Wordnet report   B06505004 莊博翰

****

* Describe concisely the data structure(s) you used to store the
* information in synsets.txt. Why did you make this choice?

```
    // index to noun
    private HashMap<Integer, String> synset;
    // noun to index
    private HashMap<String, Bag<Integer>> index;
```

兩個 hashmap 分別是由 string 查 id，和由 id 查 string
這裡可以注意到一點 ，由 string 查 id 可能會查詢到不只一個 id，這是因為我們需要的 synset 如下:

synset

```
% more synsets.txt
 ⋮
34,AIDS acquired_immune_deficiency_syndrome,a serious (often fatal) disease of the
35,ALGOL,a programming language used to express computer programs as algorithms
36,AND_circuit AND_gate,a circuit in a computer that fires only when all of its i
37,APC,a drug combination found in some over-the-counter headache remedies
38,ASCII_character,any member of the standard code for representing characters by
39,ASCII_character_set,(computer science) 128 characters that make up the ASCII c
40,ASCII_text_file,a text file that contains only ASCII characters without specia
41,ASL American_sign_language,the sign language used in the United States
42,AWOL,one who is away or absent without leave
 ⋮
```

gloss

可以看到一個 id 雖然可以查到一組 string 如 36➔ { AND_circuit, AND_gate }
然而實際用到的時機只有最後輸出結果，也就是反正最後都要接起來輸出
不如一開始就存成接起來的模式，所以才會是:

HashMap<Integer, String> synset

另一方面，一個 string 也有機會對應到多個 id，這些 id 是以後做 shortest comment ancestor 會使用到的，所以用一個 iterable 的 Bag(linked list) 進行儲存，以方便將來使用

```
public int ancestorSubset(Iterable<Integer> subsetA, Iterable<Integer> subsetB)
public int lengthSubset(Iterable<Integer> subsetA, Iterable<Integer> subsetB)
```

所以才會是以下型態

private HashMap<String, Bag<Integer>> index;

****
*   Describe concisely the data structure(s) you used to store the
*   information in hypernyms.txt. Why did you make this choice?
****

這就滿直觀的使用 Digraph ，畢竟這個檔案本身就是在描述一個有向圖

```
Digraph g = new Digraph(vnum);
```

****
*   Describe concisely the algorithm you use in the constructor of
*   ShortestCommonAncestor to check if the digraph is a rooted DAG.
*   What is the order of growth of the worst-case running times of
*   your algorithm? Express your answer as a function of the
*   number of vertices V and the number of edges E in the digraph.
*   (Do not use other parameters.) Use Big Theta notation to simplify
*   your answer.

****

Description:

```
private void valid(Digraph G) {
    if (G == null) {
        throw new IllegalArgumentException();
    }
    int totalv = G.V();
    int rootcount = 0;
    for (int v = 0; v < totalv; v++) {
        if (G.outdegree(v) == 0) {
            rootcount++;
        }
    }
    if (rootcount > 1) {
        throw new IllegalArgumentException();
    }
    DirectedCycle dc = new DirectedCycle(G);
    if (dc.hasCycle()) {
        throw new IllegalArgumentException();
    }
```

總共分三步，第一步確認 G 是不是 null　$\Theta(1)$

第二步確認是否只有一個 root(沒有 fanout 的 vertix)　$\Theta(V)$

第三步確認是否有 circle，這裡使用 [DirectedCycle.java]

takes $\Theta(V + E)$ time in the worst case

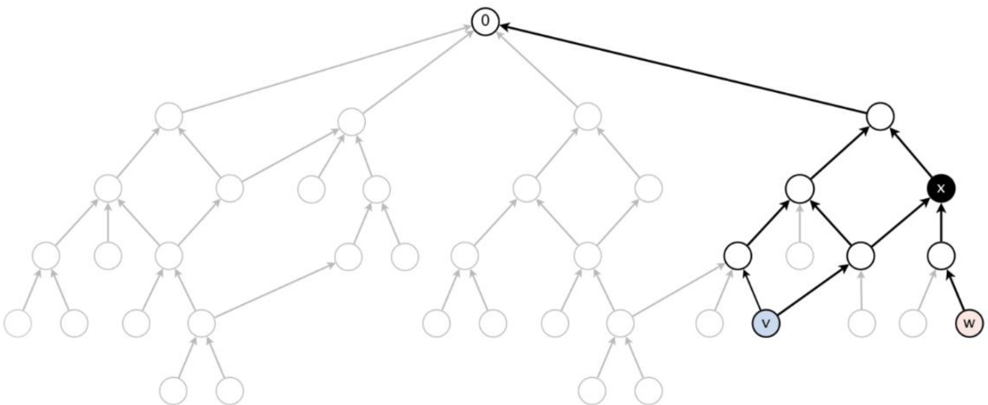
Order of growth of running time: $\Theta(V+E)$

****
* Describe concisely your algorithm to compute the shortest common ancestor
* in ShortestCommonAncestor. For each method, give the order of growth of
* the best- and worst-case running times. Express your answers as functions
* of the number of vertices V and the number of edges E in the digraph.
* (Do not use other parameters.) Use Big Theta notation to simplify your
* answers.
*
* If you use hashing, assume the uniform hashing assumption so that put()
* and get() take constant time per operation.
*
* Be careful! If you use a BreadthFirstDirectedPaths object, don't forget
* to count the time needed to initialize the marked[], edgeTo[], and
* distTo[] arrays.

