111550113 謝詠晴

Algorithms Exercise #1

a. Environment

I. OS: Windows 11

II. Compiler: GNU GCC Compiler

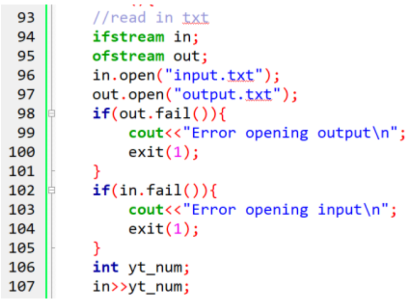
III. IDE: Code::Blocks 20.03

b. Results

I. Methods:

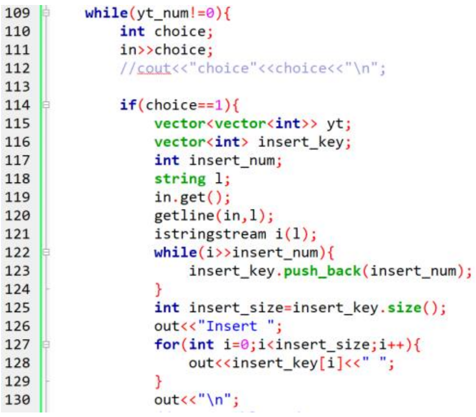
i. <u>Main function:</u>

(1) Using *ifstream* and *ofstream* to input and output txt.file.



```cpp
//read in txt
ifstream in;
ofstream out;
in.open("input.txt");
out.open("output.txt");
if(out.fail()){
    cout<<"Error opening output\n";
    exit(1);
}
if(in.fail()){
    cout<<"Error opening input\n";
    exit(1);
}
int yt_num;
in>>yt_num;
```

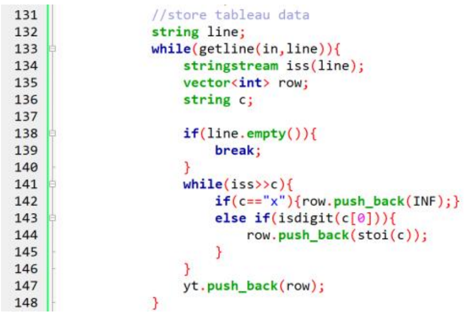
(2) Using while loop to handle each tableau, and perform certain operation based on ‘choice’. Using *getline* and *istringstream* to read in a whole line of input. The inserted numbers are stored in ‘insert_key’ vector.



```cpp
while(yt_num!=0){
    int choice;
    in>>choice;
    //cout<<"choice"<<choice<<"\n";

    if(choice==1){
        vector<vector<int>> yt;
        vector<int> insert_key;
        int insert_num;
        string l;
        in.get();
        getline(in,l);
        istringstream i(l);
        while(i>>insert_num){
            insert_key.push_back(insert_num);
        }
        int insert_size=insert_key.size();
        out<<"Insert ";
        for(int i=0;i<insert_size;i++){
            out<<insert_key[i]<<" ";
        }
        out<<"\n";
```

(3) Store the tableau index in ‘vector<vector<int>> yt’, if the read in index is ‘x’, we store it as INF, which is define as INT_MAX.

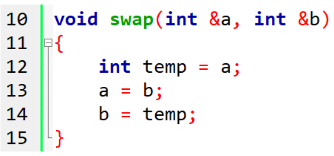


```cpp
        //store tableau data
        string line;
        while(getline(in,line)){
            stringstream iss(line);
            vector<int> row;
            string c;

            if(line.empty()){
                break;
            }
            while(iss>>c){
                if(c=="x"){row.push_back(INF);}
                else if(isdigit(c[0])){
                    row.push_back(stoi(c));
                }
            }
            yt.push_back(row);
        }
```

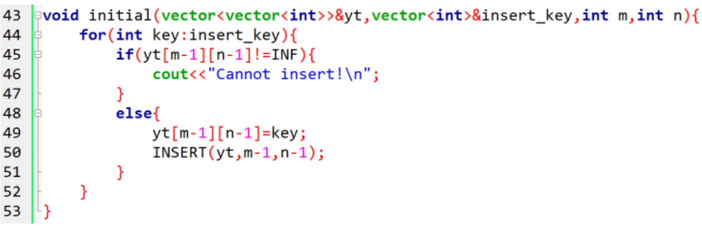
ii. Insert:

(1) First, set a swap function for exchanging two indices. We will use it later.



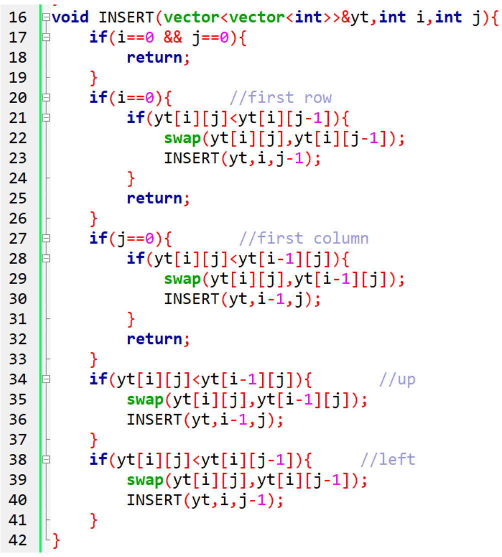
```
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

(2) Then we insert the inserted number aka. 'key' in the last position of tableau (yt [m-1][n-1]), and perform the INSERT function, which reconstruct the tableau that make each index be in the correct position.



