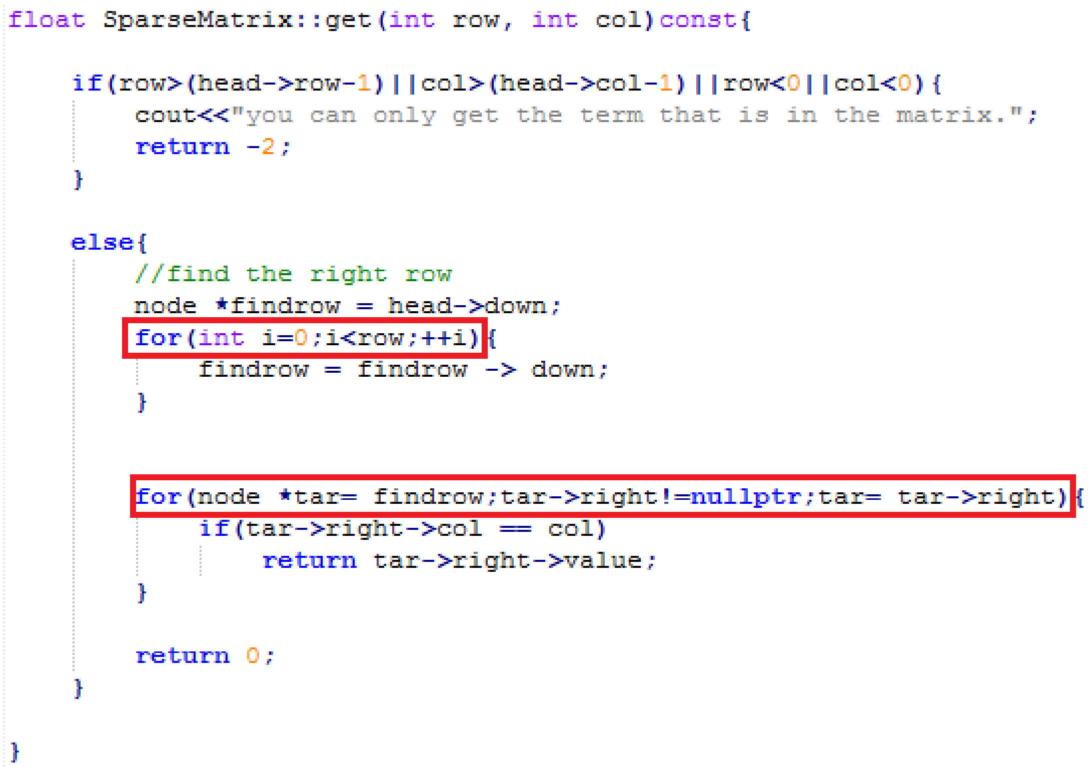
Part 1

- flaot **get** (int row, int col) const

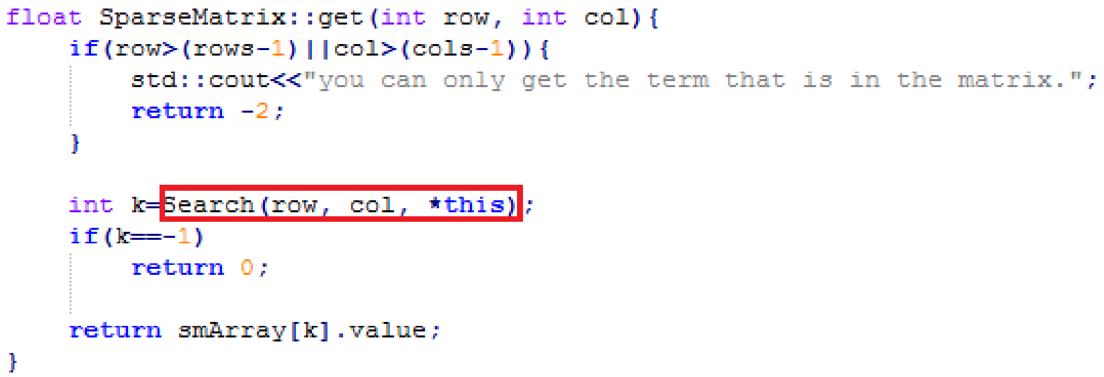
time complexity → **O(max(row, col))**



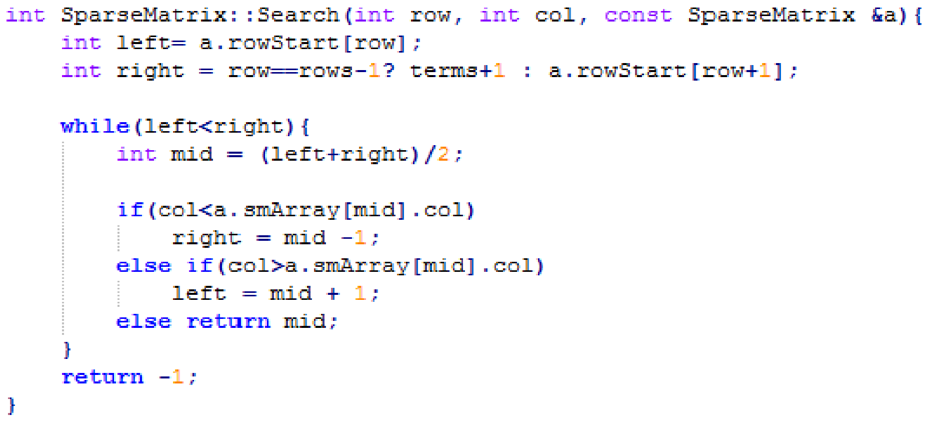
```
float SparseMatrix::get(int row, int col)const{

    if(row>(head->row-1)||col>(head->col-1)||row<0||col<0){
        cout<<"you can only get the term that is in the matrix.";
        return -2;
    }

    else{
        //find the right row
        node *findrow = head->down;
        for(int i=0;i<row;++i){
            findrow = findrow -> down;
        }


        for(node *tar= findrow;tar->right!=nullptr;tar= tar->right){
            if(tar->right->col == col)
                return tar->right->value;
        }

        return 0;
    }

}
```

