Breadth-First Search

Input: Tree 之 root r 和 goal node g

Output: The path from r to g or NiL if fail to find a solution

Steps: 1. 建構一Queue為Q，其中：Q = [r]

```
2. while Q ≠ ∅
       x = dequeue(Q)
       if x == g return path(r, g)
       else
          for each u ∈ T.adj[x]
               enqueue(Q, u)
3. return NiL
```

Depth-first search

Input: Tree 之 root r 和 goal node g

Output: The path from r to g or NiL if fail to find a solution

Steps: 1. 建構一Stack為S，其中：S = [r]

```
2. while S ≠ ∅
       x = pop(S)
       if x == g return path(r, g)
       else
          for each u ∈ T.adj[x]
               push(S, u)
3. return NiL
```

Depth-first search

Step 1. Form a 1-element stack consisting of the root node.

Step 2. Test to see if the top element in the stack is a goal node. If it is, then stop; otherwise, go to Step 3.

Step 3. Remove the top element from the stack and add its descendants, if any, to the top of the stack.

Step 4. If the stack is empty, then failure. Otherwise, go to Step 2.

Hill Climbing Algorithm

After reading the section about depth-first search, the reader may wonder about one problem: Among all the descendants, which node should be selected by us to expand? In this section, we shall introduce a scheme, called hill climbing. Hill climbing is a variant of depth-first search in which some greedy method is used to help us decide which direction to move in the search space. Usually, the greedy method uses some heuristic measure to order the choices. And, the better the heuristics, the better the hill climbing is.

Let us consider the 8-puzzle problem again. Assume that the greedy method uses the following simple evaluation function $f(n)$ to order the choices:

$$f(n) = w(n)$$

where $w(n)$ is the number of misplaced tiles in node n. Thus, if the starting node is positioned as shown in Figure 5–14, then $f(n)$ is equal to 3 because 1, 2 and 8 are misplaced.

差別：和 DFS 差別在每次選擇 evaluation value 最佳者做 DFS

Scheme of hill climbing

Step 1. Form a 1-element stack consisting of the root node.
Step 2. Test to see if the top element in the stack is a goal node. If it is, stop; otherwise, go to Step 3.
Step 3. Remove the top element from the stack and expand the element. Add the descendants of the element into the stack ordered by the evaluation function.
Step 4. If the list is empty, then failure. Otherwise, go to Step 2.

Best-First Search Algorithm

5-4 BEST-FIRST SEARCH STRATEGY

The best-first search strategy is a way of combining the advantages of both depth-first and breadth-first search into a single method. In best-first search, there is an evaluation function and we select the node with the least cost among all nodes which we have generated so far. It can be seen that the best-first search approach, unlike the hill climbing approach, has a global view.

Best-first search scheme

Step 1. Construct a heap by using the evaluation function. First, form a 1-element heap consisting of the root node.

Step 2. Test to see if the root element in the heap is a goal node. If it is, stop; otherwise, go to Step 3.

Step 3. Remove the root element from the heap and expand the element. Add the descendants of the element into the heap.

Step 4. If the heap is empty, then failure. Otherwise, go to Step 2.

Input: Tree 之 root r 和 goal node g 和 evaluation function

Output: The path from r to g or Nil if fail to find a solution

steps:
1. 建構一 heap H, 其中: $H=[r]$
2. 從 H 取出 first element s, 展開 s 為 root 之 subtree
   若任何一个 descedants 為 g, return path(r, g)
   否則加入 descedant 至 H
3. 依 evaluation function 做 Heap sort
4. 若 $H=\phi$, return Nil
   否則, 回至 step 2.