lab2 实验报告

1. 解题思路

1.1 如果start_time给定

那么按照dijkstra算法的思想。

以下是对算法进行了2处改变：

1. 其中weight值是变化的，需要根据当前时间`t`，先计算权重，再更新时间。

> 计算权重的时候，对每个潮汐变化计算出一个周期`cycle`，以判断是涨潮还是退潮。特殊处理了权重不变(`cycle = 0`)的情况。

```
//计算当前时刻边的权重
int curr_weight;
if(w.cycle) {
    curr_weight = curr.t % (2 * w.cycle);
    if(curr_weight > w.cycle) //涨潮
        curr_weight = (curr_weight - w.cycle) * w.step + w.lower;
    else  //退潮
        curr_weight = w.upper - w.step * curr_weight;
}
else //不变
    curr_weight = w.upper;
```

2. 没有用visited数组做标记，也没有用distance数组。

> 在dijkstra算法中，为了更快找到当前列表中的最小的distance，维护了一个小顶堆。

由于数据**只有一个起点，且无环**，所以开始时，`priority_queue`里只push了起点，之后（由于不要记录路径）每传递到一个结点，就向`priority_queue`里push进一个包含**结点编号**和**时间**的结构体，以代表一种可能的路径。

```
/**
 * @brief 用于在Dijkstra算法中维护一个小顶堆。
 *
 * @param point 节点的编号。
 * @param t 到该节点时计时器的读数。
 */
struct Dist_heap{
    int point;
    int t;
    Dist_heap(int p,int time = 0):point(p),t(time){}
    Dist_heap(){}
};
```

循环的停止条件是队列为空。当路径到达终点时，更新最小值。不再向队列中push。

> 剪枝：由于没有负权值，如果当前的时间已经大于目前的最小值，则不考虑这条路径。
>
> 没有用distance数组：因为贪心算法存在弊端，由于权重是动态变化的，当前的最小值也许会让下一条边weight变得很大。所以不能只记录最小值。
>
> 没有用visited数组进行剪枝：理由同上，遍历过的结点，即使新的t值比原来大，但可能会让下一条weight减小更多。

1.2 暴力求解最佳的start_time

从0开始依次用dijkstra算法算出最小值。

剪枝：start_time的上界动态设置为当前的最小值（初值为`INT_MAX`）

> 一些问题：
>
> 个人认为，上界还可以加一个：所有潮汐变化周期和K的最小公倍数。
>
> 如此避免了路径过长时，i需要循环多次。
>
> 但是每次加上最小公倍数，总是过不了testcase6。这里面可能还有欠考虑的地方，或者是最小公倍数代码有问题。最终放弃了这点。

2. 起点需要等待的情况

- 如果刚开始weight浮动的值较小，路径的后部分weight浮动交大，凑时间让后面路径的weight变小，回报更高。
- 如果刚开始weight浮动值很大，等待一会让潮汐降落，回报大于惩罚。