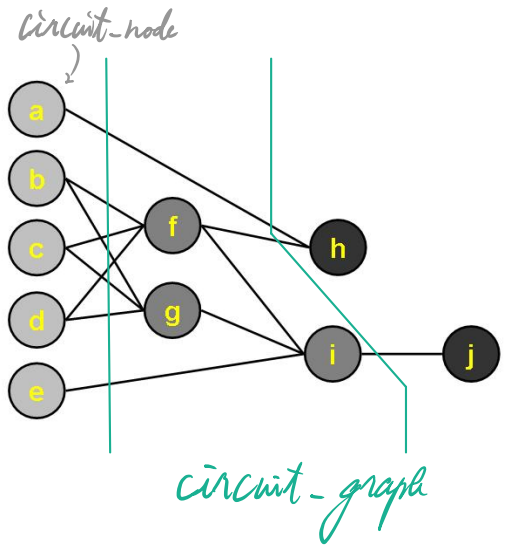
CAD Design Project 1 – Circuit Graph Construction
Due: 23:55, Oct. 2, 2019

Graph is a commonly used computer data structure to handle electronic circuits. In this kickoff project, you are required to construct a circuit graph from a text-input file in BLIF format. BLIF is developed by University of California, Berkeley to describe a circuit in Boolean level. You need to read Section 1 and Section 2 of the BLIF document first. After understanding how BLIF works, you are required to write a program, which parses a BLIF file and constructs its corresponding circuit graph, on a Linux environment. The constructed circuit graph will be used in following projects. There are 3 requirements for your program:

1. Output the Boolean functionality of each node into a text file (file name: function.out).
2. Report the predecessors and successors of a node.
3. Upload your source code tarball (*.tgz) to moodle (including your Makefile).
   (NOTE: The uploaded file name should be the same with your student ID.)

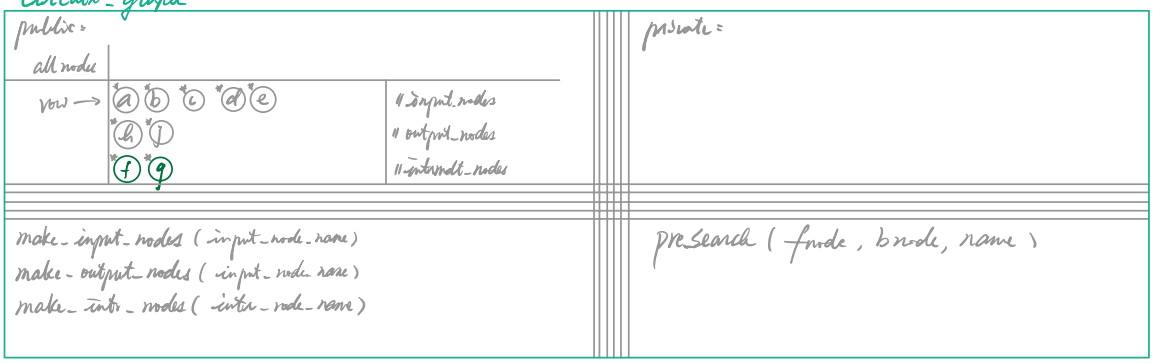
BLIF Example: sample01.blif

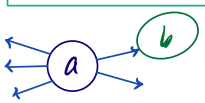
```
.model sample01
.inputs a b c d e
.outputs h j
.names b c d f
101 1
.names b c d g
11- 1
--1 1
.names a f h
1- 1
-0 1
.names e f g i
110 1
001 1
.names i j
0 1
.end
```



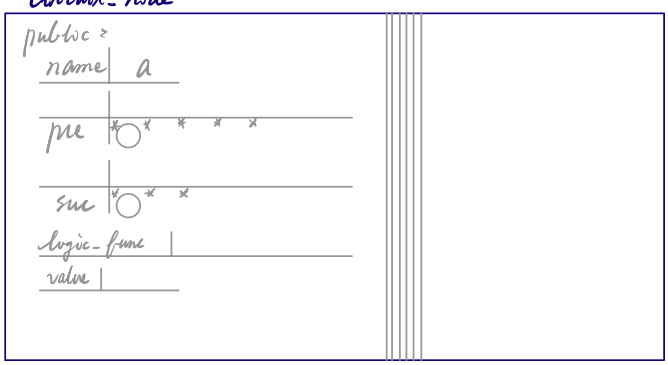

```
Run-time Example:
%> blif_parser sample01.blif
Please input a node: f
predecessor: b, c, d
successor: h, i
Please input a node: d
predecessor: -                  # no predecessor
successor: f, g
Please input a node: x
node x does not exist
Please input a node: h
predecessor: a, f
successor: -                    # no successor
...
Please input a node: 0          # '0' stands for end of user inputs
%>

%> cat function.out
Node function:
f = b c' d
g = b c + d
h = a + f'
i = e f g' + e' f' g
j = i'
END
```

