

ASSESSMENT COVER SHEET

Student Name	Shane Reynolds
Student ID	262538
Assessment Title	Assignment 3
Unit Number and Title	HIT365
Lecturer/Tutor	Kai Wang
Date Submitted	24/5/2014
Date Received	24/5/2014

KEEP A COPY

Please be sure to make a copy of your work. If you have submitted assessment work electronically make sure you have a backup copy.

PLAGIARISM

Plagiarism is the presentation of the work of another without acknowledgement. Students may use a limited amount of information and ideas expressed by others but this use must be identified by appropriate referencing.

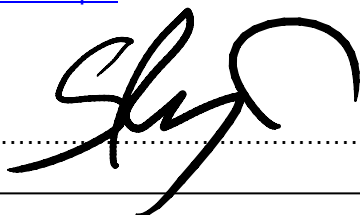
CONSEQUENCES OF PLAGIARISM

Plagiarism is misconduct as defined under the Student Conduct By-Laws. The penalties associated with plagiarism are designed to impose sanctions on offenders that reflect the seriousness of the University's commitment to academic integrity.

I declare that all material in this assessment is my own work except where there is a clear acknowledgement and reference to the work of others. I have read the University's Academic and Scientific Misconduct Policy and understand its implications.*

<http://www.cdu.edu.au/governance/documents/AcademicandScientificMisconductPolicyv1.03Jan2011.pdf>.

Signed.....



Date.....

13/4/2014

QUESTION 1 (20 MARKS)

Using Assignment 2. Insure that your code can read in a graph successfully and that it can ignores edge if the same edge is recorded twice in the input file and report an error to stderr.

Write new test functions to test that your program handles this case.

```
/*
 * Question 1.c
 * Created on: 24/05/2014
 * Author: Shane
 */

#define LEN 4
#include <stdio.h>

int duplicate(int array[LEN][LEN]);
void testFunc(int singleValue);

int main(void){

    int matrix[LEN][LEN] = {{1, 1, 1, 1},
                             {1, 1, 0, 0},
                             {1, 0, 1, 0},
                             {1, 0, 0, 1}};

    int sentinel = duplicate(matrix);

    if (sentinel == 0){
        printf("Error: an edge was recorded twice.\n\n");
    }

    /* Run test function */
    testFunc(sentinel);

    return 0;
}

int duplicate(int array[LEN][LEN]){
    int sentinel = 1;
    int i, j;

    /* First for loop iterates through the rows */
    for (i = 0; i < 4; i++){
        /* Second for loop iterates through the columns */
        for (j = i; j < 4; j++){
            /* If statement checks for duplication */
            /* Duplication checking doesn't occur */
            if ((array[i][j] == 1) &&
                (i != j) &&
                (array[i][j] == array[j][i])){

                printf("Duplication with edges: %d-%d and %d-%d\n", i,j,j,i);
                sentinel = 0;
            }
        }
    }
}
```

```

    }

}

}

printf("\n");
/* Sentinel is returned in case an indicator is needed for duplication
 * in the matrix when we return to the main function */
return sentinel;

}

void testFunc(int singleValue){

    /*****
    * So the matrix being tested is
    *
    * 1 1 1 0
    * 1 1 0 0
    * 1 0 1 0
    * 1 0 0 1
    *
    * This matrix has duplicates
    * Duplicates = 0
    * No Duplicates = 1
    *****/

    if (singleValue == 0){
        printf("Pass\n");
    } else {
        printf("Fail\n");
    }

}

}

```

QUESTION 2 (20 MARKS)

Write a function that will allow you to delete a vertex a second function that allows you to undo this delete operation. Include two test function.

Response

```

/*
 * Question2.c
 * Created on: 24/05/2014
 * Author: Shane
 */

/* Preprocessing directives */
#include <stdio.h>
#include <stdlib.h>

/* Function prototypes */
void printMat(int **array, int nrow, int ncol);
int **delete(int **array, int nrow, int ncol, int v1);
void test(int **array, int nrow, int ncol);

