4次算法实验全部结束，代码在不断优化，我也收获了许多，代码均托管在
https://github.com/Tziheng/2021USTC-Algorithm

- **在每次实验中，可以看见，每个项目都是分成了许多文件来完成，而不是一个xxx.c文件，这样做也是受到'操作系统设计'课程的启发，这样有利于维护，且提高可读性！**
- **在lab1实验中，我先是在自己电脑win10跑的，但是time.txt都是0，无法得出具体时间，于是我想到了去vlab上运行(大概我也是最早提出的)，但是放入ubuntu gcc之后有段错误，我在自己电脑上又装了WSL，最终发现ubuntu中"..\\input\\input.txt"是运行不起来的，最终改成"../input/input.txt"成功运行，windows你害人不浅。**
  WSL：win下ubuntu子系统，之后我甚至装了图形界面，用vnc就可以连上，而且速度超快，完爆VM，于是还把VM卸载了
- **在之后实验中，我也不断遇到一些问题，例如总是重复声明，查找资料，学会了extern的使用！拥有只在.c文件中声明变量，在相应的.h文件中写下extern**

如在lab4中
str.c文件中

```
char T[4500];
```

str.h文件中

```
extern char T[];
```

这样避免了重复声明的问题，而且将输入全部存入str.c文件中，也利于维护。

- **在lab3里也首次开创的使用了局部函数(Johnson重修赋予权重)，使得代码更加像伪代码，不能说完全一样，只能说几乎一模一样！**

```
void __Johnson__(int (*w)(int, int)) {
    g1 = copyg(g);
    int s = g1->VN++;
    g1->V[s].degree = 0;
    for (int i = 0; i < s; ++i)
        addEdge(g1, s, i, 0);

    if (!BellmanFord(g1, s, &weight))
        exit(0);

    for (int i = 0; i < g1->VN; ++i)
        h[i] = d[i];

    int w_(Graph* g, int u, int v) {
        return (*w)(u, v) + h[u] - h[v];
    }
    for (int u = 0; u < g->VN; ++u) {
        Dijkstra(g, u, &w_);
        for (int v = 0; v < g->VN; ++v)
            D[u][v] = d[v] + h[v] - h[u];
    }
}

void Johnson(Graph* G, int (*func)(Graph*, int, int)) {
    int w(int u, int v) {
        return (*func)(g, u, v);
    }
    g = G;
    __Johnson__(&w);
}
```

我们将这上面__Jognson__和下面伪代码对比一下，还原度确实很高

```
JOHNSON(G, w)
 1  compute G', where G'.V = G.V ∪ {s},
          G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
          w(s, v) = 0 for all v ∈ G.V
 2  if BELLMAN-FORD(G', w, s) == FALSE
 3     print "the input graph contains a negative-weight cycle"
 4  else for each vertex v ∈ G'.V
 5           set h(v) to the value of δ(s, v)
                computed by the Bellman-Ford algorithm
 6        for each edge (u, v) ∈ G'.E
 7           ŵ(u, v) = w(u, v) + h(u) − h(v)
 8        let D = (d_uv) be a new n × n matrix
 9        for each vertex u ∈ G.V
10           run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11           for each vertex v ∈ G.V
12              d_uv = δ̂(u, v) + h(v) − h(u)
13        return D
```