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

Report: HW5

Author: E64061151 林友鈞 <[E64061151@mail.ncku.edu.tw](mailto:wang@xiaoming.tw)>

Class: 資訊系二乙

Description:

這份作業讓我學到如何壓縮，以及陣列、指標、動態記憶體的使用。

以下將解釋每個function的功能。

void Zeroing(int \*array,int size):

這個函數會將陣列歸零，需要傳入的是一個陣列，與陣列的大小。

雖然可以用int a[5] = {0};的方式歸零，但是因為程式裡使用到的陣列幾乎都使動態取得的記憶體，所以只好用Zeroing函數來歸零。

int main(int argc, char const \*argv[]):

int \*\* bucket：

是一個二維指標，會先用malloc幫他取得一塊大小為n的一維指標陣列。

接下來用for 與 amounts\_of\_elements = rand()%m + 1去決定bucket中每一個元素需要的記憶體空間，需要的空間會去malloc一塊 amounts\_of\_elements + 1的記憶體。而這塊記憶體的第一個元素放的值是 amounts\_of\_elements。並利用for一一填入亂數。

int \*\* newMapping：

也是一個二維指標，儲存壓縮後的結果。

在main中呼叫Compress去做壓縮

int \*\* Compress(int \*\*original\_bucket, int \*\*newMapping, int n, int Max);

Compress函數主要功能是去做壓縮。

Parameter:

int \*\* original\_bucket是一個二維陣列，儲存的是在main中產生的bucket。

int \*\* newMapping 也是一個二維陣列，儲存的是newMapping的結果。

int n 是從command line 輸入的n。

int Max是newMapping每個row的size最多可以為多少。

一開始會宣告int newMappingRows(紀錄newMapping有幾個row), char \*\*\* bucket\_index。它是一個三維指標，儲存的是bucket壓縮後的row index 與 bit pattern，都是以字串方式儲存，所以需要三維指標。

接下來會對 bucket\_index做初始化，bucket\_index需要malloc n個char \*\*形態的空間。接下來會對bucket\_index中每個element malloc 2個char \*形態的空間。而接下來會對row中第一個元素malloc 5個char型態的記憶體(未來儲存的會是每個original\_bucket的row被壓在newMapping的那個row)，而第二個元素則會被malloc Max + 2(因為還須考慮字串字尾是’\0’)個 char 型態的空間。

程式邏輯：

每次從original中取一個row出來，與目前newMapping中所有row做判斷(使用bucket\_union)如果能成功merge進newMapping，bucket\_union會回傳最適合merge的row，在Compress就會依照bucket\_union回傳的row做merge，並記錄bucket\_index。如果在bucket\_union找不到任何一個row能夠merge，則會回傳-1，如果不能merge，在Compress就會將newMappingRows + 1(本來有寫realloc newMapping，但是不知道為何OS不讓我這樣做)，並直接將original\_bucket的row直接merge進新的row。

original\_bucket中所有的row都判斷完後，就會呼叫fprint\_result將結果輸出。

int bucket\_union(int newMappingRows, int \* original\_bucket, int \*\* newMapping, char \*\* bucket\_index, int Max)

這個函數是會判斷newMapping中哪個row是最適合merge的，如果有則回傳row的編號，沒有則回傳-1。

Parameter:

int newMappingRows: newMapping中有多少個Row。

int \*original\_bucket: 一個需要被判斷merge的row。

int \*\* newMapping: 二維陣列newMapping。

char \*\* bucket\_index: 會去紀錄merge後的bit pattern是如何。

int Max : 是newMapping每個row的size最多可以為多少。

這個函數會Scan newMapping中所有的row，max\_match與max\_match\_row會去紀錄最多match到的數目與最多match 到的row是哪一行。 如果沒有適合的Row可以match就會回傳-1，否則回傳max\_match\_row。

int Merge(int \* original\_bucket, int \* newMapping , char \* bucket\_index, int Max, int \* newMapping0):