```
void initial(vector<vector<int>>&yt,vector<int>&insert_key,int m,int n){
    for(int key:insert_key){
        if(yt[m-1][n-1]!=INF){
            cout<<"Cannot insert!\n";
        }
        else{
            yt[m-1][n-1]=key;
            INSERT(yt,m-1,n-1);
        }
    }
}
```

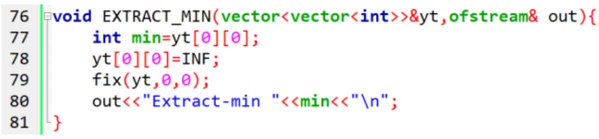
(3) In INSERT function, we use recursive concept. We move the bottom-right corner's index upwards and leftwards until it is in the correct position. We handle the first row and column separately, and the base case is when we reach the top-left corner.



```
void INSERT(vector<vector<int>>&yt,int i,int j){
    if(i==0 && j==0){
        return;
    }
    if(i==0){       //first row
        if(yt[i][j]<yt[i][j-1]){
            swap(yt[i][j],yt[i][j-1]);
            INSERT(yt,i,j-1);
        }
        return;
    }
    if(j==0){        //first column
        if(yt[i][j]<yt[i-1][j]){
            swap(yt[i][j],yt[i-1][j]);
            INSERT(yt,i-1,j);
        }
        return;
    }
    if(yt[i][j]<yt[i-1][j]){        //up
        swap(yt[i][j],yt[i-1][j]);
        INSERT(yt,i-1,j);
    }
    if(yt[i][j]<yt[i][j-1]){      //left
        swap(yt[i][j],yt[i][j-1]);
        INSERT(yt,i,j-1);
    }
}
```

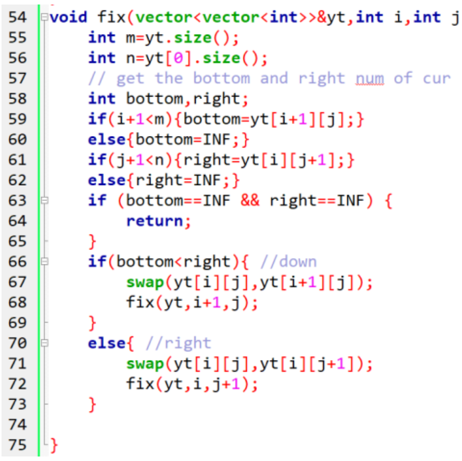
iii. Extract-min:

(1) We know that the min in the tableau is the top-left corner index, we assigned it to 'min' and set its position value to INF. Then use fix function to reconstruct the tableau.



```
void EXTRACT_MIN(vector<vector<int>>&yt,ofstream& out){
    int min=yt[0][0];
    yt[0][0]=INF;
    fix(yt,0,0);
    out<<"Extract-min "<<min<<"\n";
}
```

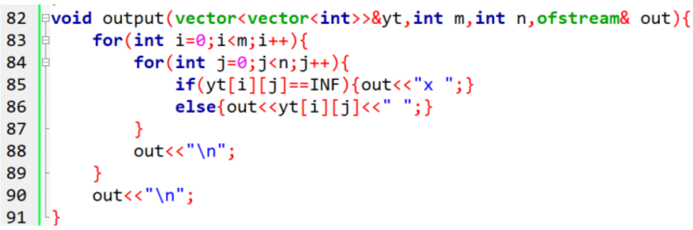
(2) We consider (i, j) as the root node, and find the right and bottom nodes of it. Then we compare its right and bottom node, swap the root node with the smaller node. The function starts from (0, 0) node which value is INF, and implement the fix function recursively until its right and bottom nodes are both INF.



```
void fix(vector<vector<int>>&yt,int i,int j){
    int m=yt.size();
    int n=yt[0].size();
    // get the bottom and right num of cur
    int bottom,right;
    if(i+1<m){bottom=yt[i+1][j];}
    else{bottom=INF;}
    if(j+1<n){right=yt[i][j+1];}
    else{right=INF;}
    if (bottom==INF && right==INF) {
        return;
    }
    if(bottom<right){ //down
        swap(yt[i][j],yt[i+1][j]);
        fix(yt,i+1,j);
    }
    else{ //right
        swap(yt[i][j],yt[i][j+1]);
        fix(yt,i,j+1);
    }

}
```

iv. Output: Using output function to output the result in the txt.file.



```
void output(vector<vector<int>>&yt,int m,int n,ofstream& out){
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            if(yt[i][j]==INF){out<<"x ";}
            else{out<<yt[i][j]<<" ";}
        }
        out<<"\n";
    }
    out<<"\n";
}
```

II. Running time analysis:

- m is the number of rows, n is the number of columns.

i. Insert:

(1) 'INSERT' function: Time Complexity is O(m+n), which is the worst case that it needs to go upwards m times and leftwards n times to its correct position.

(2) 'initial' function: Assume there are k inserted number (key), then it called k times INSERT function. Time Complexity is **O (k* (m+n))**.

ii. Extract-min:

(1) 'EXTRACT-MIN' function called fix function. We divided the m*n problem into (m-1)*n or m*(n-1) subproblem. The worst case is the index at the top-left corner moves downwards m times and rightwards n times, so the time complexity is **O(m+n)**.

iii. Output: Time Complexity is O(m*n).