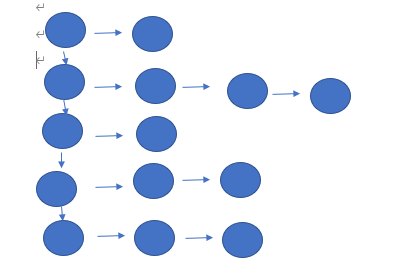
# 总体流程



# DPLL

结构采用双向链表的方式存储CNF，

如下图：



具体结构定义为：

**子句节点**：

typedef struct HeadNode {

int Num = 0;

DataNode \*right{};

HeadNode \*down{};

}HeadNode;

Num记录该行子句的变元节点个数，指针right指向到子句第一个变元节点，指针down指向到下一行子句。

**子句变元节点：**

typedef struct DataNode {

int data = 0;

DataNode \*next{};

}DataNode;

data记录该变元节点的值（即几号变元，如163即代表第163个变元），指针next指向下一个变元节点。

**变元真值集：**

struct consequence {

int value = -1;//存真值 真时为true-1，假时为false-0

};

最终使用时，构造一个consequence数组，下标即代表几号变元，最终的答案就是这个

DPLL算法：

DPLL算法是一种基于树的回溯算法，主要使用两种基本处理策略：

单子句规则。如果子句集S中有一个单子句L,那么L一定取真值，于是可以从S中删除所有包含L的子句（包括单子句本身），得到子句集S1，如果它是空集，则S可满足。否则对S1中的每个子句，如果它包含文字¬L,则从该子句中去掉这个文字，这样可得到子句集合S2。S可满足当且仅当S2可满足。单子句传播策略就是反复利用单子句规则化简S的过程。

分裂策略。按某种策略选取一个文字L.如果L取真值，则根据单子句传播策略，可将S化成S2；若L取假值（即¬L成立）时，S可化成S1.

伪代码如下：

Status DPLL( S) {

/\* S为公式对应的子句集。若其满足，返回TURE；否则返回FALSE. \*/

while(S中存在单子句) {

在S中选一个单子句L，并将单子句装入结果数组，value置为1；

依据单子句规则，利用L化简S；

if S = Φ return(TRUE);

else if (S中有空子句 ) return（FALSE）；

}

V = S第一个数据节点数值

if DPLL（S ∪v ）return(TURE);

return DPLL(S ∪¬v);

}

1）HeadNode\* IsSingleClause（HeadNode\*）函数：寻找是否存在单子句。遍历子句头节点，若任意一行的Num存在为1的情况，即证明存在单子句，并返回单子句的指针。

2）void DeleteHeadNode(HeadNode\* des, HeadNode\* &src) 函数：在src指向的子句集中删掉des子句。此处应用了src的引用指针变量（能够同时改变指针指向的内容和指针的值）。

3）void DeleteDataNode(int data.HeadNode\*&src) 函数：在src指向的子句集中若有变元的数值与data相等，则删除该变元所在的子句；若只是绝对值相等（即互为相反数），则删除该变元。

4）status IsEmptyClause(HeadNode\* src)函数：判断src函数中是否存在空子句，即遍历头节点，若存在头节点不为空且Num值为0的，即表示存在空子句，返回TRUE；否则，返回FALSE。

5）HeadNode\* ADDSingleClause(HeadNode\* src,int Var)函数：将变元Var装载成为单子句，并将该单子句装载到src的子句集中，依照函数内容，该单子句始终添加到头节点，最终返回头节点指针。

6）HeadNode\* Duplication(HeadNode\* src)函数：将src所指向的子句集复制一个副本，并将副本的头指针返回。

总体实现的思路是递归+循环+剪枝，同时答案是保存在一个consequense数组（答案即在此之中）中，这里的consequense数组曾考虑过也每次都拷贝一个副本用于严格的保存答案，但是仔细想想发现我们只需得到一个解就结束整个DPLL过程，那么最终的consequense数组一定是保存了答案序列的，但是别的变元取的值可能改变过（显然），但是这并不会影响最终的答案，所以为了性能，并没有每次都拷贝result，当然如果为了得到完美的答案（即在最终的答案result中体现到底是哪些变元决定了这个解），拷贝consequense数组是值得的

# 数独部分

这里考虑的是一个9\*9的数独，那么可以把每一个位置的每种数字（1-9）是否存在（true or false）看成是一个变元，那么就有729（9\*9\*9）个变元，然后再利用数独的规则对变元采取限制，得到子句集，转换成cnf公式，然后再利用我们的DPLL求解这一CNF范式，得到结果。

公式如下：

