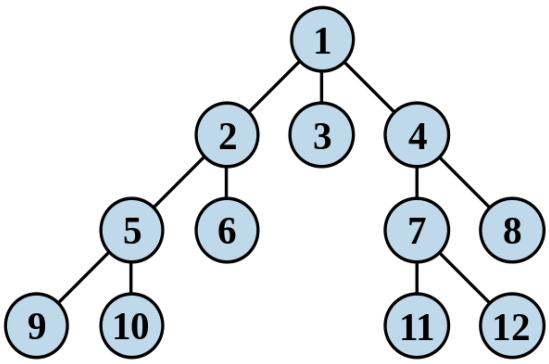
维基百科

广度优先搜索

维基百科，自由的百科全书

广度优先搜索算法（英语：Breadth-First Search，缩写为BFS），又译作宽度优先搜索，或横向优先搜索，是一种图形搜索算法。简单的说，BFS是从根节点开始，沿着树的宽度遍历树的节点。如果所有节点均被访问，则算法中止。广度优先搜索的实现一般采用open-closed表。



节点进行广度优先搜索的顺序

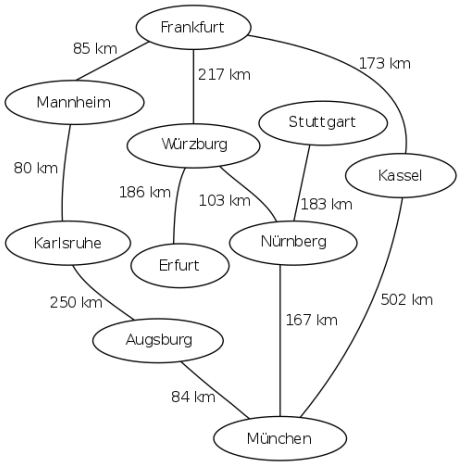
广度优先搜索	
概况	
类别	搜索算法
数据结构	图
复杂度	
平均时间复杂度	$O(\|V\|+\|E\|)=O(b^d)$
最坏时间复杂度	$O(\|V\|+\|E\|)=O(b^d)$
空间复杂度	$O(\|V\|)=O(b^d)$
最佳解	是
完全性	是
相关变量的定义	

目录

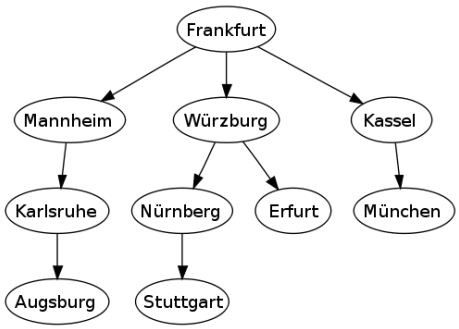
作法

BFS是一种暴力搜索算法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能地址，彻底地搜索整张图，直到找到结果为止。BFS并不使用经验法则算法。

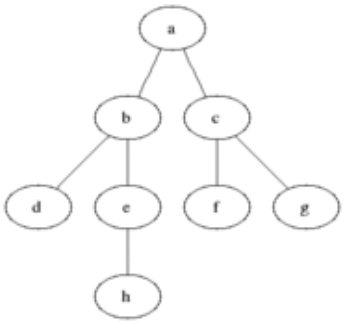
从算法的观点，所有因为展开节点而得到的子节点都会被加进一个先进先出的队列中。一般的实现里，其邻居节点尚未被检验过的节点会被放置在一个被称为 *open* 的容器中（例如队列或是链表），而被检验过的节点则被放置在被称为 *closed* 的容器中。（open-closed表）



