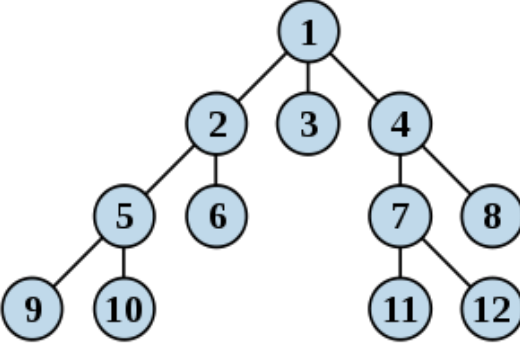


广度优先搜索

维基百科，自由的百科全书

广度优先搜索算法（英語：Breadth-First Search，縮寫為BFS），又譯作**寬度優先搜索**，或**橫向優先搜索**，是一種圖形搜索演算法。簡單的說，BFS是從根節點開始，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。

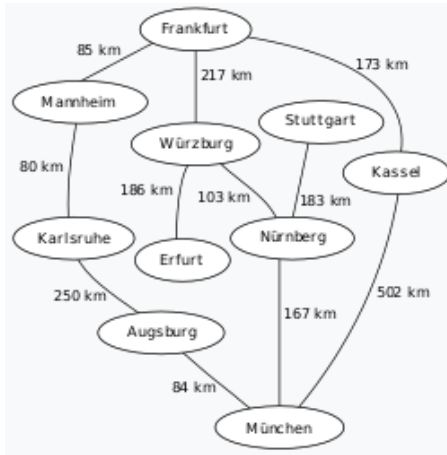
目录
作法
實作方法
C 的实例
C++ 的實作
特性
空間複雜度
時間複雜度
完全性
最佳解
應用
尋找連接元件
測試是否二分圖
應用於電腦遊戲中平面網格
參見
參考資料
外部連結

广度优先搜索	
<div><p>節點進行广度优先搜索的順序</p></div>	
概况	
類別	搜索演算法
資料結構	圖
复杂度	
平均時間複雜度	$O(V + E) = O(b^d)$
最坏时间复杂度	$O(V + E) = O(b^d)$
空間複雜度	$O(V) = O(b^d)$
最佳解	是
完全性	是
相关变量的定义	

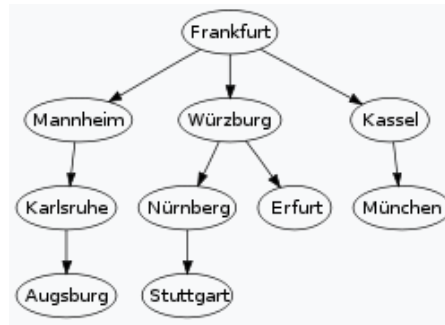
作法

BFS是一種暴力搜索算法，目的是系統地展開並檢查圖中的所有節點，以找尋結果。換句話說，它並不考慮結果的可能位址，徹底地搜索整張圖，直到找到結果為止。BFS並不使用經驗法則演算法。

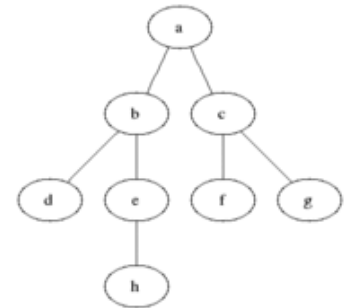
從演算法的觀點，所有因為展開節點而得到的子節點都會被加進一個先進先出的队列中。一般的實作裡，其鄰居節點尚未被檢驗過的節點會被放置在一個被稱為 *open* 的容器中（例如佇列或是链表），而被檢驗過的節點則被放置在被稱為 *closed* 的容器中。（open-closed表）



以德國城市為範例的地圖。城市間有數條道路相連接。



從法蘭克福開始執行廣度優先搜索算法，所產生的廣度優先搜索算法樹。



廣度優先搜索算法的動畫範例

實作方法

1. 首先將根節點放入队列中。
2. 從队列中取出第一個節點，並檢驗它是否為目標。
 - 如果找到目標，則結束搜尋並回傳結果。
 - 否則將它所有尚未檢驗過的直接子節點加入队列中。
3. 若队列为空，表示整張圖都檢查過了——亦即圖中沒有欲搜尋的目標。結束搜尋並回傳「找不到目標」。
4. 重複步驟2。

```

s为初始点
R := {s}, Q := {s}, T = ∅
while Q ≠ ∅
  從Q中選一點 v /* 若改選最後插入進Q的點，則為深度遍歷, 可以说後進先出。 */
  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

```

1  /*
2   ADDQ (Q, p) - p PUSH 入 Q
3   DELQ (Q) - POP Q 并返回 Q 顶
4   FIRSTADJ (G,v) - v 的第一个邻接点, 找不到则返回 -1
5   NEXTADJ (G,v) - v 的下一个邻接点, 找不到则返回 -1
6   VISIT (v) - 访问 v
7   visited [] - 是否已访问
8  */
9
10 // 广度优先搜索算法
11 void BFS(VLink G[], int v) {
12     int w;
13     VISIT(v); // 访问 v 并入队
14     visited[v] = 1;
15     ADDQ(Q, v);
16     // 对队列 Q 的各元素
17     while (!EMPTYQ(Q)) {
18         v = DELQ(Q);
19         w = FIRSTADJ(G, v);
20         do {
21             // 进行访问和入队
22             if (visited[w] == 0) {
23                 VISIT(w);
24                 ADDQ(Q, w);
25                 visited[w] = 1;
26             }
27         } while ((w = NEXTADJ(G, v)) != -1);
28     }
29 }
30
31 // 对图G=(V,E)进行广度优先搜索的主算法
32 void TRAVEL_BFS(VLink G[], bool visited[], int n) {
33     // 清零标记数组
34     for (int i = 0; i < n; ++i)
35         visited[i] = 0;
36     for (int i = 0; i < n; ++i)
37         if (visited[i] == 0)
38             BFS(G, i);
39 }

```

C++ 的實作

(這個例子僅針對Binary Tree)

定义一个结构体来表达一个節點的结构：