****

Description:

為了達成 *Additional performance requirements* 為了只 traversal *reachable* vertices and edge ，必須 implement 自己的 BFS



先描述需要的資料結構

```
private int[] mindisA;
// same as mindisA
private int[] mindisB;
// markA is mark when se
private int[] markA;
// same as markA
private int[] markB;
```

記錄 set A 到該點的最短路徑

紀錄 set B 到該點的最短路徑

紀錄該點是否被 set A 走過

紀錄該點是否被 set B 走過

這裡注意到，除了 initilize　ShortestCommonAncestor 需要 O(V)的時間
其他操作都只能使用 O(Vr+Er) ,Vr Er➔ *reachable* vertices and edge
所以，如果我們在重複使用 SCA(ShortestCommonAncestor 簡寫)
時，比如用很多次 int ancestor(int v, int w)，我們會需要 unmark 之前
mark 過的 markA ,markB ,這可能會花 O(V)，所以我改變了 mark 的方法
讓 mark 是 int 而非 boolean，並加上新的常數，

```
private int markconstA;
// as markconstA
private int markconstB;
```

只有在 mark==markconst 時才算是 mark
而把 markA[i] 設為 markconstA 代表
它被 mark 過，如此一來我們 unmark 時
只要 markconst++ 就可確保全部的 mark
都小於 markconst(因為 markconst 只會不斷往上加)，實現 O(1)的 unmark
(當然執行非常非常多次後，可能要 unmark O(V)一次，避免 markconst overflow)
接下來討論 lengthSubset()　，ancestorSubset()
同時，因為 length()和 ancestor()可以被視為只有一個元素的 subset 所以跳過
又其實 lengthSubset() ,ancestorSubset()的結果會同時產生，所以便僅已
lengthSubset() 作為說明

```
public int lengthSubset(Iterable<Integer> subsetA, Iterable<Integer> subsetB) {
    initbfs();
    for (int v : subsetA) {
        bfsA(v, 0);
    }
    for (int v : subsetB) {
        bfsB(v, 0);
    }
    return shortestlength;
}
```

基本結構極度簡單，就是先用 setA 掃一遍 BFS，找到 setA reachable 的 vertex
的最短距離，再用 setB 掃一次 BFS，找出 setA,setB 都 reachable 的 vertex
並挑出 mindisA+mindisB 最小的 vertex(ShortestCommenAncestor)，
同時 mindisA+mindisB=shortestlenght;

接下來介紹 bfsA 的內容

```java
    private void bfsA(int nowv, int nowdis) {
        if (ismarkA(nowv)) {
            // if this vertex already vistied an
            // smaller distance , no need to con
            if (mindisA[nowv] < nowdis) {
                return;
            }
            else {
                mindisA[nowv] = nowdis;
            }
        }
        else {
            mindisA[nowv] = nowdis;
            markA(nowv);
        }
        for (int v : G.adj(nowv)) {
            bfsA(v, nowdis + 1);
        }
```

簡單來說，每走到一個 vertex ，它會先檢查是否走過了(ismarkA)，如果沒有現在走的距離(nowdis)就是它的最短距離，並把它標為走過的(markA)。如果走過就檢查他之前的最短距離和現在走到的距離誰小，如果現在的比較大就沒有必要往下走了，因為這個點有更小的最短路徑代表了，這個點以後的路徑都會比現在往下走所走過的距離小。

接下來簡介 bfsB 的內容

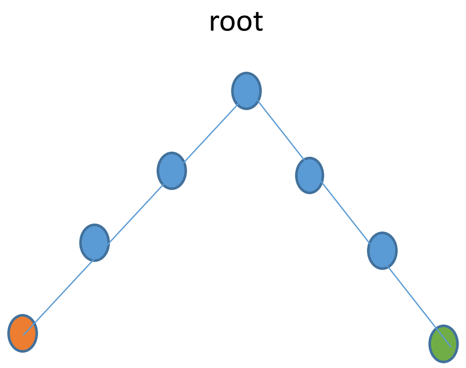
```java
private void bfsB(int nowv, int nowdis) {
    if (ismarkB(nowv)) {
        if (mindisB[nowv] < nowdis) {
            return;
        }
        else {
            mindisB[nowv] = nowdis;
        }
    }
    else {
        mindisB[nowv] = nowdis;
        markB(nowv);

    }
    if (ismarkA(nowv)) {
        if (mindisB[nowv] + mindisA[nowv] < shortestlength) {
            shortestlength = mindisB[nowv] + mindisA[nowv];
            shortestancestor = nowv;
        }
        else {
            return;
        }
    }
    for (int v : G.adj(nowv)) {
        bfsB(v, nowdis + 1);
    }
```

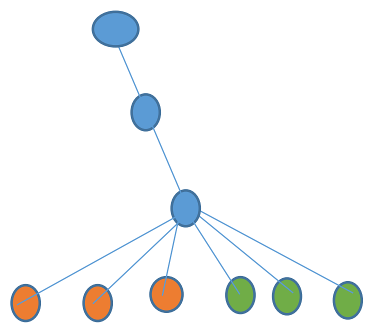
跟 BFSA 很像，也要 handle setB 到每個 reachable vertex 的最短距離
不過多了要計算 mindisA+mindisB 跟現在的 shortestlength 比較
如果比較大就 cutoff。
因此，雖然看起來 subset 有 n 個 element 就要 n*O(Vr+Er)，但是其實越靠後的元素越容易被提早 return ，

	running time	
method	best case	worst case
length() :	Θ(1) 就在隔壁	Θ(V+E) V 字形兩端
ancestor() :	Θ(1)	Θ(V+E)
lengthSubset(): n element set	Θ(n)	Θ(n(V+E))
ancestorSubset():	Θ(n)	Θ(n(V+E))

V 字形兩端示意圖



best case lengthSubset():



worst case lengthSubset()

setA setB 都由下往上開始找最短路徑，找到的永遠被下一個推翻

time=O((V+E)+(v-2+E-2)+...+(v-2n+E-2n)) when V,E>>n time: Θ(n(V+E))