Merge 會去merge original\_bucket與 newMapping，並將merge結果填入bucket\_index中。

Parameter:

int \* original\_bucket: 一個original\_bucket中需要被merge的row。

int \* newMapping: 一個newMapping中需要被merge的row。

int Max : 是newMapping每個row的size最多可以為多少。

int \*newMapping0 : 傳入的是newMapping一個row中第一個元素(記錄著這個row中已經merge進來的數字)

這個函數會比較original\_bucket中所有數字，判斷newMapping中是否有數字能match，如果newMapping都scan完，original\_bucket的數字仍沒有match到，就會被記錄下來。如果能夠成功成功match，那bucket\_index就在他對應到的位置填入1紀錄這個結果，並將result+=2。

最後判斷newMapping剩下的位置能否放得下original\_bucket中沒有match到的數字。如果能就放進去，bucket\_index做紀錄，result++，最後回傳result的結果，不能則return -1。

所以Merge最後回傳的只有-1或是不是-1的數，是-1代表merge失敗，不是-1則代表match的品質，數值愈高代表品質越好代表越多數字能merge入原本newMapping的row中。以這個標準來讓bucket\_union來判斷哪個row應該被merge入。

void fprint\_result(int newMappingRows, int \*\*newMapping, char \*\*\*bucket\_index, int \*\* original\_bucket,int n):

這個函數是去將結果輸出至file中。

int newMappingRows: newMapping中有多少個Row。

int \*\* newMapping: 二維陣列newMapping。

char \*\*\*bucket\_index：

紀錄的是original\_bucket merge的結果，每個row分別為row/bit pattern。

int \*\* original\_bucket是一個二維陣列，儲存的是在main中產生的bucket。

int n 是從command line 輸入的n。

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

Code:

#include<stdio.h>

#include<stdlib.h>

#define EOB -1

void fprint\_result(int newMappingRow,int \*\*newMapping,char \*\*\* bucket\_index,int \*\*original\_bucket,int n);

void Zeroing(int \*array,int size){

for(int i = 0; i < size; i++) array[i] = 0;

}

int Merge(int \* original\_bucket, int \* newMapping, char \* bucket\_index,int Max,int \*newMapping0){

int map\_match[newMapping[0] + 1];

int no\_match[original\_bucket[0]+1];

int count = 0,i,j, result = 0;

Zeroing(no\_match,original\_bucket[0] + 1);

Zeroing(map\_match,newMapping[0] + 1);

for(int i = 0; i < Max + 1; i++) bucket\_index[i] = '0';// initialize the string;

bucket\_index[Max+1] = '\0'; // set the end of bucket\_index, because bucket\_index is string type data;

// search begin

for(i = 1; i <= original\_bucket[0]; i++){

for(j = 1; j <= \*newMapping0; j++){

if(!map\_match[j] && newMapping[j] == original\_bucket[i]){

map\_match[j] = 1;

bucket\_index[j] = '1';

result += 2;

break;

}

}

if(j > \*newMapping0){

no\_match[count] = original\_bucket[i];

++count;

}

}

if(Max - \*newMapping0 < count) return EOB;

for(i = 0; i < count; i++){

newMapping[\*newMapping0 + 1] = no\_match[i];

bucket\_index[\*newMapping0 + 1] = '1';

(\*newMapping0)++;

result ++;

}

return result;

}

int bucket\_union(int newMappingRows,int \*original\_bucket,int \*\* newMapping,char \*\* bucket\_index, int Max){

int original\_match = 0;

int temp;

int max\_match = -1, max\_match\_row = 0;

for(int i = 0; i < newMappingRows; i++){

original\_match = newMapping[i][0];

temp = Merge(original\_bucket,newMapping[i],bucket\_index[1],Max,&original\_match);

max\_match = temp > max\_match ? max\_match\_row = i,temp : max\_match;

}

return max\_match == EOB ? max\_match : max\_match\_row;

