEIGHT 84
1 ---> 6 PATH: [1, 6] WEIGHT 59
2 ---> 0 PATH: [2, 3, 0] WEIGHT 76
2 ---> 1 PATH: [2, 4, 1] WEIGHT 70
2 ---> 2 PATH: [2, 2] WEIGHT 0
2 ---> 3 PATH: [2, 3] WEIGHT 33
2 ---> 4 PATH: [2, 4] WEIGHT 36
2 ---> 5 PATH: [2, 7, 5] WEIGHT 32
2 ---> 6 PATH: [2, 7, 6] WEIGHT 29
3 ---> 0 PATH: [3, 0] WEIGHT 43
3 ---> 1 PATH: [3, 1] WEIGHT 85
3 ---> 2 PATH: [3, 2] WEIGHT 33
3 ---> 3 PATH: [3, 3] WEIGHT 0
3 ---> 4 PATH: [3, 2, 4] WEIGHT 69
3 ---> 5 PATH: [3, 5] WEIGHT 43
3 ---> 6 PATH: [3, 6] WEIGHT 36
4 ---> 0 PATH: [4, 2, 3, 0] WEIGHT 112
4 ---> 1 PATH: [4, 1] WEIGHT 34
4 ---> 2 PATH: [4, 2] WEIGHT 36
4 ---> 3 PATH: [4, 2, 3] WEIGHT 69
4 ---> 4 PATH: [4, 4] WEIGHT 0

4 ---> 5 PATH: [4, 2, 7, 5] WEIGHT 68
4 ---> 6 PATH: [4, 2, 7, 6] WEIGHT 65
5 ---> 0 PATH: [5, 0] WEIGHT 47
5 ---> 1 PATH: [5, 6, 1] WEIGHT 84
5 ---> 2 PATH: [5, 7, 2] WEIGHT 32
5 ---> 3 PATH: [5, 3] WEIGHT 43
5 ---> 4 PATH: [5, 7, 2, 4] WEIGHT 68
5 ---> 5 PATH: [5, 5] WEIGHT 0
5 ---> 6 PATH: [5, 6] WEIGHT 25
6 ---> 0 PATH: [6, 5, 0] WEIGHT 72
6 ---> 1 PATH: [6, 1] WEIGHT 59
6 ---> 2 PATH: [6, 7, 2] WEIGHT 29
6 ---> 3 PATH: [6, 3] WEIGHT 36
6 ---> 4 PATH: [6, 7, 2, 4] WEIGHT 65
6 ---> 5 PATH: [6, 5] WEIGHT 25
6 ---> 6 PATH: [6, 6] WEIGHT 0
7 ---> 0 PATH: [7, 5, 0] WEIGHT 68
7 ---> 1 PATH: [7, 6, 1] WEIGHT 77
7 ---> 2 PATH: [7, 2] WEIGHT 11
7 ---> 3 PATH: [7, 2, 3] WEIGHT 44
7 ---> 4 PATH: [7, 2, 4] WEIGHT 47
7 ---> 5 PATH: [7, 5] WEIGHT 21
7 ---> 6 PATH: [7, 6] WEIGHT 18
------------------DEFAULT ALG-------------------
Dijkstra Alg! cost 279 ms per 100 times
0 ---> 0 PATH: [0]
0 ---> 1 PATH: [0, 3, 1]
0 ---> 2 PATH: [0, 3, 2]
0 ---> 3 PATH: [0, 3]
0 ---> 4 PATH: [0, 3, 2, 4]
0 ---> 5 PATH: [0, 5]
0 ---> 6 PATH: [0, 5, 6]
1 ---> 0 PATH: [1, 3, 0]
1 ---> 1 PATH: [1]
1 ---> 2 PATH: [1, 4, 2]
1 ---> 3 PATH: [1, 3]
1 ---> 4 PATH: [1, 4]
1 ---> 5 PATH: [1, 6, 5]
1 ---> 6 PATH: [1, 6]
2 ---> 0 PATH: [2, 3, 0]
2 ---> 1 PATH: [2, 4, 1]
2 ---> 2 PATH: [2]
2 ---> 3 PATH: [2, 3]
2 ---> 4 PATH: [2, 4]

