加分題目錄

此文件有4個部分如下:

1. 題目詳述 : 描述要解什麼，題目的需要用到的定義(ex: follow set定義)、計算follow set的pseudocode
2. 題目 : 簡單的說題目要解甚麼，給予input和output的範例，並提供一些說明
3. 題目的公開測資的input和output，避免有些人因為奇奇怪怪的原因，無法正常看到公開測資，所以附在題目上
4. 一些可以參考的程式碼，像是input處理那部分的程式，避免有人卡到陰，浪費一堆時間，有現成的程式碼可以直接用或參考

題目詳述

給予文法定義與各個**nonTerminal**的**first set**，求所有**nonTerminal**的**follow set**

下面為之後描述的symbol定義

1. 大寫字母代表nonTerminal symbol，小寫字母代表Terminal symbol
2. First(A)代表A的first set
3. Follow(A)代表A的follow set

first set的定義

就是對於一個symbol而言，其使用production rule代換後的第一個terminal symbol(可能是λ)，ex: 下面有三條production rule

```
A -> BC
B -> b
C -> c
```

A → BC 經由代換掉B會得 A → bC，因此b會在A的first set中

下面描述完整的定義:
對於`terminal a`而言
First(a) = a

對於`nonTerminal A`而言，有多條規則決定

1. $\begin{cases} A \to B_1B_2B_3 \dots B_n, n \geq 1 \\ B_1, B_2, B_3, \dots, B_n \ all\ can\ be\ \lambda \end{cases} \to \ First(A) \supseteq \lambda$

2. $\begin{cases} A \to B_1 \dots \ B_i \dots \ B_n, \ 1 \leq i \leq n \\ B_1, B_2, \dots, B_{i-1} \ all\ can\ be\ \lambda \end{cases} \to \ First(A) \supseteq \bigcup_{j=1}^{i} (First(B_j) - \lambda)$

1. 第一條規則很明顯將$B_1, B_2, B_3, \dots, B_n$都換成λ會得到$A \to \lambda$，因此First(A)會包含λ
2. 第二條規則比較複雜一點，依序用λ代換掉$B_1, B_2, .., B_{i-1}$可得下列結果(這邊為了方便舉例假設n>5)
    1. $A \to B2B3 \dots \ B_n$ ⇒ $First(A) \supseteq (First(B_2) - \lambda)$
    2. $A \to B3B4 \dots \ B_n$ ⇒ $First(A) \supseteq (First(B_3) - \lambda)$
    3. ....
    4. $A \to B_iB_{i+1} \dots \ B_n$ ⇒ $First(A) \supseteq (First(B_i) - \lambda)$
    再來加上原本沒代換前能得到的$First(A) \supseteq (First(B_1) - \lambda)$，就能知道A的First set包含的B1到Bi的聯集(聯集不包含λ)，不包含λ的原因很簡單，因為可能不是所有symbol都可代換成λ，ex: A → BCD，若B,C可能為λ且First(D)={d}，那麼代換所有可能為λ的symbol後為A→d，不可能代換出A→λ，因此在將First(B)加入First(A)時，連First(A)中的λ一起加會是錯的。

follow set的定義

就是對於一個symbol而言，在它後面的第一個terminal symbol，會在它的follow set中，ex:
ex: 下面有三條production rule

```
1. A -> BC
2. B -> b
3. C -> c
```

每條production rule會有兩部分組成，LHS(left hand side)和RHS(right hand side) ，LHS是可以被代換掉的symbol，RHS是代換後產生的symbols，對於計算Follow(B)而言，要看每條production rule的RHS是否有出現B，若有的話，要看B後面的symbol是否為nonTerminal symbol，如果是terminal symbol就將其加入Follow(B)，如果是nonTerminal就將其代換掉，將代換出的第一個terminal symbol加入Follow(B)

像是可以看到只有第一條production rule有B，所以替換掉在它之後的C，得到`A -> Bc`，因此我們能得知在B後面的第一個terminal symbol有c，c會在Follow(B)中。
跟first set的概念結合，若定義在First(C)中的symbol為S，可以這樣替換`A -> BC` ⇒ `A -> B S ...`，可得知Follow(B)會包含First(C)。