Compare with last assignment →**O(log(terms)) < O(max(row, col))**

因為上次使用 **binary search** 所以效率較佳



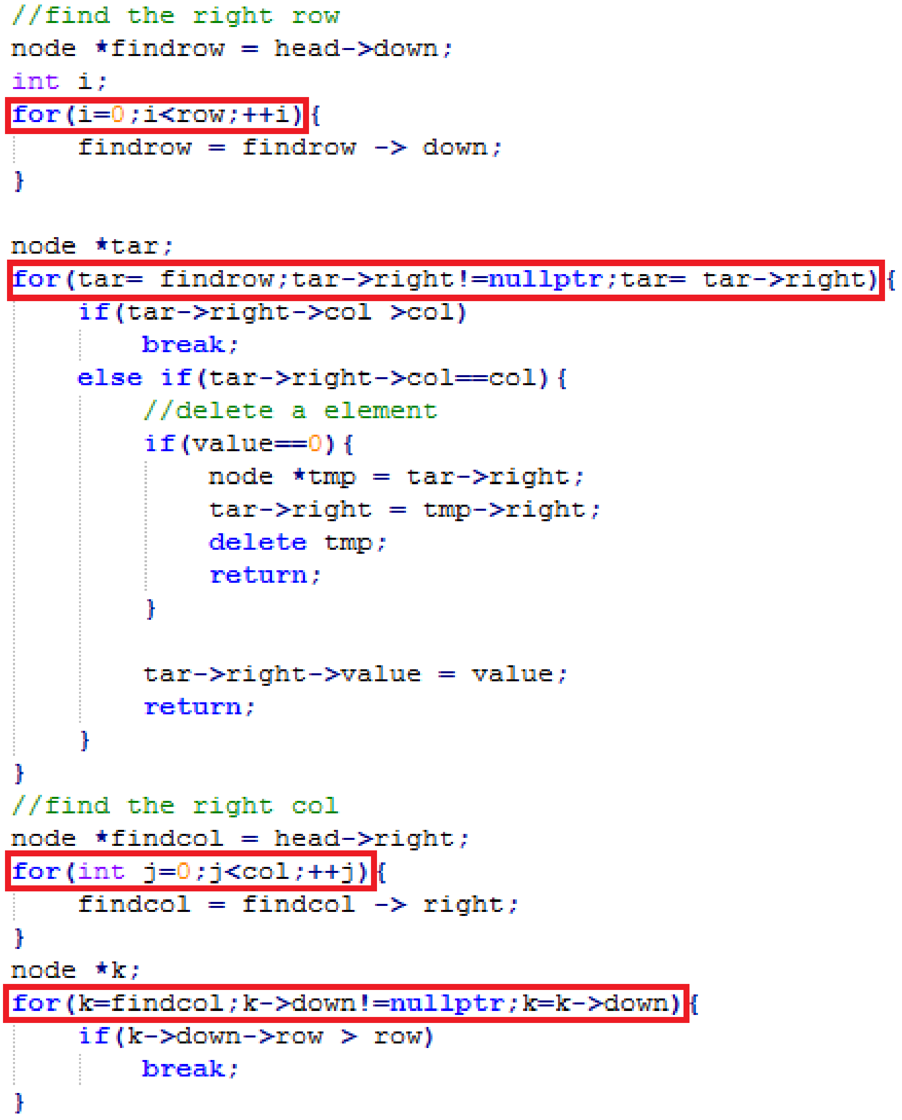
```
float SparseMatrix::get(int row, int col){
    if(row>(rows-1)||col>(cols-1)){
        std::cout<<"you can only get the term that is in the matrix.";
        return -2;
    }

    int k=Search(row, col, *this);
    if(k==-1)
        return 0;

    return smArray[k].value;
}
```



```
int SparseMatrix::Search(int row, int col, const SparseMatrix &a){
    int left= a.rowStart[row];
    int right = row==rows-1? terms+1 : a.rowStart[row+1];

    while(left<right){
        int mid = (left+right)/2;

        if(col<a.smArray[mid].col)
            right = mid -1;
        else if(col>a.smArray[mid].col)
            left = mid + 1;
        else return mid;
    }
    return -1;
}
```

- void **set** (int row, int col, float value)