2 ---> 5 PATH: [2, 7, 5]
2 ---> 6 PATH: [2, 7, 6]
3 ---> 0 PATH: [3, 0]
3 ---> 1 PATH: [3, 1]
3 ---> 2 PATH: [3, 2]
3 ---> 3 PATH: [3]
3 ---> 4 PATH: [3, 2, 4]
3 ---> 5 PATH: [3, 5]
3 ---> 6 PATH: [3, 6]
4 ---> 0 PATH: [4, 2, 3, 0]
4 ---> 1 PATH: [4, 1]
4 ---> 2 PATH: [4, 2]
4 ---> 3 PATH: [4, 2, 3]
4 ---> 4 PATH: [4]
4 ---> 5 PATH: [4, 2, 7, 5]
4 ---> 6 PATH: [4, 2, 7, 6]
5 ---> 0 PATH: [5, 0]
5 ---> 1 PATH: [5, 6, 1]
5 ---> 2 PATH: [5, 7, 2]
5 ---> 3 PATH: [5, 3]
5 ---> 4 PATH: [5, 7, 2, 4]
5 ---> 5 PATH: [5]
5 ---> 6 PATH: [5, 6]
6 ---> 0 PATH: [6, 5, 0]
6 ---> 1 PATH: [6, 1]
6 ---> 2 PATH: [6, 7, 2]
6 ---> 3 PATH: [6, 3]
6 ---> 4 PATH: [6, 7, 2, 4]
6 ---> 5 PATH: [6, 5]
6 ---> 6 PATH: [6]
7 ---> 0 PATH: [7, 5, 0]
7 ---> 1 PATH: [7, 6, 1]
7 ---> 2 PATH: [7, 2]
7 ---> 3 PATH: [7, 2, 3]
7 ---> 4 PATH: [7, 2, 4]
7 ---> 5 PATH: [7, 5]
7 ---> 6 PATH: [7, 6]

对于 N=9 时的某个无向图

----------Graph----------
0 : {3: {'capability': 36, 'passing_rate': 0.34, 'weight': 74}, 4: {'capability': 25, 'passing_rate': 0.47, 'weight': 34}, 5: {'capability': 25, 'passing_rate': 0.45, 'weight': 95}}
1 : {8: {'capability': 46, 'passing_rate': 0.82, 'weight': 38}, 3: {'capability': 4, 'passing_rate': 0.82,

'weight': 25}, 6: {'capability': 6, 'passing_rate': 0.69, 'weight': 14}}
2 : {3: {'capability': 50, 'passing_rate': 0.35, 'weight': 27}, 4: {'capability': 11, 'passing_rate': 0.92, 'weight': 84}, 5: {'capability': 37, 'passing_rate': 0.47, 'weight': 28}, 6: {'capability': 22, 'passing_rate': 0.27, 'weight': 70}}
3 : {0: {'capability': 36, 'passing_rate': 0.34, 'weight': 74}, 1: {'capability': 4, 'passing_rate': 0.82, 'weight': 25}, 2: {'capability': 50, 'passing_rate': 0.35, 'weight': 27}, 4: {'capability': 32, 'passing_rate': 0.3, 'weight': 66}, 6: {'capability': 27, 'passing_rate': 0.27, 'weight': 13}, 7: {'capability': 26, 'passing_rate': 0.81, 'weight': 68}}
4 : {0: {'capability': 25, 'passing_rate': 0.47, 'weight': 34}, 2: {'capability': 11, 'passing_rate': 0.92, 'weight': 84}, 3: {'capability': 32, 'passing_rate': 0.3, 'weight': 66}, 5: {'capability': 28, 'passing_rate': 0.27, 'weight': 29}, 7: {'capability': 6, 'passing_rate': 0.86, 'weight': 57}, 8: {'capability': 48, 'passing_rate': 0.93, 'weight': 88}}
5 : {0: {'capability': 25, 'passing_rate': 0.45, 'weight': 95}, 2: {'capability': 37, 'passing_rate': 0.47, 'weight': 28}, 4: {'capability': 28, 'passing_rate': 0.27, 'weight': 29}, 7: {'capability': 32, 'passing_rate': 0.44, 'weight': 82}}
6 : {1: {'capability': 6, 'passing_rate': 0.69, 'weight': 14}, 2: {'capability': 22, 'passing_rate': 0.27, 'weight': 70}, 3: {'capability': 27, 'passing_rate': 0.27, 'weight': 13}}
7 : {8: {'capability': 37, 'passing_rate': 0.7, 'weight': 78}, 3: {'capability': 26, 'passing_rate': 0.81, 'weight': 68}, 4: {'capability': 6, 'passing_rate': 0.86, 'weight': 57}, 5: {'capability': 32, 'passing_rate': 0.44, 'weight': 82}}
8 : {1: {'capability': 46, 'passing_rate': 0.82, 'weight': 38}, 4: {'capability': 48, 'passing_rate': 0.93, 'weight': 88}, 7: {'capability': 37, 'passing_rate': 0.7, 'weight': 78}}