下面是龍書上寫的計算follow set的演算法:
To compute FOLLOW(A) for all nonterminals A, apply the following rules
until nothing can be added to any FOLLOW set

1. Place $ in FOLLOW(S), where S is the start symbol, and $ is the input
   right endmarker.
2. if there is a production $A \to \alpha B \beta$, then everything in FIRST(β) except ϵ is in FOLLOW(B).
3. If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$, where FIRST(β) contains ϵ, then everything in FOLLOW (A) is in FOLLOW (B)

如果龍書上的定義看不懂，或是看的懂，但是會沒有頭緒要怎麼實作或具體能**正確**算出follow set的演算法，可以往下看說明龍書的定義中隱含的東西，以及另一種定義方式、圖片解釋與pseudocode

龍書的「a production」解釋與隱涵的東西:
龍書指的a production的適用範圍並非只是最初文法給你的production rule，也包含所有根據最初production rule產生出的一堆production rule，ex:
若文法為下

```
A -> B C D
B -> b | λ
C -> c | λ
D -> d
```

針對最初這些production rule應該可以求出(不展開產生更多production的情況)

```
Follow(A) = {$} ; Follow(B) = {c} ; Follow(C) = {d} ; Follow(D) = {$}
```

結果很明顯是錯的這是因為也要包含經過代換後產生的production rule，如下(**這裡只代換出關鍵的production rule**)

```
A -> B C D | B D
B -> b | λ
C -> c | λ
D -> d
```

然後在以上述的對照龍書上的定義算出follow set，如下

```
Follow(A) = {$} ; Follow(B) = {c, d} ; Follow(C) = {d} ; Follow(D) = {$}
```

至於要怎麼**只代換出關鍵的production rule**這就是龍書隱涵的地方，雖然也可以全部爆開，產生出超級多production rule

在這邊用三條規則描述另一種nonTerminal's follow set如何計算的定義

$$1.\ \begin{cases} A \to B_1 \dots B_i B_{i+1} \dots B_n,\ 1 \le i \le n \ and \ n \ge 1 \\ B_{i+1}, B_{i+2}, \dots, B_n \ all \ can \ be \ \lambda \\ B_i \ is \ nonTerminal \ symbol \end{cases} \to \ Follow(B_i) \supseteq Follow(A)$$

$$2.\ \begin{cases} A \to B_1 \dots B_i B_{i+1} \dots B_n,\ 1 \le i \le n-1 \ and \ n \ge 2 \\ B_{i+1}, B_{i+2}, \dots, B_k \ all \ can \ be \ \lambda,\ i \le k \le n-1 \\ B_i \ is \ nonTerminal \ symbol \end{cases} \to \ Follow(B_i) \supseteq \bigcup_{j=i+1}^{k+1} (First(B_j) - \lambda)$$

$$3.\ if A \ is \ start \ symbol, add \ \$ \ into \ Follow(A)$$

一些表示法的說明:
出現的連續一連串symbols，往右一個symbol的下標都是遞增1，且可能代表沒有symbol，如下例子

1. $B_1 \dots B_i B_{i+1} \dots B_n$，當n=0時，會沒有任何symbol，當n=1時，為一個symbol B_1；當n=2時，為兩個symbol $B_1 B_2$，...以此類推
2. $B_{i+1}, B_{i+2}, \dots, B_k$，當i=1且k=1時，會沒有任何symbol；當i=1且k=2時，會有一個symbol B_2；當i=1且k=3時，會有兩個symbol $B_2 B_3$