}

int \*\* Compress(int \*\* original\_bucket, int \*\* newMapping, int n,int Max){

int newMappingRows = 0;

int max\_match,max\_match\_row;

int i,j;

char \*\*\* bucket\_index = (char \*\*\*)malloc(n\*sizeof(char \*\*));

    newMapping = malloc(102\*sizeof(int \*\*));

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

bucket\_index[i] = (char \*\*)malloc(2\*sizeof(char \*));

bucket\_index[i][0] = (char \*)malloc(5\*sizeof(char));

bucket\_index[i][1] = (char \*)malloc((Max + 2)\*sizeof(char));

}

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

max\_match = EOB;

max\_match = bucket\_union(newMappingRows,original\_bucket[i],newMapping,bucket\_index[i],Max);

if(max\_match == EOB){

newMapping[newMappingRows] = (int \*)malloc((Max + 1) \* sizeof(int));

newMapping[newMappingRows][0] = 0;

Merge(original\_bucket[i],newMapping[newMappingRows],bucket\_index[i][1],Max,&newMapping[newMappingRows][0]);

sprintf(bucket\_index[i][0],"%d",newMappingRows);

newMappingRows ++;

}else{

max\_match\_row = max\_match;

Merge(original\_bucket[i],newMapping[max\_match\_row],bucket\_index[i][1],Max,&newMapping[max\_match\_row][0]);

sprintf(bucket\_index[i][0],"%d",max\_match\_row);

}

}

fprint\_result(newMappingRows,newMapping,bucket\_index,original\_bucket,n);

return newMapping;

}

void fprint\_result(int newMappingRow,int \*\*newMapping,char \*\*\* bucket\_index,int \*\*original\_bucket,int n){

FILE \* f\_out = NULL;

f\_out = fopen("hw5.out","w");

if(f\_out){

fprintf(f\_out,"%3d\n\n",newMappingRow);

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

fprintf(f\_out,"%3s /",bucket\_index[i][0]);

for(int j = 1; j <= original\_bucket[i][0]; j++){

fprintf(f\_out,"%3d",original\_bucket[i][j]);

}

fprintf(f\_out,"\n");

}

fprintf(f\_out,"\n");

for(int i = 0; i < newMappingRow; i++){

fprintf(f\_out,"%3d /",i);

for(int j = 1; j <= newMapping[i][0]; j++)

fprintf(f\_out,"%3d",newMapping[i][j]);

fprintf(f\_out,"\n");

}

}else{

printf("Cannot open hw5.out");

exit(0);

}

}

int main(int argc, char const \*argv[])

{

int n = atoi(argv[1]),m = atoi(argv[2]),d = atoi(argv[3]), s = atoi(argv[4]);

srand(s);

int amounts\_of\_elements;

int \*\*bucket = (int \*\*)malloc(n\*sizeof(int \*));

int \*\* newMapping;

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

amounts\_of\_elements = (rand() % m) + 1;

bucket[i] = (int \*)malloc((amounts\_of\_elements + 1)\*sizeof(int)); // malloc amount\_of\_element + 1 '1' is for first element

bucket[i][0] = amounts\_of\_elements;

for(int j = 1; j <= bucket[i][0]; j++){

bucket[i][j] = rand()%d;

}

}

// for(int i = 0; i < n; i++){

// printf("%d | ",bucket[i][0]);

// for(int j = 1;j <= bucket[i][0];j++){

// printf("%5d",bucket[i][j]);

// }

// printf("\n");

// }

Compress(bucket,newMapping,n,m);

return 0;

}

Compilation:

gcc -std=c99 -o hw5 hw5.c

Execution:

./hw5 n m d s

Output:

./hw5 10 10 10 10

3

0 / 8 8 5 8 7 5

0 / 7 7

0 / 3 8 3 4

1 / 0 8

0 / 8 8

1 / 5 1 2 8 2

1 / 8 6 7 3

2 / 8 0 5 5 5 9 2 3

0 / 5 3 7

1 / 6 8

0 / 8 8 5 8 7 5 7 3 3 4

1 / 0 8 5 1 2 2 6 7 3

2 / 8 0 5 5 5 9 2 3