以德国城市为示例的地图。城市间有数条道路相连接。



从法兰克福开始执行广度优先搜索算法，所产生的广度优先搜索算法树。



广度优先搜索算法的动画示例

实现方法

1. 首先将根节点放入队列中。
2. 从队列中取出第一个节点，并检验它是否为目标。
   - 如果找到目标，则结束搜索并回传结果。
   - 否则将它所有尚未检验过的直接子节点加入队列中。
3. 若队列为空，表示整张图都检查过了——亦即图中没有欲搜索的目标。结束搜索并回传“找不到目标”。
4. 重复步骤2。

```
s为初始点
R := {s}, Q := {s}, T = ∅
while Q ≠ ∅
    从Q中选一点 v /* 若改选最后插入进Q的点，则为深度遍历,可以说后进先出。*/
    if ∃w ∈ N(v) \ R then    /* N(v):v的邻接点 */
        Q := Q ∪ {w}
        R := R ∪ {w}
        T := T ∪ {vw}
    else Q := Q \ {w}
return H=(R,T)
```

C 的实例

```
/*
    ADDQ (Q, p) - p PUSH 入 Q
    DELQ (Q) - POP Q 并返回 Q 顶
    FIRSTADJ (G,v) - v 的第一个邻接点，找不到则返回 -1
    NEXTADJ (G,v) - v 的下一个邻接点，找不到则返回 -1
    VISIT (v) - 访问 v
    visited [] - 是否已访问
*/

// 广度优先搜索算法
void BFS(VLink G[], int v) {
    int w;
    VISIT(v); // 访问 v 并入队
    visited[v] = 1;
    ADDQ(Q, v);
    // 对队列 Q 的各元素
    while (!EMPTYQ(Q)) {
        v = DELQ(Q);
        w = FIRSTADJ(G, v);
        do {
            // 进行访问和入队
            if (visited[w] == 0) {
                VISIT(w);
                ADDQ(Q, w);
                visited[w] = 1;
            }
        } while ((w = NEXTADJ(G, v)) != -1);
    }
}

// 对图G=(V,E)进行广度优先搜索的主算法
void TRAVEL_BFS(VLink G[], bool visited[], int n) {
    // 清零标记数组
    for (int i = 0; i < n; ++i)
        visited[i] = 0;
    for (int i = 0; i < n; ++i)
        if (visited[i] == 0)
            BFS(G, i);
}
```

C++ 的实现

(这个例子仅针对Binary Tree)
定义一个结构体来表达一个节点的结构：

```
struct node {
    int self;     //数据
    node *left;   //左节点
    node *right;  //右节点
};
```

那么，我们在搜索一个树的时候，从一个节点开始，能首先获取的是它的两个子节点。例如：

```
   A
B     C
```

A是第一个访问的，然后顺序是B和C；然后再是B的子节点，C的子节点。那么我们怎么来保证这个顺序呢？

这里就应该用queue数据结构，因为queue采用先进先出(first-in-first-out)的顺序。

使用C++的STL库，下面的程序能帮助理解：

```
 std::queue<node *> visited, unvisited;
node nodes[9];
node *current;

unvisited.push(&nodes[0]); // 先把root放入unvisited queue

while (!unvisited.empty()) { // 只有unvisited不空
    current = (unvisited.front()); // 目前應該檢驗的
    if (current->left != NULL)
        unvisited.push(current->left); // 把左邊放入queue中
    if (current->right != NULL) // 右邊壓入。因為QUEUE是一個先進先出的結構构，所以即使後面再壓其他东西，依然會先訪問這個。
        unvisited.push(current->right);
    visited.push(current);
    cout << current->self << endl;
    unvisited.pop();
}
```

特性

空间复杂度

因为所有节点都必须被存储，因此BFS的空间复杂度为$O(|V|+|E|)$，其中$|V|$是节点的数目，而$|E|$是图中边的数目。注：另一种说法称BFS的空间复杂度为$O(B^M)$，其中B是最大分支系数，而M是树的最长路径长度。由于对空间的大量需求，因此BFS并不适合解非常大的问题，对于类似的问题，应用IDDFS以达节省空间的效果。