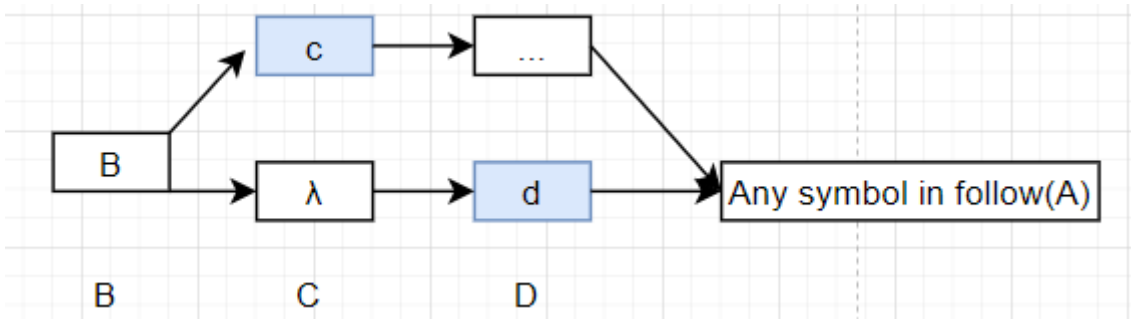
規則解說:

1. 第一條規則: 令F為任何在Follow(A)的symbol，可將原本A symbol出現的地方描述為`... A F ...`，也就是A後面跟著的symbol一定屬於A的follow set，
   把A用「$B_1 \dots B_iB_{i+1} \dots B_n$」換掉後得「$B_1 \dots B_iB_{i+1} \dots B_n\ F$」，
   因為「$B_{i+1}, B_{i+2}, \dots, B_n$」可能都是λ，將那些symbol都代換成λ，會得到「$\dots B_i\ F$」，因此Follow(B_i)會包含Follow(A)。
   這邊要特別注意，i 可能為 n，因為後面沒有symbol，因此「$B_{i+1}, B_{i+2}, \dots, B_n$可能都是λ」會成立，推論出Follow($B_n$)會包含Follow(A)
2. 第二條規則: 比較複雜一點，依序用λ代換掉$B_{i+1}, B_{i+2}, \dots, B_k$可得下列結果(這邊為了方便舉例假設n>5，以及$B_i$為nonTerminal symbol)
   1. $A \rightarrow \dots B_iB_{i+2}B_{i+3} \dots B_n \Rightarrow Follow(B_i) \supseteq (First(B_{i+2}) - \lambda)$
   2. $A \rightarrow \dots B_iB_{i+3}B_{i+4} \dots B_n \Rightarrow Follow(B_i) \supseteq (First(B_{i+3}) - \lambda)$
   3. ....
   4. $A \rightarrow \dots B_iB_{k+1} \dots \Rightarrow Follow(B_i) \supseteq (First(B_{k+1}) - \lambda)$
   再來加上原本沒代換前能得到的$Follow(B_i) \supseteq (First(B_{i+1}) - \lambda)$，就能得到如同規則中的結果。
   First set都要去掉λ，因為就算在文法中沒特別描述`$(EOF symbol)`，nonTerminal symbol的後面接的就算看起來是空的也應該是`$(EOF symbol)`
3. start symbol為文法最一開始要被代換掉的symbol，因此後面一定會接$(EOF symbol)

為了方便理解有提供圖來幫助理解，如下面例子:
grammar為

```
1. A -> B C D
2. B -> b | λ
3. C -> c | λ
4. D -> d
```