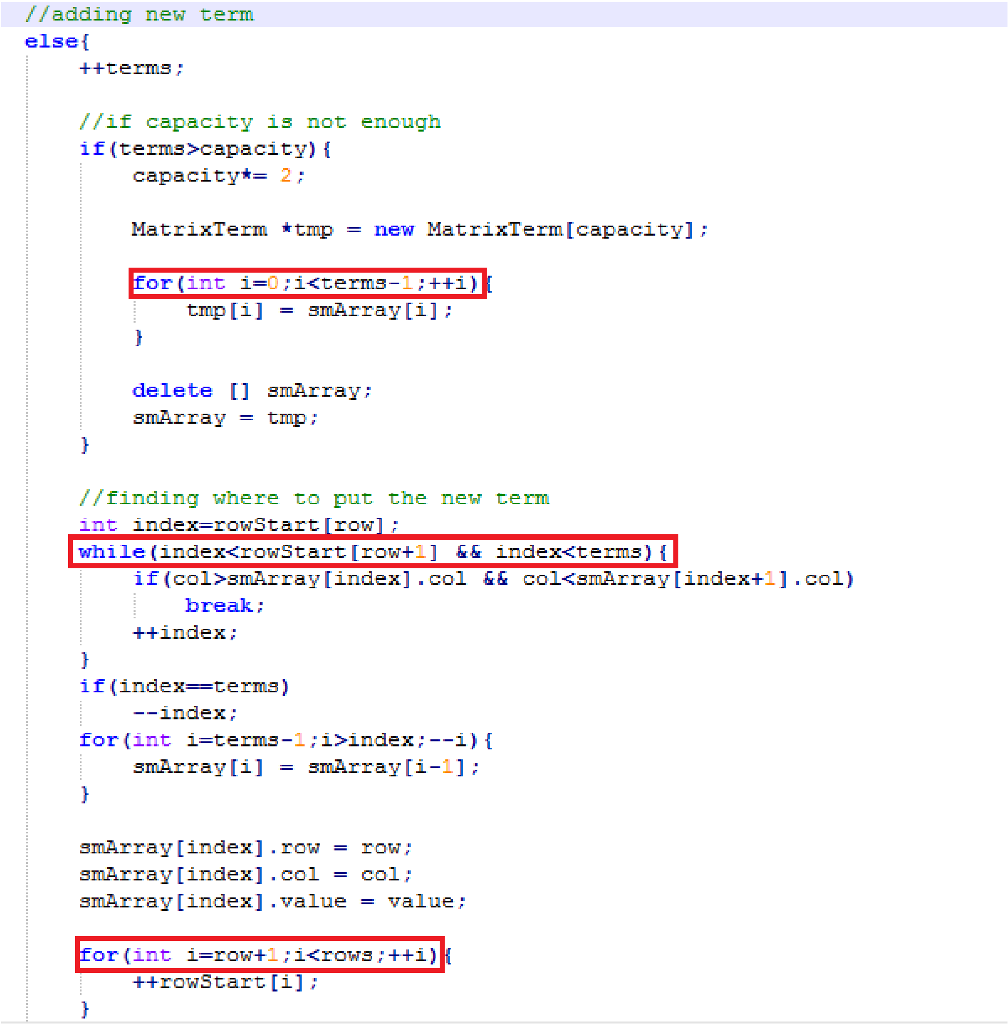
  time complexity →**O(max(row, col))+O(max(col, row)) = O(max(col, row))**



```
//find the right row
node *findrow = head->down;
int i;
for(i=0;i<row;++i){
    findrow = findrow -> down;
}

node *tar;
for(tar= findrow;tar->right!=nullptr;tar= tar->right){
    if(tar->right->col >col)
        break;
    else if(tar->right->col==col){
        //delete a element
        if(value==0){
            node *tmp = tar->right;
            tar->right = tmp->right;
            delete tmp;
            return;
        }

        tar->right->value = value;
        return;
    }
}
//find the right col
node *findcol = head->right;
for(int j=0;j<col;++j){
    findcol = findcol -> right;
}
node *k;
for(k=findcol;k->down!=nullptr;k=k->down){
    if(k->down->row > row)
        break;
}
```