```

1  struct node {
2      int self;      //数据
3      node *left;    //左节点
4      node *right;   //右节点
5  };

```

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資料結構，因为queue採用先进先出(first-in-first-out)的顺序。

使用C++的STL函式庫，下面的程序能帮助理解：

```

1  std::queue<node *> visited, unvisited;
2  node nodes[9];
3  node *current;
4
5  unvisited.push(&nodes[0]); // 先把root放入unvisited queue
6
7  while (!unvisited.empty()) { // 只有unvisited不空
8      current = (unvisited.front()); // 目前應該檢驗的
9      if (current->left != NULL)
10         unvisited.push(current->left); // 把左邊放入queue中
11     if (current->right != NULL) // 右邊壓入。因為QUEUE是一個先進先出的結構，所以即使後面再壓其他東西，依然會先訪問
        這個。
12         unvisited.push(current->right);
13     visited.push(current);
14     cout << current->self << endl;
15     unvisited.pop();
16 }

```

特性

空間複雜度

因為所有節點都必須被儲存，因此BFS的空間複雜度為 $O(|V| + |E|)$ ，其中 $|V|$ 是節點的數目，而 $|E|$ 是圖中邊的數目。註：另一種說法稱BFS的空間複雜度為 $O(B^M)$ ，其中B是最大分支係數，而M是樹的最長路徑長度。由於對空間的大量需求，因此BFS並不適合解非常大的問題，對於類似的問題，應用IDDFS以達節省空間的效果。

時間複雜度

最差情形下，BFS必須尋找所有到可能節點的所有路徑，因此其時間複雜度為 $O(|V| + |E|)$ ，其中 $|V|$ 是節點的數目，而 $|E|$ 是圖中邊的數目。

完全性

廣度優先搜索演算法具有完全性。這意指無論圖形的種類如何，只要目標存在，則BFS一定會找到。然而，若目標不存在，且圖為無限大，則BFS將不收斂（不會結束）。

最佳解

若所有邊的長度相等，廣度優先搜索演算法是最佳解——亦即它找到的第一個解，距離根節點的邊數目一定最少；但對一般的圖來說，BFS並不一定回傳最佳解。這是因為當圖形為加權圖（亦即各邊長度不同）時，BFS仍然回傳從根節點開始，經過邊數目最少的解；而這個解距離根節點的距離不一定最短。這個問題可以使用考慮各邊權值，BFS的改良演算法成本一致搜尋法來解決。然而，若非加權圖形，則所有邊的長度相等，BFS就能找到最近的最佳解。

應用

廣度優先搜索演算法能用來解決圖論中的許多問題，例如：

- 尋找圖中所有連接元件（Connected Component）。一個連接元件是圖中的最大相連子圖。
- 尋找連接元件中的所有節點。
- 尋找非加權圖中任兩點的最短路徑。
- 測試一圖是否為二分圖。
- （Reverse）Cuthill–McKee演算法

尋找連接元件

由起點開始，執行廣度優先搜索演算法後所經過的所有節點，即為包含起點的一個連接元件。

測試是否二分圖

BFS可以用以測試二分圖。從任一節點開始搜尋，並在搜尋過程中給節點不同的標籤。例如，給開始點標籤0，開始點的所有鄰居標籤1，開始點所有鄰居的鄰居標籤0.....以此類推。若在搜尋過程中，任一節點有跟其相同標籤的鄰居，則此圖就不是二分圖。若搜尋結束時這種情形未發生，則此圖為一二分圖。

應用於電腦遊戲中平面網格

BFS可用來解決電腦遊戲（例如即時策略遊戲）中找尋路徑的問題。在這個應用中，使用平面網格來代替圖形，而一個格子即是圖中的一個節點。所有節點都與它的鄰居（上、下、左、右、左上、右上、左下、右下）相接。

值得一提的是，當這樣使用BFS演算法時，首先要先檢驗上、下、左、右的鄰居節點，再檢驗左上、右上、左下、右下的鄰居節點。這是因為BFS趨向於先尋找斜向鄰居節點，而不是四方的鄰居節點，因此找到的路徑將不正確。BFS應該先尋找四方鄰居節點，接著才尋找斜向鄰居節點1。

參見

- 先验算法
- 深度優先搜索

參考資料

- 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.

外部連結

- （英文） 資料結構與演算法字典：廣度優先搜索 (<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函式庫：廣度優先搜索 (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） 深度與廣度優先搜索：解釋與原始碼 (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 動畫說明 (<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）

本页面最后修订于2022年4月24日 (星期日) 14:58。

本站的全部文字在知识共享 署名-相同方式共享 3.0协议之条款下提供，附加条款亦可能应用。（请参阅使用条款）

Wikipedia®和维基百科标志是维基媒体基金会的注册商标；维基™是维基媒体基金会的商标。

维基媒体基金会是按美国国内稅收法501(c)(3)登记的非营利慈善机构。