```

```

int main(void){

    /*****

    /* This and the real results in the test code are the only
     * parts of the code that need changing - everything else is
     * dynamically allocated */

    int data[6][6] =    {    {0,1,1,0,1,1},
                                {1,0,1,1,1,0},
                                {1,1,0,1,1,0},
                                {0,1,1,0,1,1},
                                {1,1,1,1,0,1},
                                {1,0,0,1,1,0}    };

    /*****/

    /* Variables for main function specified */
    int **graphOld;
    int i,j;
    int vertex1;
    int nrow = sizeof(data) / sizeof(data[0]);
    int ncol = nrow;

    /* Dynamically allocated memory to store the existing adjacency matrix.
     * This data structure is a array of pointers which make up the rows.
     * Each pointer in the array of pointers each point to an array of
     * pointers which makes up the dynamic 2D array */
    graphOld = calloc(nrow, sizeof(int*));

    for (i = 0; i < nrow; i++){
        graphOld[i] = calloc(ncol, sizeof(int));
    }

    /* Populate our dynamically allocated 2D array with the actual
     * adjacency matrix */
    for (i = 0; i < nrow; i++){
        for (j = 0; j < ncol; j++){
            graphOld[i][j] = data[i][j];
        }
    }

    /* Print the original adjacency matrix */
    printf("\nOriginal adjacency matrix:\n\n");
    printMat(graphOld, nrow, ncol);

    /* The edge to be contracted can be thought of as two vertices -
     * that is the edge can be defined by two vertices. This code
     * prompts the user for the two vertices which make up the
     * edge they want to contract */
    printf("Enter the number of the vertex to be deleted: ");
    scanf("%d", &vertex1);

    printf("\n");

    /* Call to the function which contracts the graph. The result is
     * returned to an pointer which is initialised in the line */
    int **graphPtr = delete(graphOld, nrow, ncol, vertex1);

    /* Print the contracted graph */
    printf("Adjacency matrix with deleted vertex:\n\n");

```

```

    printMat(graphPtr, nrow-1, ncol-1);

    /* A test function to ensure that the correct results are
       * achieved */
    test(graphPtr, nrow, ncol);

    return 0;
}

/* This function receives a dynamically allocated 2D array and
   * prints it */
void printMat(int **array, int nrow, int ncol){
    int i, j;

    for (i = 0; i < nrow; i++){
        for (j = 0; j < ncol; j++){
            printf("%d ", array[i][j]);
        }
        printf("\n\n");
    }
}

/* This function receives a 2D array as input, along with the two
   * vertices that need to be contracted to one. It outputs a pointer
   * to an array. */
int **delete(int **array, int nrow, int ncol, int v1){

    /* specify and initialise variables */
    int **graphNew;
    int i, j;
    int m = 0;
    int n = 0;

    /* Create a place in the heap memory (initialised with zeros)
       * for the contracted graph to be stored */
    graphNew = calloc(nrow-1, sizeof(int*));

    for (i = 0; i < nrow-1; i++){
        graphNew[i] = calloc(ncol-1, sizeof(int));
    }

    /* This piece of code takes the columns which aren't the contracting vertexes
       * and shifts them to the left and takes the rows which aren't the
       * contracting vertexes and shifts them up NOTE: there will be
       * a final row and final column which represents the contracted vertex. */
    for (i = 0; i < nrow; i++){
        n = 0;
        if (i != (v1-1)){
            for (j = 0; j < ncol; j++){
                if (j != (v1-1)){
                    graphNew[m][n] = array[i][j];
                    n += 1;
                }
            }
            m += 1;
        }
    }

    return graphNew;
}

```

```

}

/* Basic test which ensures that the results from the code are
 * correct */
void test(int **array, int nrow, int ncol){

    /*****/
    /* User needs to put the actual results in to a 2D array
     * for the test function to accurately test the output of
     * the removeEdge() function */

    int data[5][5] = { {0,1,0,1,1},
                        {1,0,1,1,0},
                        {0,1,0,1,1},
                        {1,1,1,0,1},
                        {1,0,1,1,0} };

    /*****/

    int i,j;
    int pass = 1;

    for (i = 0; i < (nrow-2); i++){
        for (j = 0; j < (ncol-2); j++){
            if (array[i][j] != data[i][j]){
                pass = 0;
                break;
            }
        }
    }

    if (pass == 1){
        printf("Test passed.\n");
    } else {
        printf("Test failed.\n");
    }
}

```

QUESTION 3 (20 MARKS)

Consider using three data structures. A matrix representation of the graph, a list representation of the graph and an array that holds the degree of each vertex. Can you improve the performance of your program explain.

Response

The performance of the code cannot be increased for the duplication detection function in question 1 as it is already at its optimum. Using the matrix representation of the graph allows the symmetry of the data structure to be exploited when finding duplicates. The list and array data structures would require you to look at an entry in the list (or array) and then linearly iterate through the list (or array) to try and find a duplicate – this requires many more operations than the simple if statement used with the matrix data structure.

The performance of the code would not be improved by using a list or array structure – it would be the same no matter which data structure you used. One entry on the matrix represents an edge (or vertex pair). Each edge has to be visited to determine if the edge contains the vertex being deleted. This is true for each of the data structures.