\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Report: HW7

Author: F74042086郭泰佑 <st9540808@yhaoo.com.tw>

Class: 乙班

Description:

How do you finish this homework?

Textbook & stackoverflow

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Code:

//學號：F74042086

//姓名：郭泰佑

//編譯方式：gcc -std=c99 -o hw7 hw7.c

//執行方式：./hw7

//程式功能：generate buckets.out mapping.out

#include <stdio.h>

#include <stdlib.h>

int rows = 0, cols = 0, size = 1, n, BucketIndex;

int bucket\_union(int \*\*, int []);

int new\_mapping(int \*\*, int [rows][cols], int[rows][n + 1], int);

void print\_result(FILE \*, FILE \*, int \*\*, int[rows][n + 1]);

int \*\*dynamic\_alloc\_2darray();//generate a "1 \* n\_cols" 2D array "enter by scanf"

int \*\*add\_1\_row(int\*\*);//add a row to 2D array

void scanfile(FILE \*);//read the maximum cols needed and total rows

int count\_element(int n , int []);//count element for an array

int main()

{

int i = 0, j = 0, index ;

char ch;

FILE \*fp, \*fp\_buckets, \*fp\_BitMap;

fp = fopen("buckets.in","r");

fp\_buckets = fopen("buckets.out","w+");

fp\_BitMap = fopen("mapping.out","w+");

if(fp == NULL){

printf("file cannot open");

exit(1);

}

scanfile(fp);//determine the size of DirectMapping array

rewind(fp);

while(1){scanf("%d", &n); if(n >= cols) break;}//iput a number to n (must >= maximum number of element in )

int DirectMapping[rows][cols],

BitMap[rows][n + 1],\*p,

\*\*NewMapping = dynamic\_alloc\_2darray();//later be input by scanf

for(p = &DirectMapping[0][0]; p <= &DirectMapping[rows - 1][cols - 1]; p++)

\*p = 0;//initalize to 0

for(p = &BitMap[0][0]; p <= &BitMap[rows - 1][(n + 1) - 1]; p++)

\*p = 0;//initalize to 0

//store data into two dimensional array

while( fscanf(fp,"%d%c",&DirectMapping[i][j], &ch) != EOF){

j++;

if(ch == '\r'){j = 0;i++;}//change to '\r'for file generated in windows

}

for(index = 0; index < rows; index++)

{

int current,

storage\_left, //number of 0 in a certain row of NewMapping

element\_ori = count\_element(cols, DirectMapping[index]),//element of bucket

element = element\_ori, mark[element];

for(int k = 0; k < element; k++) mark[k] = 0;

switch(bucket\_union(NewMapping,DirectMapping[index]))

{

case 0:

for(i = 0; i < size; i++)

{

storage\_left = n - count\_element(n, NewMapping[i]);

if( storage\_left == 0 ) continue;

for(current = 0; current < cols; current++)

{

if( DirectMapping[index][current] == 0 ) break;

for(j = 0; j < n; j++)

if( NewMapping[i][j] == DirectMapping[index][current] && NewMapping[i][j] != 0 )

{ element-- ; mark[current] = 1; break;}

}

//here indicate bucket can union with NewMapping[i]

if( element <= storage\_left )

{

for(current = 0; current < element\_ori; current++)

{

if( mark[current] == 0)

{

for(j = 0; j < n; j++)

{

if( NewMapping[i][j] == 0)

{

NewMapping[i][j] = DirectMapping[index][current];

break;

}

}

}

}

}

}

if( element <= storage\_left )

{

BucketIndex = i - 1;

new\_mapping(NewMapping, DirectMapping, BitMap, index);

break;

}

NewMapping = add\_1\_row(NewMapping);

for(j = 0; j < cols; j++)

if(DirectMapping[index][j] != 0) NewMapping[i][j] = DirectMapping[index][j];

BucketIndex = i;

new\_mapping(NewMapping, DirectMapping, BitMap, index);

size++;

break;

case 1:

new\_mapping(NewMapping, DirectMapping, BitMap, index);

break;

}//switch ends here

}//loop ends here

print\_result(fp\_buckets, fp\_BitMap, NewMapping, BitMap);

fclose(fp);

fclose(fp\_buckets);

fclose(fp\_BitMap);

for(i = 0; i < size; i++)

free(NewMapping[i]);

free(NewMapping);

return 0;

}

int bucket\_union(int \*\*NewMapping,int bucket[])

{

int i, j, current, flag = 0;

for(i = 0; i < size; i++)

{

if( count\_element(cols, bucket) > count\_element(n , NewMapping[i]) )

continue;

for(current = 0; current < cols; current++)

{

if( bucket[current] == 0 && flag == 1 ) break;

flag = 0;

for(j = 0; j < n; j++)

if( NewMapping[i][j] == bucket[current] ){flag = 1; break;}

if( flag == 0 ) break;

}

if( flag == 1 ){BucketIndex = i; break;}

}

return flag;

}

int new\_mapping(int \*\*NewMapping, int DirectMapping[rows][cols], int BitMap[rows][n + 1], int index)

{

int i, j;

BitMap[index][0] = BucketIndex;

for(i = 0; i < n; i++)

{

for( j = 0; j < n; j++)

{

if( DirectMapping[index][i] == NewMapping[BucketIndex][j] && DirectMapping[index][i] != 0)

{

BitMap[index][j + 1] = 1;

break;

}

}

}

return 0;

}

void print\_result(FILE \*fp\_buckets, FILE \*fp\_BitMap, int \*\*NewMapping, int BitMap[rows][n + 1])

{

int i, j;

for(i = 0; i < size; i++)

{

for(j = 0; j < n; j++)

{

if ( NewMapping[i][j] != 0)

fprintf(fp\_buckets, "%d ", NewMapping[i][j]);

}

fprintf(fp\_buckets ,"\n");

}

for(i = 0; i < rows; i++)

{

for(j = 0; j < n + 1; j++)

{

fprintf(fp\_BitMap ,"%d", BitMap[i][j]);

if(j < 1)

fprintf(fp\_BitMap," ");

}

fprintf(fp\_BitMap ,"\n");

}

}

int \*\* dynamic\_alloc\_2darray()

{

int \*\*array;

array = calloc(1 , sizeof(int \*));

if(array == NULL){

printf("out of memory(1)\n");

exit(1);

}

array[0] = calloc(n , sizeof(int));

if(array[0] == NULL){

printf("out of memory(2)\n");

exit(1);

}

return array;

}

int \*\* add\_1\_row(int \*\* array)

{

array = realloc(array, (size + 1) \* sizeof(int \*));

array[size] = calloc(n, sizeof(int));

if(array[size] == NULL){

printf("out of memory(3)\n");

exit(1);

}

return array;

}

void scanfile(FILE \*fp)

{

int count\_cols = 0;

char ch;

while( (ch = fgetc(fp)) != EOF)

{

if(ch == ' ')count\_cols++;

if(ch == '\n')

{

rows++;

if(count\_cols > cols)//cols here means maximum cols to store D.M

cols = count\_cols;

count\_cols = 0;

}

} rows++; //delete it on linux

cols++;

}

int count\_element(int element,int array[])

{

int i;

for(i = 0; i < element; i++)

if( array[i] == 0 ) break;

return i;

}

Compilation:

gcc -std=c99 -o hw7 hw7.c

Execution:

./hw7

Output:

F74042086@c-2015-1:~/hw7> ./hw7

38

F74042086@c-2015-1:~/hw7> ./hw7\_checker

no error