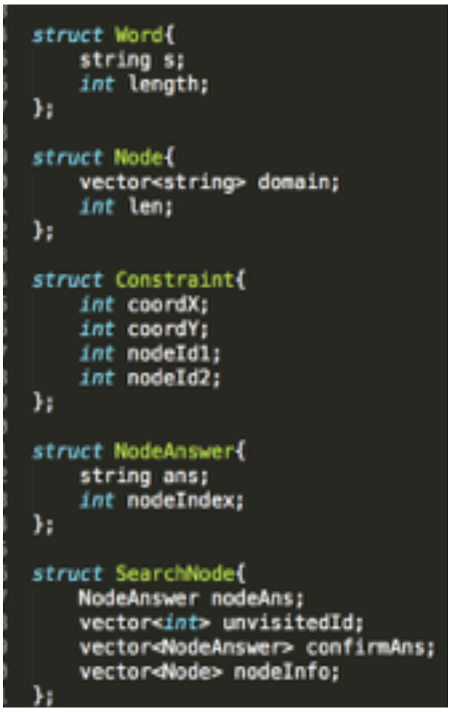
AI Assignment #1 0416206 李明峻

Data Structure :



```
struct Word{
    string s;
    int length;
};

struct Node{
    vector<string> domain;
    int len;
};

struct Constraint{
    int coordX;
    int coordY;
    int nodeId1;
    int nodeId2;
};

struct NodeAnswer{
    string ans;
    int nodeIndex;
};

struct SearchNode{
    NodeAnswer nodeAns;
    vector<int> unvisitedId;
    vector<NodeAnswer> confirmAns;
    vector<Node> nodeInfo;
};
```

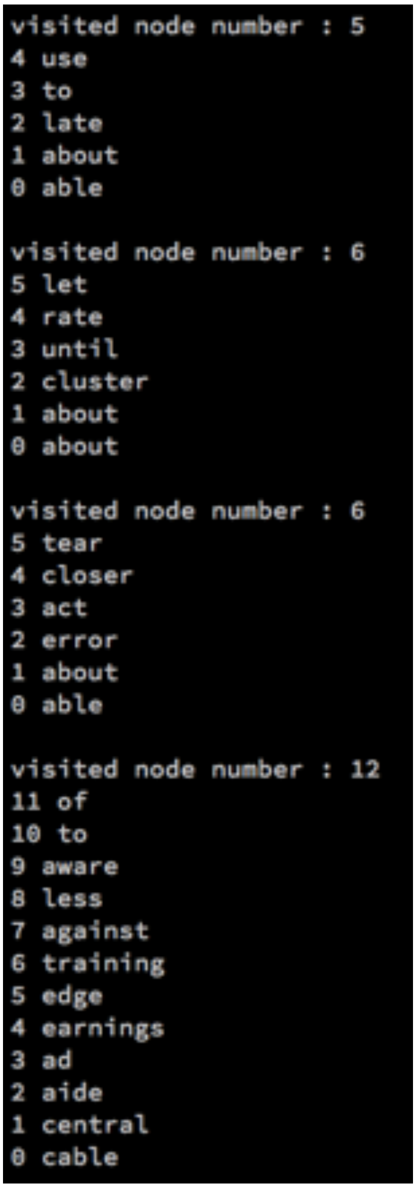
1.struct Word : this is the data structure of word dictionary
2.struct Node : this is the data structure of the variables(nodes) , which represents the row or column of the problem map , it contains the domain of this nodes.
3.struct Constraint : this is the data structure of the constraint of nodes , representing the intersection of two vertical nodes , it contains the coordinate of the intersection point and the relevant nodes index.
4.struct NodeAnswer : this is the data structure of the assignment of each node.
5.struct SearchNode : this is another data structure of node while expanding the node , the reason that I create this data structure is I have to record the assignment of each step(node expansion) , including the node index of the selected node and the word of its domain to be assigned ; and the unvisited node array ; also store the confirmed assignment information of nodes , which can help me break out the loop , the goal is to store the final solution of this problem ; finally , vector<Node> nodeInfo is to store the domain of all the node , because after AC-3 trimming , the domain would be checked and modified under any circumstance.

Method : first step , I use unary constraint (word length) to trim the domain , then I create a queue to store the binary constraint of all the nodes (binary constraint is the constraint created by intersection of two nodes) , then I do the first time AC-3 process to trim the domain , the AC-3 algorithm process is : Check all the constraint by the nodes pair in the queue , starting from the beginning of the queue , according to each constraint's intersection coordinate and some information to further filter the domain of each node , the most important part is that after I check out one constraint , I will push it to a vector , next time when I modify the relevant node's domain of this constraint , I will again push this constraint to the original queue , because this constraint's domain need to be checked again , and remember to pop out that constraint from the vector names "modifiedConstraint " . After first step domain processing , the domain of each node has shrink a lot , this can speed up my searching step , then I start to search (node expansion) , firstly , I create a stack to store the nodes to be expanded , every time getting a node popped from the stack top , I will choose a word from its domain to fit in this variable as the domain of this node , and this word is the only word in the domain , so the domain size of this node would be 1 , then pass this information to the AC3 function , to check whether this word can be the answer or not , if it fails to pass AC3 algorithm (some node's domain become empty) , it will return false and keep searching another word from its origin domain , until find a word that can pass AC3

algorithm , keep doing this process , until I find that the assignment record list ("confirmAns") size is equal to the number of all nodes , this means I find all the certain answer of each node , then break out the loop , then finishing searching.

Results:

My result is on the right , my program will jump out while finding one solution , so the answer would be only one no matter how many times I run , because the dictionary words order stand still all the time , so does the order I find the solution.



```
visited node number : 5
4 use
3 to
2 late
1 about
0 able

visited node number : 6
5 let
4 rate
3 until
2 cluster
1 about
0 about

visited node number : 6
5 tear
4 closer
3 act
2 error
1 about
0 able

visited node number : 12
11 of
10 to
9 aware
8 less
7 against
6 training
5 edge
4 earnings
3 ad
2 aide
1 central
0 cable
```

Observation:

1.How the numbers of visited nodes change with or without any of the heuristics?

ANS: The number of searching node with heuristics would much less than the method without heuristics , because with heuristics , we can filter out much more abundant words , which efficiently speed up our trimming process , we can skip a lot of unnecessary path when searching for solution.

2.How the numbers of visited nodes change with or without the initial run of AC-3?

ANS: Obviously , if trimming the initial domain by AC-3 , the number of visited nodes would drop down , because before node expansion , we can shrink node's domain as much as possible , in order to make the testing domain much smaller while searching for the suitable domain that can pass AC-3.

3.Is it possible to do a complete search (not stopping when solutions are found) and determine the numbers of valid solutions for the different puzzles?

ANS: it's absolutely possible to find all the solution of one problem , in my program , just keep searching while finding one answer , don't break out when finding a solution , jump out once all the domain has been checked ,

Experience:

For the degree heuristics , I found that pop out the node that has the most constraints from the stack really help for deduce searching step , because degree heuristics limit the domain of those nodes which have higher number of constraints , so the domain is very small at the beginning of the search , this brings high probability that the searching node can be reduced at the end of the searching step , guaranteeing all the process can be efficient enough.

And for AC-3 algorithm , I think the performance is good when the problem is small , like this assignment , but if the problem size become huge , and the dictionary also contains a large amount of words , the efficiency won’t be that good , maybe we have to find out more effective algorithm to solve this problem , and find another way to process the data.