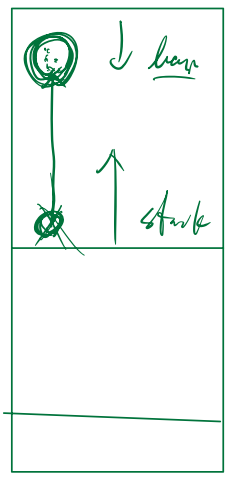
circuit_graph





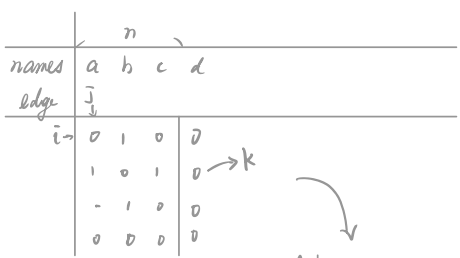
circuit_node





new

make_shared



e[0]

	0	1
j→	a	a'
	b	b'
	c	c'

e[1]

	0	1
j→	a'	a
	b'	b
	c'	c

d = (a+b'+c)(a'+b+c')(b'+c)(a+b+c)

A

a	b' c
a'	b c'
b'	c
a	b c

k = 0

e[k][j][edge[i][j]]

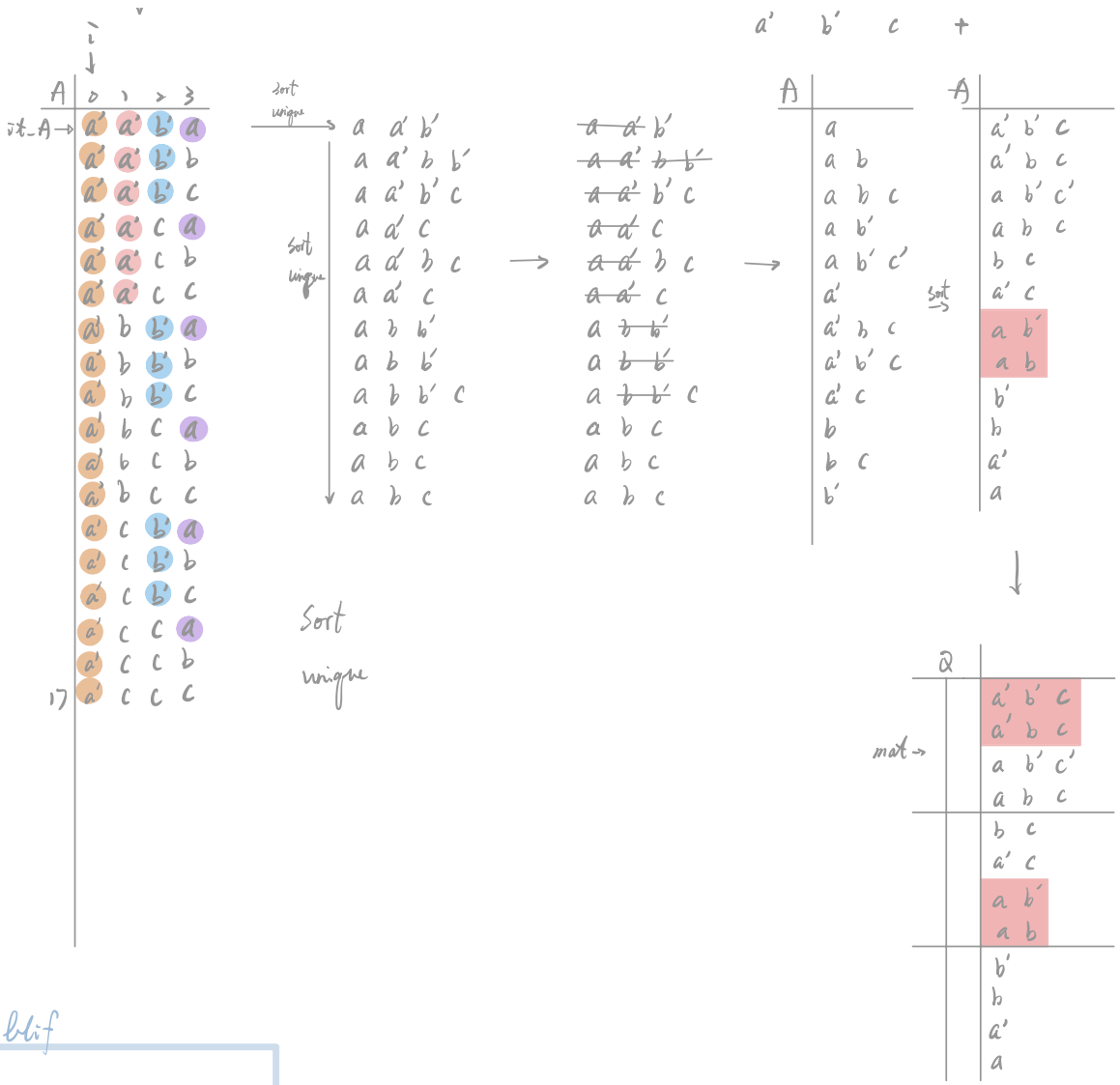
d = a' b c' + a b' c + b c' + a' b' c'

