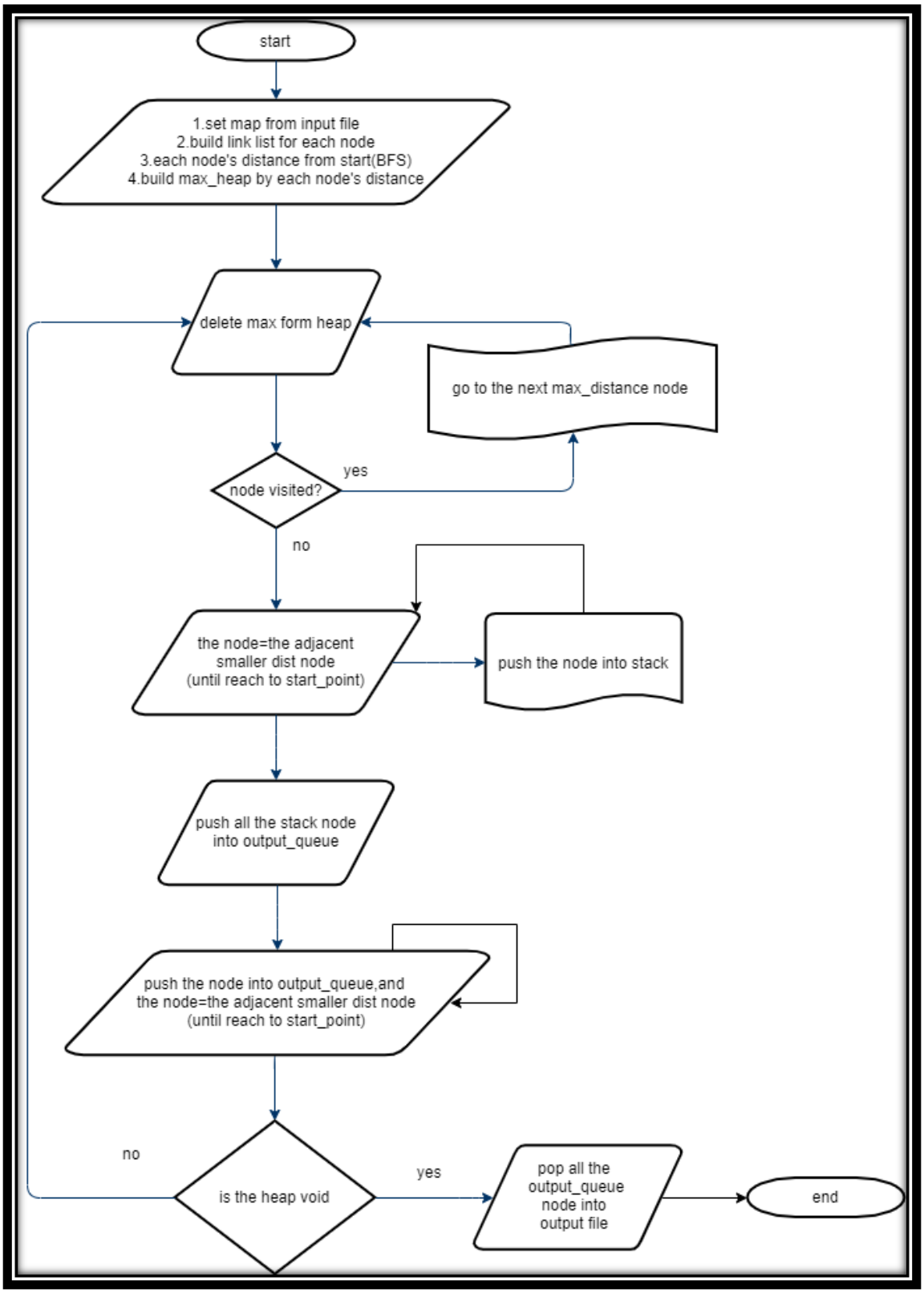
My flow chart:



1. data structure:

用一個二維陣列儲存每個 node，node 中的 dir[4]記錄該 node 四個方向的節點地址，如指向 NULL 代表是牆或無法走訪。在 node 中另設兩個變數 dist 與 visited 分別記錄該節點到電池的最短距離跟是否走訪過。
計算 dist 用從原點(電池)出發的 BFS。

```
void cal_dist(node *startnode)
{
    startnode->visit = true;
    QueueArrayCircular Q;
    Q.queue_Push(startnode);
    while (!Q.queue_IsEmpty())
    {
        node *u = Q.queue_getFront();
        Q.queue_Pop();
        for (int i = 0; i < 4; i++) //for all adjacent node from u
        {
            if ((u->dir[i] != NULL) && !(u->dir[i]->visit))
            {
                u->dir[i]->visit = true;
                Q.queue_Push(u->dir[i]); //mark the distence from startpoint
                u->dir[i]->dist = u->dist + 1;
            }
        }
    }
}
```

2. algorithm:

1. 先將所有 node 根據它的 dist 值存進 max heap 中。
2. 依序從 heap 中取出 dist 最大的 node 並計算該 node 至原點的來回路徑。
3. 計算去程與回程時用 go_smaller 去判斷一路往原點的最有效路徑。
4. 將經過的點依序存入 output_queue 中，最後 pop 出 queue 中所有的節點即是掃地機器人的走訪路徑。

```
while (now != startpoint) //go path
{
    now = go_smaller(now);
    s.stack_Push(now);
    step_count++;
    if (now->visit = true)
        now->visit = false;
}
now = des_point;
// cout stack path
while (!s.stack_IsEmpty())
{
    //cout << s.stack_Top()->r << " " << s.stack_Top()->r << endl;
    output_queue->queue_Push(s.stack_Top());
    s.stack_Pop();
}
```

↑去程時先從 now node 走回原點並 push 經過節點進 stack，最後再一次 pop stack 進 output_queue 中。

```
while (now != startpoint) //back path
{
    // push path into output queue
    output_queue->queue_Push(now);
    now = go_smaller(now);
    step_count++;
    if (now->visit = true)
        now->visit = false;
}
```

↑回程時一樣從 now node 走回原點並直接 push 經過節點進 output_queue 中。

```
node *go_smaller(node *now)
{
    node *tar = NULL, *small;
    for (int i = 0; i < 4; i++)
    {
        if (now->dir[i] != NULL && now->dir[i]->dist < now->dist)
        { //if there is a smaller node adjacent to "now node"
            if (now->dir[i]->visit == true)
                tar = now->dir[i]; //if the adjacent node hasn't been visited
            else
                small = now->dir[i]; // else place the node in "small"
        }
    }
    if (tar != NULL)
        return tar; //give the not visted node higher priority
    else
        return small; //all the node between "now" is visited, go whichever
}
```

↑go_smaller 判斷周圍的 node 那些是更接近原點的 node(dist 較小)，其中優先選擇還沒有走訪過的 node。

Git 紀錄