Designed by my Alg! cost 805 ms per 100 times
0 ---> 0 PATH: [0, 0] WEIGHT 0
0 ---> 1 PATH: [0, 3, 1] WEIGHT 99
0 ---> 2 PATH: [0, 4, 5, 2] WEIGHT 91
0 ---> 3 PATH: [0, 3] WEIGHT 74
0 ---> 4 PATH: [0, 4] WEIGHT 34
0 ---> 5 PATH: [0, 4, 5] WEIGHT 63
0 ---> 6 PATH: [0, 3, 6] WEIGHT 87
1 ---> 0 PATH: [1, 3, 0] WEIGHT 99
1 ---> 1 PATH: [1, 1] WEIGHT 0
1 ---> 2 PATH: [1, 3, 2] WEIGHT 52
1 ---> 3 PATH: [1, 3] WEIGHT 25
1 ---> 4 PATH: [1, 3, 4] WEIGHT 91
1 ---> 5 PATH: [1, 3, 2, 5] WEIGHT 80
1 ---> 6 PATH: [1, 6] WEIGHT 14
2 ---> 0 PATH: [2, 5, 4, 0] WEIGHT 91
2 ---> 1 PATH: [2, 3, 1] WEIGHT 52
2 ---> 2 PATH: [2, 2] WEIGHT 0
2 ---> 3 PATH: [2, 3] WEIGHT 27
2 ---> 4 PATH: [2, 5, 4] WEIGHT 57
2 ---> 5 PATH: [2, 5] WEIGHT 28

2 ---> 6 PATH: [2, 3, 6] WEIGHT 40
3 ---> 0 PATH: [3, 0] WEIGHT 74
3 ---> 1 PATH: [3, 1] WEIGHT 25
3 ---> 2 PATH: [3, 2] WEIGHT 27
3 ---> 3 PATH: [3, 3] WEIGHT 0
3 ---> 4 PATH: [3, 4] WEIGHT 66
3 ---> 5 PATH: [3, 2, 5] WEIGHT 55
3 ---> 6 PATH: [3, 6] WEIGHT 13
4 ---> 0 PATH: [4, 0] WEIGHT 34
4 ---> 1 PATH: [4, 3, 1] WEIGHT 91
4 ---> 2 PATH: [4, 5, 2] WEIGHT 57
4 ---> 3 PATH: [4, 3] WEIGHT 66
4 ---> 4 PATH: [4, 4] WEIGHT 0
4 ---> 5 PATH: [4, 5] WEIGHT 29
4 ---> 6 PATH: [4, 3, 6] WEIGHT 79
5 ---> 0 PATH: [5, 4, 0] WEIGHT 63
5 ---> 1 PATH: [5, 2, 3, 1] WEIGHT 80
5 ---> 2 PATH: [5, 2] WEIGHT 28
5 ---> 3 PATH: [5, 2, 3] WEIGHT 55
5 ---> 4 PATH: [5, 4] WEIGHT 29
5 ---> 5 PATH: [5, 5] WEIGHT 0
5 ---> 6 PATH: [5, 2, 3, 6] WEIGHT 68
6 ---> 0 PATH: [6, 3, 0] WEIGHT 87
6 ---> 1 PATH: [6, 1] WEIGHT 14
6 ---> 2 PATH: [6, 3, 2] WEIGHT 40
6 ---> 3 PATH: [6, 3] WEIGHT 13
6 ---> 4 PATH: [6, 3, 4] WEIGHT 79
6 ---> 5 PATH: [6, 3, 2, 5] WEIGHT 68
6 ---> 6 PATH: [6, 6] WEIGHT 0
7 ---> 0 PATH: [7, 4, 0] WEIGHT 91
7 ---> 1 PATH: [7, 3, 1] WEIGHT 93
7 ---> 2 PATH: [7, 3, 2] WEIGHT 95
7 ---> 3 PATH: [7, 3] WEIGHT 68
7 ---> 4 PATH: [7, 4] WEIGHT 57
7 ---> 5 PATH: [7, 5] WEIGHT 82
7 ---> 6 PATH: [7, 3, 6] WEIGHT 81
8 ---> 0 PATH: [8, 4, 0] WEIGHT 122
8 ---> 1 PATH: [8, 1] WEIGHT 38
8 ---> 2 PATH: [8, 1, 3, 2] WEIGHT 90
8 ---> 3 PATH: [8, 1, 3] WEIGHT 63
8 ---> 4 PATH: [8, 4] WEIGHT 88
8 ---> 5 PATH: [8, 4, 5] WEIGHT 117
8 ---> 6 PATH: [8, 1, 6] WEIGHT 52
----------DEFAULT ALG----------

