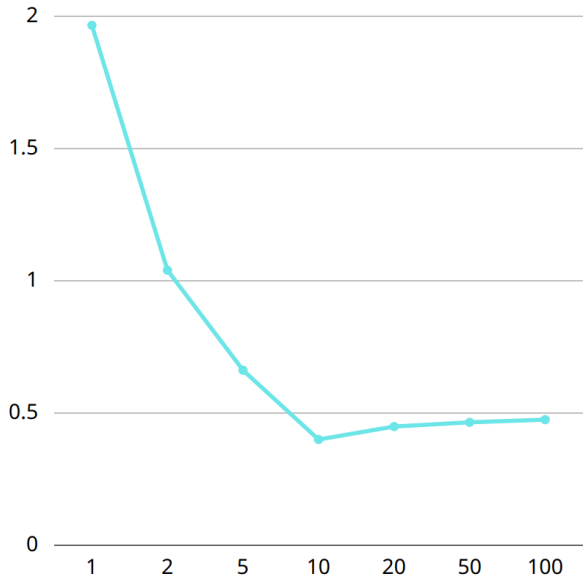
4.
用thread = 2做largeCase_in.txt的execution time結果為1.140, 1.170, 1.013(s)
用thread = 20做largeCase_in.txt的executionl time結果為0.458, 0.474, 0.442(s)
可以知道20個比2個快約2.5倍
5.
下圖X軸為thread數(個)，Y軸為execution time(s)



可以發現1~10個的時間變化大(2秒~0.4秒)，10~100個的時間幾乎沒變，大約在10個速度最快，大約0.4秒，比10個多反而開始越來越慢，可能是因為產生多餘的thread需要的overhead。

程式分析:
1.

```
int global_row, global_col;
int global_quantity;
pthread_mutex_t mutex;    //mutex for finish
pthread_cond_t cond_main, cond_worker; //cond_main used for wake up thread_main
int finish = 0;             //num of thread who finishs his job
int global_left;         //means 0 ~ global_left-1 need to do (row/quantity+1)個row
                         //global_left ~ row-1 need to do (row/quantity)個row
char **global_graph;  //live = 1, dead = 0, 2表示原本是活的但下次是死的, -1表示原本是死的等等變活的
/*
```

我用global memory存thread會用到的值
(a)finish: 是用來給main thread判斷所有的worker thread是否完成目前的工作。
(b)global_left: 每個worker thread會負責操作row/thread數的row，然後把餘數(global_left = row%thread數)給thread_id靠前的人分掉，因此id小於global_left的要多做一個row。
(c)global_graph: 存整張圖的二維陣列指標，因為是指標所以不會佔很大空間。

2.

```
pthread_mutex_lock(&mutex);   //確保main是第一個拿到mutex的
for(int i = 0; i < quantity; i++){
    pthread_create(&tid[i], NULL, Thread_funct, (void *)i);
}
int round = 0;
int task = 0;
while(round < epoch){       //有thread完成工作
    if(finish != quantity){           //有worker還沒完成
        pthread_cond_wait(&cond_main, &mutex);
        continue;
    }
    else if(finish == quantity){        //大家都完成了，叫醒大家準備做下一個round
        pthread_cond_broadcast(&cond_worker);
        //Refresh_graph();
        finish = 0;
        task ++;
        if(task == 1){
            //printf("read done\n");
        }
        else if(task == 2){
            task = 0;
            round ++;
            //printf("write done\n");
            //printf("round = %d\n", round);
            //Print_graph();
        }

    }
}
```

main_thread的工作在於判斷大家是否都完成目前的task，一開始main_thread睡覺，有worker完成task就叫main起床，main醒來判斷finish(已完成的worker數)，還沒達標就繼續睡，達標就叫醒所有worker做下一個task，做完2個task表示完成1個epoch。

3.

```
//printf("start = %d, end = %d\n", start, start + (global_row/global_quantity + k));
pthread_mutex_lock(&mutex);
pthread_mutex_unlock(&mutex);
while(1){
    Task_read(id, graph);
    pthread_mutex_lock(&mutex);
    finish++;
    pthread_cond_signal(&cond_main);
    pthread_cond_wait(&cond_worker, &mutex);
    pthread_mutex_unlock(&mutex);

    Task_write(id, graph);
    pthread_mutex_lock(&mutex);
    finish++;
    pthread_cond_signal(&cond_main);
    pthread_cond_wait(&cond_worker, &mutex);
    pthread_mutex_unlock(&mutex);
}
```

這是worker_thread要做的事，分為兩件事，先做Task_read，然後睡覺，等到main_thread確定所有人都讀完了後叫你起床，後做Task_write，然後睡覺，這樣算完成一個epoch，做Task_read和Task_write可以平行進行(不用等mutex)以節省時間。
Task_read: 讀圖，紀錄自己負責的row下次的樣子。
Task_write: 將紀錄更新在圖上。