如果是計算第一條規則會加入甚麼symbol到B的follow set的話，可以畫出下圖

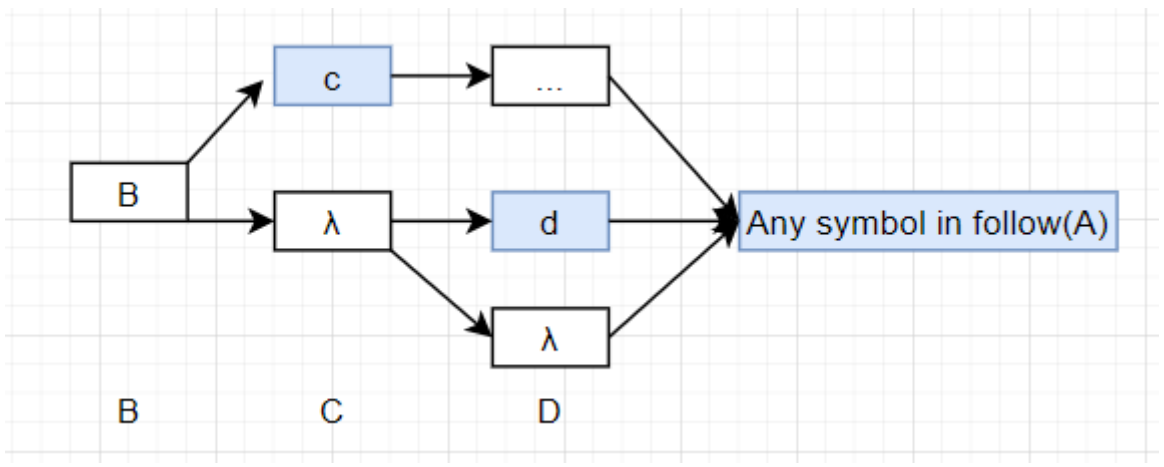


1. 從左到有三層，從root - B，到第一條規則結束的symbol D，像是C這層因為C可以被換成c或λ，因此從B延伸出兩個節點。
2. 內容為「...」的節點是不需要關注的部分，因為很確定不會在B的follow set
3. 標為藍色的節點中的symbol是屬於B的follow set，很明顯可以看到它會是出現在B的第一個terminal symbol，因此符合定義

最後結果就是Follow(B)要加入(First(C) - λ)和(First(D) - λ)，然後由於D不能被換成λ，因此不會再繼續往後
另一個例子是如果剛才的文法的D symbol可以被換成λ，文法如下

```
1. A -> B C D
2. B -> b | λ
3. C -> c | λ
4. D -> d | λ
```

可以畫出下圖



由於B後面的symbol都可以被換成λ(也就是first set中有λ)，因此會走到底，會碰到Follow(A)中的symbol，因此要將(First(C) - λ)、(First(D) - λ)和Follow(A)都加到**Follow(B)**

計算follow set的pseudocode

這邊提供一種可以參考的計算方式，是個採取反覆計算，直到所有nonTerminal的follow set不變為止的做法，此做法會做一個以上個pass，每個pass，會計算每個prodoction rule's RHS的每個nonTerminal symbol的follow set，因為follow set在計算時會有相依關係，相依的變了就必須重算，當遇到一個pass，計算過程中follow set會改變，就要進到下個pass重算，反之，當遇到一個pass，在計算過程中follow set都不變，就可以停下，pseudocode如下

```
add $ into Follow(start symbol)
do {
    loop each rule R like A -> B1 B2 B3 ... Bn in production rules:
        loop each nonTerminal symbol B in rule R's RHS - 「B1 B2 B3 ... Bn」:
            loop each symbol C after B in rule R:
                add (First(C)-λ) into Follow(B)
                if symbol C can not be λ:
                    break loop
            if 「each symbol C after B in rule R」 can be λ:
                add Follow(A) into Follow(B)
} while(break if all nonTerminal symbol's follow set not change);
```

1. 這邊指的production rules只要看原本文法的production rules就可，不用包含經由代換產生的production rule
2. 上面的pseudocode要注意的是這邊描述的 `if 「each symbol C after B in rule R」 can be λ`，如果B後面沒symbol會符合此條件。
3. 還有loop each XXX，這部分會由左到右遍歷每個XXX

題目

此題會給予文法定義與各個**nonTerminal的first set**，求所有**nonTerminal的follow set**

1. 輸入 : 一開始先輸入一行一個nonTerminal的文法規則，之後接一行 `END_OF_GRAMMAR` 後，一條一行nonTerminal's First set的symbols，之後接一行 `END_OF_FIRST_SET` 就結束輸入。要特別注意這邊會用 `;` 來代替 `λ` (空)，以及會用 `|` 來區隔規則，然後第一條文法規則會是start symbol S的production rules，ex:
   輸入範例如下

   ```
   S BCD|E
   B b|;
   C c|;
   D d
   E e
   END_OF_GRAMMAR
   S bcde
   B b;
   C c;
   D d
   E e
   END_OF_FIRST_SET
   ```