Dijkstra Alg! cost 364 ms per 100 times
0 ---> 0 PATH: [0]
0 ---> 1 PATH: [0, 3, 1]
0 ---> 2 PATH: [0, 4, 5, 2]
0 ---> 3 PATH: [0, 3]
0 ---> 4 PATH: [0, 4]
0 ---> 5 PATH: [0, 4, 5]
0 ---> 6 PATH: [0, 3, 6]
1 ---> 0 PATH: [1, 3, 0]
1 ---> 1 PATH: [1]
1 ---> 2 PATH: [1, 3, 2]
1 ---> 3 PATH: [1, 3]
1 ---> 4 PATH: [1, 3, 4]
1 ---> 5 PATH: [1, 3, 2, 5]
1 ---> 6 PATH: [1, 6]
2 ---> 0 PATH: [2, 5, 4, 0]
2 ---> 1 PATH: [2, 3, 1]
2 ---> 2 PATH: [2]
2 ---> 3 PATH: [2, 3]
2 ---> 4 PATH: [2, 5, 4]
2 ---> 5 PATH: [2, 5]
2 ---> 6 PATH: [2, 3, 6]
3 ---> 0 PATH: [3, 0]
3 ---> 1 PATH: [3, 1]
3 ---> 2 PATH: [3, 2]
3 ---> 3 PATH: [3]
3 ---> 4 PATH: [3, 4]
3 ---> 5 PATH: [3, 2, 5]
3 ---> 6 PATH: [3, 6]
4 ---> 0 PATH: [4, 0]
4 ---> 1 PATH: [4, 3, 1]
4 ---> 2 PATH: [4, 5, 2]
4 ---> 3 PATH: [4, 3]
4 ---> 4 PATH: [4]
4 ---> 5 PATH: [4, 5]
4 ---> 6 PATH: [4, 3, 6]
5 ---> 0 PATH: [5, 4, 0]
5 ---> 1 PATH: [5, 2, 3, 1]
5 ---> 2 PATH: [5, 2]
5 ---> 3 PATH: [5, 2, 3]
5 ---> 4 PATH: [5, 4]
5 ---> 5 PATH: [5]
5 ---> 6 PATH: [5, 2, 3, 6]
6 ---> 0 PATH: [6, 3, 0]

6 ---> 1 PATH: [6, 1]
6 ---> 2 PATH: [6, 3, 2]
6 ---> 3 PATH: [6, 3]
6 ---> 4 PATH: [6, 3, 4]
6 ---> 5 PATH: [6, 3, 2, 5]
6 ---> 6 PATH: [6]
7 ---> 0 PATH: [7, 4, 0]
7 ---> 1 PATH: [7, 3, 1]
7 ---> 2 PATH: [7, 3, 2]
7 ---> 3 PATH: [7, 3]
7 ---> 4 PATH: [7, 4]
7 ---> 5 PATH: [7, 5]
7 ---> 6 PATH: [7, 3, 6]
8 ---> 0 PATH: [8, 4, 0]
8 ---> 1 PATH: [8, 1]
8 ---> 2 PATH: [8, 1, 3, 2]
8 ---> 3 PATH: [8, 1, 3]
8 ---> 4 PATH: [8, 4]
8 ---> 5 PATH: [8, 4, 5]
8 ---> 6 PATH: [8, 1, 6]

结果分析：

正确性

可以看到与 networkx 内置的 all-pairs-dijkstra 算法结果完全相同，说明自己的算法是没有错误的

时间比较

自己的结果计算速度慢了规范 dijkstra 算法将近一半，但是自己的算法运用了 dijkstra 的算法思想，运算可能是因为其他消耗过多（比如对象的创建，计算时堆栈维护频繁，属于程序算法设计的缺陷），理论上来讲两个算法的结果耗时应该大致相同