时间复杂度

最差情形下，BFS必须查找所有到可能节点的所有路径，因此其时间复杂度为$O(|V|+|E|)$，其中$|V|$是节点的数目，而$|E|$是图中边的数目。

完全性

广度优先搜索算法具有完全性。这意指无论图形的种类如何，只要目标存在，则BFS一定会找到。然而，若目标不存在，且图为无限大，则BFS将不收敛（不会结束）。

最佳解

若所有边的长度相等，广度优先搜索算法是最佳解——亦即它找到的第一个解，距离根节点的边数目一定最少；但对一般的图来说，BFS并不一定回传最佳解。这是因为当图形为加权图（亦即各边长度不同）时，BFS仍然回传从根节点开始，经过边数目最少的解；而这个解距离根节点的距离不一定最短。这个问题可以使用考虑各边权值，BFS的改良算法成本一致搜索法来解决。然而，若非加权图形，则所有边的长度相等，BFS就能找到最近的最佳解。

广度优先搜索算法的应用

广度优先搜索算法能用来解决图论中的许多问题，例如：

- 查找图中所有连接组件（Connected Component）。一个连接组件是图中的最大相连子图。
- 查找连接组件中的所有节点。
- 查找非加权图中任两点的最短路径。
- 测试一图是否为二分图。
- （Reverse）Cuthill–McKee算法

查找连接组件

由起点开始，执行广度优先搜索算法后所经过的所有节点，即为包含起点的一个连接组件。

测试是否二分图

BFS可以用以测试二分图。从任一节点开始搜索，并在搜索过程中给节点不同的标签。例如，给开始点标签0，开始点的所有邻居标签1，开始点所有邻居的邻居标签0……以此类推。若在搜索过程中，任一节点有跟其相同标签的邻居，则此图就不是二分图。若搜索结束时这种情形未发生，则此图为一二分图。

应用于电脑游戏中平面网格

BFS可用来解决电脑游戏（例如即时策略游戏）中找寻路径的问题。在这个应用中，使用平面网格来代替图形，而一个格子即是图中的一个节点。所有节点都与它的邻居（上、下、左、右、左上、右上、左下、右下）相接。

值得一提的是，当这样使用BFS算法时，首先要先检验上、下、左、右的邻居节点，再检验左上、右上、左下、右下的邻居节点。这是因为BFS趋向于先查找斜向邻居节点，而不是四方的邻居节点，因此找到的路径将不正确。BFS应该先查找四方邻居节点，接着才查找斜向邻居节点1。

参见

- 先验算法
- 深度优先搜索

参考资料

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein], *Introduction to Algorithms*, Second Edition. MIT Press and McGraw–Hill, 2001. ISBN 0–262–03293–7. Section 22.2: Breadth–first search, pp. 531—539.

外部链接

- （英文）数据结构与算法字典：广度优先搜索 (http://www.nist.gov/dads/HTML/breadthfirst.html)（页面存档备份 (https://web.archive.org/web/20070401190651/http://www.nist.gov/dads/HTML/breadthfirst.html)，存于互联网档案馆）
- （英文）C++ Boost Graph库：广度优先搜索 (http://www.boost.org/libs/graph/doc/breadth_first_search.html)（页面存档备份 (https://web.archive.org/web/20070329004953/http://www.boost.org/libs/graph/doc/breadth_first_search.html)，存于互联网档案馆）
- （英文）深度与广度优先搜索：解释与源代码 (http://www.kirupa.com/developer/actionscript/depth_breadth_search.htm)（页面存档备份 (https://web.archive.org/web/20070410024509/http://www.kirupa.com/developer/actionscript/depth_breadth_search.htm)，存于互联网档案馆）
- （英文）BFS 动画说明 (http://www.cs.duke.edu/csed/jawaa/BFSanim.html)（页面存档备份 (https://web.archive.org/web/20070403052434/http://www.cs.duke.edu/csed/jawaa/BFSanim.html)，存于互联网档案馆）