Compare with last assignment → **O(terms)**

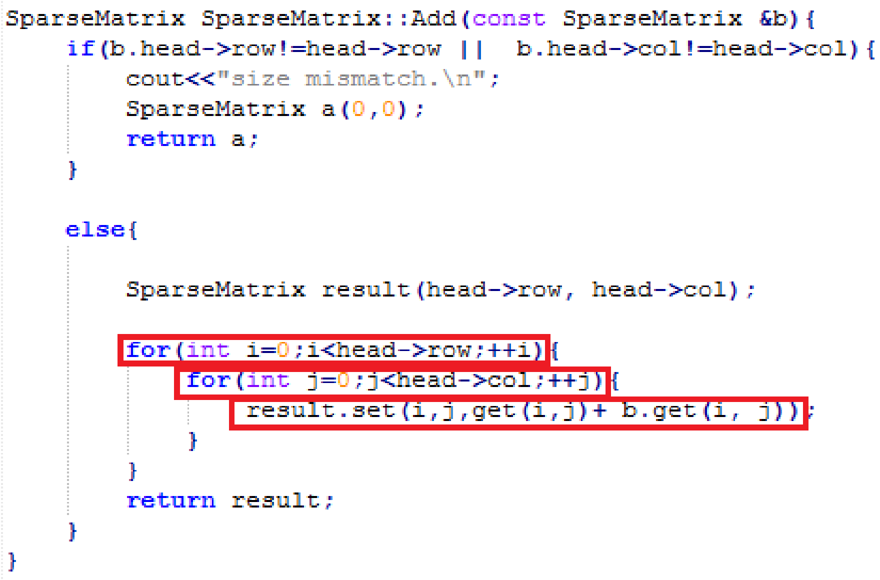
當加入陣列的值很多的時候，反倒是這次作業的 operation 比較有效率



```
//adding new term
else{
    ++terms;

    //if capacity is not enough
    if(terms>capacity){
        capacity*= 2;

        MatrixTerm *tmp = new MatrixTerm[capacity];

        for(int i=0;i<terms-1;++i){
            tmp[i] = smArray[i];
        }

        delete [] smArray;
        smArray = tmp;
    }

    //finding where to put the new term
    int index=rowStart[row];
    while(index<rowStart[row+1] && index<terms){
        if(col>smArray[index].col && col<smArray[index+1].col)
            break;
        ++index;
    }
    if(index==terms)
        --index;
    for(int i=terms-1;i>index;--i){
        smArray[i] = smArray[i-1];
    }

    smArray[index].row = row;
    smArray[index].col = col;
    smArray[index].value = value;

    for(int i=row+1;i<rows;++i){
        ++rowStart[i];
    }
```

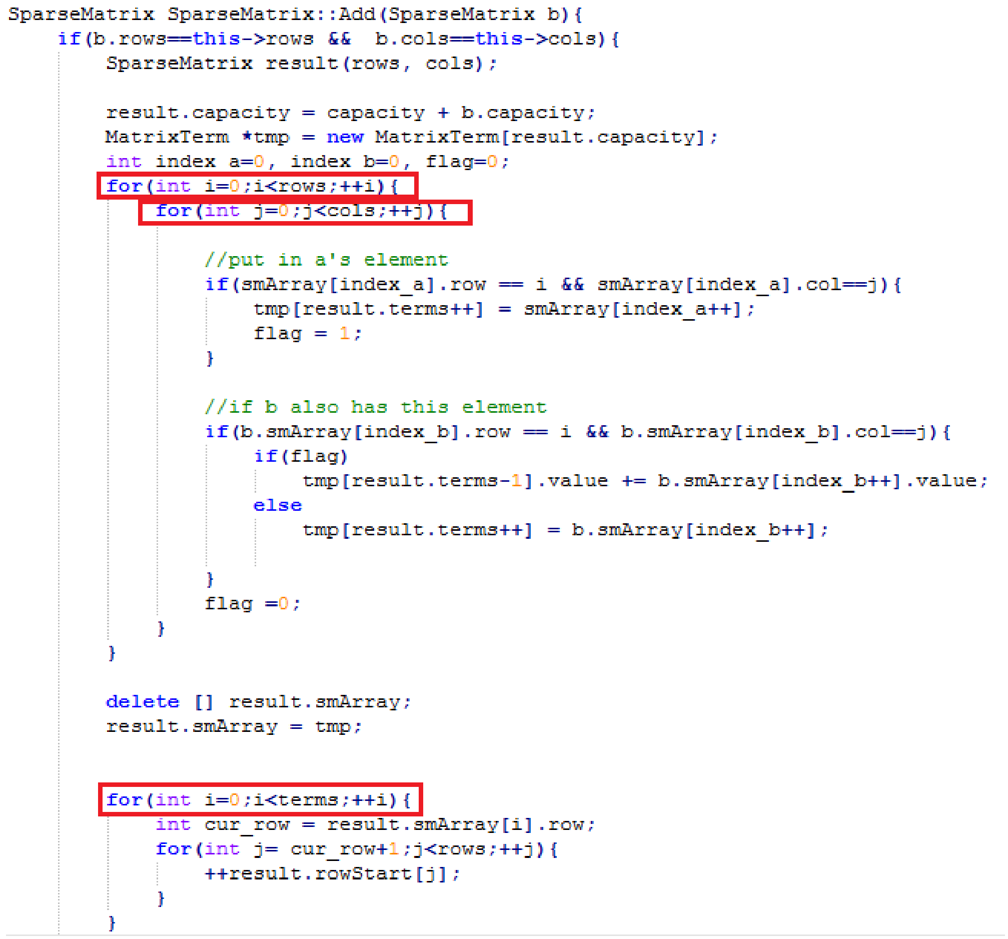
- SpraseMatirx **Add** (const SparseMatirx &b)

