搜索方法对比

概念

- g(n)：路径耗散，表示从搜索节点到节点n产生的路径代价之和
- h(n)：从节点n到搜索的目标节点最小代价路径的**估计值**，h源于heuristic（启发式的）

1. BFS(breadth-first search)广度优先搜索

即优先扩展浅层节点

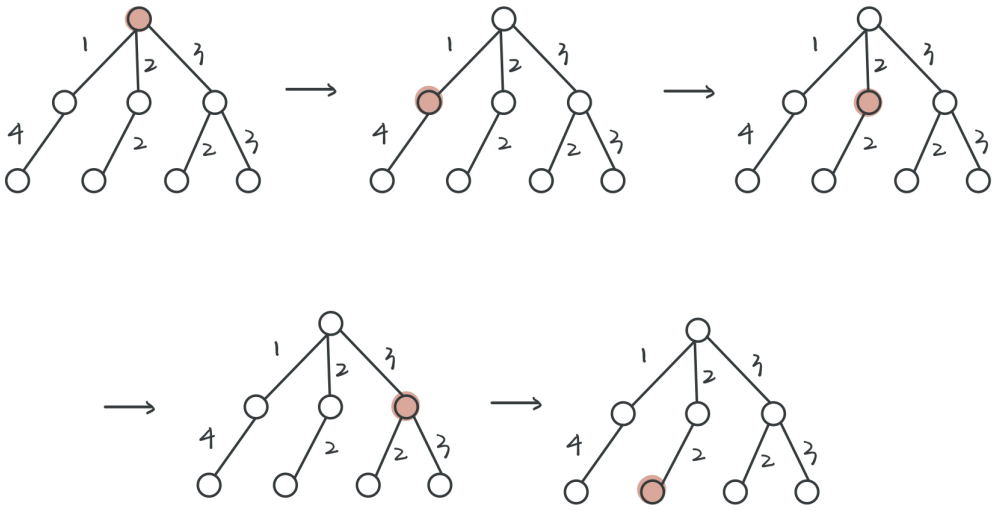
```
BFS(){       //摘自chp7 ppt
    初始化队列
    while(队列不为空且未找到目标结点){
        取队首结点扩展，并将扩展出的结点放入队尾
        必要时要记住每个结点的父结点
    }
}
```

2. UCS(uniform-cost search)一致代价搜索

一致代价搜索每次选择扩展的是g(n)最小的节点，当任意搜索树上任意相连节点之间路径代价相同时，一致代价搜索和BFS较为类似，可以认为BFS是UCS的一个特例。

```
UCS(){
    priorqueue.push(start, 0)  //利用优先队列，权值为节点g(n)
    while(!priorqueue.isEmpty()){
        node, cost = priorqueue.pop()  //cost是g(node)
        if(node = goal)
            return node
        for i in node.child:
            priorqueue.push(i, cost + c(node, i)) //c是从node到i的代价
    }
    return fail
}
```

搜索顺序的一个示意图为



3. DFS(depth-first search)宽度优先搜索

即总是在搜索树中扩展最深的边缘节点

```
DFS(){       //摘自chp7 ppt
    初始化栈
    while(栈不为空 且 未找到目标结点){
        取栈顶结点扩展，扩展出的结点放回栈顶
    }
    ……
}
```

4. IDDFS(iterative deepening DFS)迭代加深DFS

在DFS的基础上，从深度为1的限制逐渐依次增大直至找到目标。

```
IDDFS(){
    for depth = 1 to inf do
        result = DFS_limited(depth)
        if(result != fail)
            return result
}
```

IDDFS的时间复杂度和BFS量级相同，但由于存在很多重复搜索会有更大的常数因子，在数独问题上由于前80层都搜不到并不能提高效率。

5. (Greedy best-first search)贪婪最佳优先搜索

贪婪最佳优先搜索试图扩展离目标节点最近的节点，因此只使用启发值h(n)。由于该算法是贪心算法，因此只搜索**局部**启发值最优的节点。贪心算法的目的不是为了找到全部解，也当然找不出最优解，而只是找出一种可行解。

```
GBFS(){
    n = start
    while(s != null){
        t = s的子节点中启发值最小的
        if(t == goal)
            return t
        s = t
    }
    return fail
}
```

6. A*搜索

A*搜索的目的是找到最小的预估总代价，因此评估函数由两部分组成，即f(n) = g(n) + h(n)。

```
Astar(){
    priorqueue.push(start, 0)
    cost[start] = 0           //cost[node]存放g(node)
    while(!priorqueue.isEmpty()){
        node = priorqueue.pop()
        if(node == goal)
            return node
        for i, c in node.child:
            cost[i] = cost[node] + c
            priorqueue.push(i,cost[i]+h(i)) //权值为f(n) = g(n) + h(n)
    }
    return fail
}
```

！注意

> 在实验检查中发现很多同学在A*搜索中实现的是群里有同学提到的每次找可填数字最少的子节点，可以认为这种情况下该语义表示的是启发值，但由于并没有用到g(n)(在数独中是搜索深度)，因此**并不是A*搜索**而是最佳优先搜索。同时由于仅在当前节点的子节点中搜索，只找到了局部最优值，因此是贪婪最佳优先搜索。

另外由于数独上寻找的是可行解而非最优解，A*算法并不适用于这个问题，但最好还是不要毫无根据地设计启发式函数，只有在启发函数满足可采纳性和一致性的情况下才可以保证算法的最优性。（非本课程范围，在此不做过多介绍）

7. IDA*

将IDDFS的深度限制改为f(n)限制

说明：为了完成数独扩展部分的高级搜索部分，你可以有更多的选择范围，包括但不限于上述2,4-7，但请尽可能保证算法的正确性。