   文法規則的部分: `S BCD|E` 代表，S可以被換成 `BCD` 這三個symbol，或 `E` 這個symbol。
   First set的部分: `S bcde` 代表，S的First set中有 `bcde` 這四個symbol
2. 輸出: 依據ASCII由小到大的順序來決定輸出nonTerminal的Follow set的順序，一行輸出一個nonTerminal的Follow set - 「先輸出nonTerminal，接一個空格，再依ASCII由小到大的順序輸出其Follow set中的symbol」，ex:
   輸出範例如下(為上面的輸入範例的輸出)

   ```
   B cd
   C d
   D $
   E $
   S $
   ```

   因為S是start symbol，因此其Follow set會有$(EOF symbol)

題目的公開測資的**input和output**

1. case 1
   input

```
S ABC
A a|Cb|;
B C|dA|;
C e|f|;
END_OF_GRAMMAR
S ;abdef
A ;abef
B ;def
C ;ef
END_OF_FIRST_SET
```

output

```
A $def
B $ef
C $bef
S $
```

2. case 2
   input

```
S aBDh
B cC
C bC|;
D EF
E g|;
F f|;
END_OF_GRAMMAR
S a
B c
C ;b
D ;gf
E ;g
F ;f
END_OF_FIRST_SET
```

output

```
B fgh
C fgh
D h
E fh
F h
S $
```

3. case 3
   input

```
S AaAb|BbBa
A ;
B ;
END_OF_GRAMMAR
S ab
A ;
B ;
END_OF_FIRST_SET
```

output

```
A ab
B ab
S $
```

4. case 4
   input

```
S AC
C c|;
A aBCd|BQ
B bB|;
Q q|;
END_OF_GRAMMAR
S ;abcq
C ;c
A ;abq
B ;b
Q ;q
END_OF_FIRST_SET
```

   output

```
A $c
B $cdq
C $d
Q $c
S $
```

可以參考的程式碼

input處理程式碼片段與一些儲存的資料的struct(C++)

```cpp
#include <vector>
#include <iostream>
#include <string.h>

struct Rule {
    // if has a rule like 「A -> BCD」
    // its nonTerminal would be A
    // its production would be BCD
    char nonTerminal;
    std::string production;
public:
    Rule(char nonTerminal, std::string production) {
        this->nonTerminal = nonTerminal;
        this->production = production;
    }
};

struct FirstSet {
    // if A's first set contains b, c, d three symbol
    // its nonTerminal would be A
    // its set would be bcd
    char nonTerminal;
    std::string set;
public:
    FirstSet(char nonTerminal, std::string set) {
        this->nonTerminal = nonTerminal;
        this->set = set;
    }
};

// put the follow code into start of your main function
const int maxLineLength = 500;
// the code to handle input
```

```
std::vector<Rule> rules;
char input[maxLineLength];
// handle grammar part
std::cin.getline(input, maxLineLength);
while (strcmp(input, "END_OF_GRAMMAR")) { // if same, strcmp would return 0
    char nonTerminal = input[0];
    std::string production;
    for (int i = 2; i < strlen(input); i++)
    {
        if (input[i] != '|') {
            production += input[i];
        }
        else {
            rules.push_back(Rule(nonTerminal, production));
            production = "";
        }
    }
    rules.push_back(Rule(nonTerminal, production));
    std::cin.getline(input, maxLineLength);
}
```

```
std::vector<FirstSet> firstSets;
// handle first set part
std::cin.getline(input, maxLineLength);
while (strcmp(input, "END_OF_FIRST_SET")) { // if same, strcmp would return 0
    char nonTerminal = input[0];
    std::string production;
    for (int i = 2; i < strlen(input); i++)
    {
        production += input[i];
    }
    firstSets.push_back(FirstSet(nonTerminal, production));
    std::cin.getline(input, maxLineLength);
}
/* sample code for observing the data, which constructed by input
for (int i = 0; i < rules.size(); i++)
{
    std::cout << rules[i].nonTerminal << " -> " << rules[i].production <<
    std::endl;
}

for (int i = 0; i < firstSets.size(); i++)
{
    std::cout << firstSets[i].nonTerminal << "'s first set is " <<
    firstSets[i].set << std::endl;
}
*/
```