B

a'	b c'
a	b' c
	b c'
a'	b' c'

k = 1

A	0	1	2	up	set
0	a'	b'	c	1×3×2×3	1
1	a'	b	c	1×3×2	1×3
2	b'	c		1×3	1×3×3
3	a	b	c	1	1×3×3×3



.btif

names	a	b	c	d
0	1	0	0	
1	0	1	0	
2	1	0	0	
3	0	0	0	

(1) 第1個一維 vector

	0	1	2	3
names	a	b	c	d

(2) 第1個二維 vector

edges	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	2	1	0	0
3	0	0	0	0

(3) 傳給 vector<vector<string>> func(names, edges)

vector<vector<string>> func(names, edges) {

edges	0	1	2	3
0	0	1	0	0
1	1	0	1	0

轉成 2維 vector A

A	0	1	2
0	a	b'	c
1	a'	b	c'

表示 d = (a' + b' + c) · (a' + b + c') · (b' + c) ·

2	-	1	0	0
3	0	0	0	0

2	b' c
3	a b c

(a + b + c)

↓ dynamic programming

↓ 幂等展開

化簡後 → 回傳

B	0	1	2	3
0	a'	a'	b'	a
1	a'	a'	b'	b
2	a'	a'	b'	c
3	a'	a'	c	a
	a'	a'	c	b
	a'	a'	c	c
	a'	b	b'	a
	a'	b	b'	b
	a'	b	b'	c
	a'	b	c	a
	a'	b	c	b
	a'	b	c	c
	a'	c	b'	a
	a'	c	b'	b
	a'	c	b'	c
	a'	c	c	a
	a'	c	c	b
	a'	c	c	c

最末 = d = a' a' b' a +
a' a' b' b +
a' a' b' c +
a' a' c a
⋮

rep=1, set=1×3×3×2

rep=1×3, set=1×3×3

rep=1×3×2, set=1×3

rep=1×3×2×3, set=1

```
C(B, i, it_A) {
  if( i == B.size()-1 ) {
    for( j(0) ; j != B[i].size()-1 ; ++j ) {
      copy = *it_A ;
      it_A -> push_back( B[i][j] ) ;
      ++ it_A ;
      * it_A = copy ;
    }
    it_A -> push_back( B[i][B.size()-1] ) ;
    ++it_A ;
  }
  else {
    for( j(0) ; j != B[i].size() ; ++j ) {
      if( j != 0 ) {
        it_A.emplace_back( (it_A-1) -> begin(), (it_A-1) -> (begin()+i) ) ;
        it_A -> push_back( B[i][j] ) ;
        C( B, i+1, ++it_A ) ;
      }
      else {
        it_A -> emplace_back( B[i][j] ) ;
        C( B, i+1, ++it_A ) ;
      }
    }
  }
}
```