Yuan Gao z5239220 Q1

You are given a boolean expression consisting of a string of the symbols true, false, separated by operators AND, OR, NAND and NOR but without any parentheses. Count the number of ways one can put parentheses in the expression such that it will evaluate to true.

1. Formulated

> We can assume and put this operation in the string array:
> Use "1" to represent true and use "0" to represent.
> For example: the operation **true AND false OR true NAND true NOR flase** can be write as:
> `formula = {"1", "AND", "0", "OR", "1", "NAND", "1", "NOR", "0"};`

2. Processing formula

> Separate `formula` to the `num` array and `operator` array.
>
> > `formula = {"1", "AND", "0", "OR", "1", "NAND", "1", "NOR", "0"};`
> > `num = {1, 0, 1, 1, 0};`
> > `operator = {"AND", "OR", "NAND", "NOR"};`

3. Create dynamic programming array

> `dp[i][j]` means all solutions of the operation between `i` to `j`.
> For example: In **1 AND 0 OR 1 NAND 1 NOR 0**:
> The initial array of `formula = {"1", "AND", "0", "OR", "1", "NAND", "1", "NOR", "0"};` is:

```
dp = {
    {{1}, { }, { }, { }},
    {{ }, {0}, { }, { }},
    {{ }, { }, {1}, { }},
    {{ }, { }, { }, {1}},
};
```

> `dp[1][3]` means all solutions of the operation **(0 OR 1 NAND 0)**
>
> > Operation: `true AND false OR true NAND true NOR true`
> > Number array: `{1, 0, 1, 1, 0}`
> > Operator array: `{"AND", "OR", "NAND", "NOR"}`
> > Assume calculate `dp[1][3]`
> > Separate `0 OR 1 NAND 1` to the two part:
> >
> > - `0` and `1 NAND 1` -> `operator[1] = "OR"`
> >   The result is `0 OR 0 = 0`
> > - `0 OR 1` and `0` -> `operator[2] = "NAND"`

> > The result is `1 NAND 1 = 0`
> > Therefore, `dp[i][j] = {0, 0};`

> Every part of operation can be calculated as this function. Therefore, assume `n = num.size()`, Just checking how many "1" in the `dp[0][n - 1]` can get the result.

4. Code

```
bool isOp(string& c){
    return c == "AND" || c == "OR" || c == "NAND" || c == "NOR";
}

int calculate(int& num1, string& op, int& num2){
    if(op == "AND")
        return num1 && num2;
    else if(op == "OR")
        return num1 || num2;
    else if(op == "NAND")
        return !(num1 && num2);
    else
        return !(num1 || num2);
}

int diffWaysToCompute(vector<string> formula){
    //number array
    vector<int> nums;

    //operator array
    vector<string> ops;

    //Processing formula
    for(int i = 0; i < formula.size(); i++){
        if(isOp(formula[i])){
            ops.push_back(formula[i]);
        }else{
            if(formula[i] == "1"){
                nums.push_back(1);
            }else{
                nums.push_back(0);
            }
        }
    }

    //result
    int res = 0;
    int n = nums.size();

    //create dynamic programming array
    vector<vector<vector<int>>> dp(n, vector<vector<int>>(n));
```

```
    //init dp[][]
    for(int i = 0; i < n; i++){
        dp[i][i].push_back(nums[i]);
    }

    //push the result in every part
    for(int j = 1; j < n; ++j){
        for(int i = j - 1; i >= 0; --i){
            for(int k = i; k < j; k++){
                for(int r1: dp[i][k]){
                    for(int r2: dp[k + 1][j]){
                        dp[i][j].push_back(calculate(r1, ops[k], r2));
                    }
                }
            }
        }
    }
    return res;
}
```

5. Time complexity

> Assume the formula has n boolean value
>
> > -Calculate dynamic programming array is $O(n^3)$
> > -Merge two result is $O(n!^2)$

> Hence, total time complexity is $O(n!^2 n^3)$