  time complexity →**O(row * col * (max(col, row))^2)**



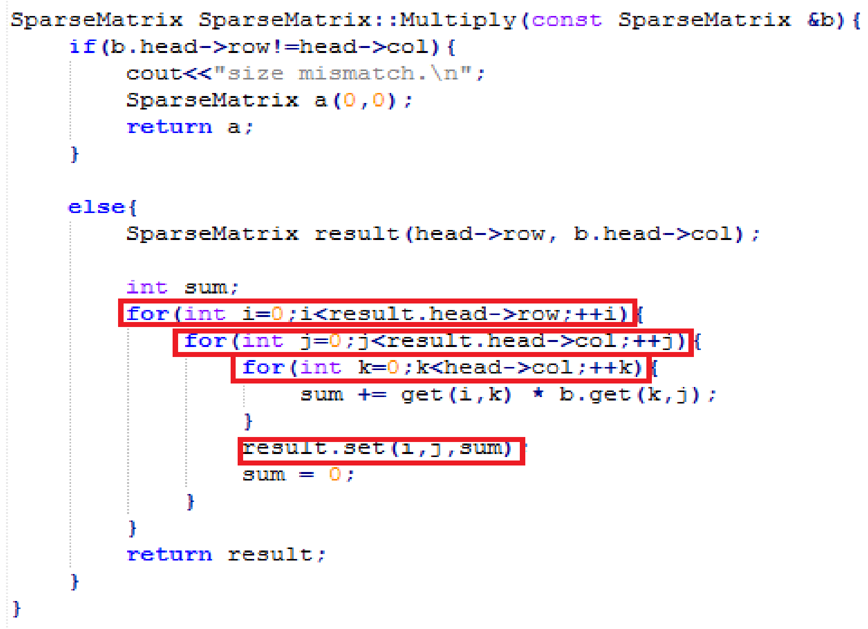
```
SparseMatrix SparseMatrix::Add(const SparseMatrix &b){
    if(b.head->row!=head->row ||  b.head->col!=head->col){
        cout<<"size mismatch.\n";
        SparseMatrix a(0,0);
        return a;
    }

    else{

        SparseMatrix result(head->row, head->col);

        for(int i=0;i<head->row;++i){
            for(int j=0;j<head->col;++j){
                result.set(i,j,get(i,j)+ b.get(i, j));
            }
        }
        return result;
    }
}
```

Compare with last assignment → **O(rows*cols) + O(terms) = O(row*col)**
兩次 add function 的實作方式不同，這次有呼叫 get & set function，因此這次的時間複雜度較大



```
SparseMatrix SparseMatrix::Add(SparseMatrix b){
    if(b.rows==this->rows &&  b.cols==this->cols){
        SparseMatrix result(rows, cols);

        result.capacity = capacity + b.capacity;
        MatrixTerm *tmp = new MatrixTerm[result.capacity];
        int index_a=0, index_b=0, flag=0;
        for(int i=0;i<rows;++i){
            for(int j=0;j<cols;++j){

                //put in a's element
                if(smArray[index_a].row == i && smArray[index_a].col==j){
                    tmp[result.terms++] = smArray[index_a++];
                    flag = 1;
                }

                //if b also has this element
                if(b.smArray[index_b].row == i && b.smArray[index_b].col==j){
                    if(flag)
                        tmp[result.terms-1].value += b.smArray[index_b++].value;
                    else
                        tmp[result.terms++] = b.smArray[index_b++];

                }
                flag =0;
            }
        }

        delete [] result.smArray;
        result.smArray = tmp;


        for(int i=0;i<terms;++i){
            int cur_row = result.smArray[i].row;
            for(int j= cur_row+1;j<rows;++j){
                ++result.rowStart[j];
            }
        }
```

