1. 写一个广度/宽度优先搜索算法，支持搜索策略定制（调整fringe集合中节点的次序）。

算法如下：

```
struct Compare{
    bool operator()(const Node* a, const Node* b) const{
        return a->cost > b->cost;
    }
};
/*
struct Node{
    int x,y;
    Node* prev:
    vector<Node*> neighbors;
    int cost;
    int depth;
};
*/
std::vector<Node*> bfs(Node* start, Node* goal) {
    std::priority_queue<Node*, std::vector<Node*>, CompareDepth> fringe;
    fringe.push(start);
    start->depth = 0;

    std::vector<Node*> path;
    while (!fringe.empty()) {
        Node* current = fringe.top();
        fringe.pop();

        if (current == goal) {
            // 回溯路径
            for (Node* step = goal; step != nullptr; step = step->prev) {
                path.push_back(step);
            }
            std::reverse(path.begin(), path.end());
            return path;
        }

        for (Node* neighbor : current->neighbors) {
            if (neighbor->depth == -1) { //避免重复（死循环）
                neighbor->depth = current->depth + 1;
                neighbor->prev = current;
                fringe.push(neighbor);
            }
        }
    }
    return path; // 如果没有找到路径，返回空路径
}
```

通过自定义 `Compare` 结构，可以调整 `fringe` 集合中节点的次序,从而实现搜索策略的定制。

2、搜索算法评估哪三个方面？各自的含义是什么？

- 完备性
    - 问题有解时，算法能否保证返回一个解？

    - 问题无解时，算法能否保证返回 failure?
- 最优性
    - 能否找到最优解?
    - 返回代价/路径耗散最小的路径?
- 复杂性:
    - 算法需求的时间和内存
    - 搜索树的大小：初态、后继函数和搜索策略决定，影响因素：分支
    - 因子（后继的最大个数），最浅目标状态的深度，路径的最大长度
    - 时间复杂度：访问过的节点数目
    - 空间复杂度：同时保存在内存中节点数目的最大值

3、广度优先搜索的时空复杂性1. 写一个广度优先搜索

- 时间复杂度:O(V + E)，其中V是图中顶点的数量，E是边的数量。
- 空间复杂度:O(V)