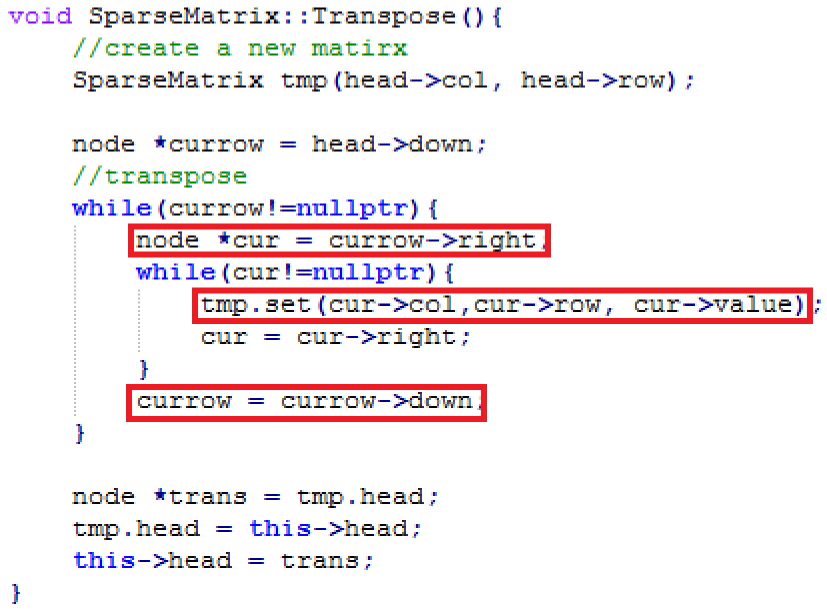
- SparseMatrix **Multiply** (const SparseMatrix &b)
  time complexity →**O(row * col^2 *(max(col, row))^2)**



```
SparseMatrix SparseMatrix::Multiply(const SparseMatrix &b){
    if(b.head->row!=head->col){
        cout<<"size mismatch.\n";
        SparseMatrix a(0,0);
        return a;
    }

    else{
        SparseMatrix result(head->row, b.head->col);

        int sum;
        for(int i=0;i<result.head->row;++i){
            for(int j=0;j<result.head->col;++j){
                for(int k=0;k<head->col;++k){
                    sum += get(i,k) * b.get(k,j);
                }
                result.set(i,j,sum);
                sum = 0;
            }
        }
        return result;
    }
}
```

- void **Transpose**()

  time complexity →**O(row * col *(max(col, row)))**



```
void SparseMatrix::Transpose(){
    //create a new matirx
    SparseMatrix tmp(head->col, head->row);

    node *currow = head->down;
    //transpose
    while(currow!=nullptr){
        node *cur = currow->right;
        while(cur!=nullptr){
            tmp.set(cur->col,cur->row, cur->value);
            cur = cur->right;
        }
        currow = currow->down;
    }

    node *trans = tmp.head;
    tmp.head = this->head;
    this->head = trans;
}
```

Part 2

- set function 的 time complexity →

  從 **O(max(row, col))+O(max(col, row))= O(max(row, col))**

  變成 **O(max(row, col))**

  get function 的 time complexity →從 **O(max(row, col))** 變成 **O(col)**

  其餘的 function 若有使用 get function 的其複雜度會因此降低

- 使用 array 版的 header nodes 可以直接取的需要用的 row 或 col，較 linked list 直觀，比較容易運用，也可以降低 time complexity。