**자료구조**

**정보대학 컴퓨터학과**

**2015410100**

**임수경**

**-source code**

\*Hash.h

#ifndef Hash\_h

#define Hash\_h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef char\* KeyType;

typedef int ValueType;

enum ElementStatus

{

EMPTY = 0,

OCCUPIED = 1

};

typedef struct tagElementType

{

KeyType Key;

ValueType Value;

enum ElementStatus Status;

} ElementType;

typedef struct tagHashTable

{

int OccupiedCount;

int TableSize;

ElementType\* Table;

} HashTable;

HashTable\* OAHT\_CreateHashTable( int TableSize );

void OAHT\_DestroyHashTable( HashTable\* HT );

void OAHT\_ClearElement( ElementType\* Element );

void OAHT\_Set( HashTable\*\* HT, KeyType Key, ValueType Value );

ValueType OAHT\_Get( HashTable\* HT, KeyType Key );

int OAHT\_Hash ( KeyType Key, int KeyLength, int TableSize );

int OAHT\_Hash2( KeyType Key, int KeyLength, int TableSize );

#endif

\*Heap.h

#ifndef Heap\_h

#define Heap\_h

#include <stdio.h>

#include <stdlib.h>

#include "RBTree.h"

#define TRUE 1

#define FALSE 0

#define HEAP\_LEN 100

typedef int PriorityComp(void\* left, void\* right);

typedef struct \_Heap

{

PriorityComp \* comp;

int numOfData;

void\*\* pHeapArr;

} Heap;

void HeapInit(Heap \* ph, int size, PriorityComp pc);

int HIsEmpty(Heap \* ph);

void HInsert(Heap \* ph, void\* data);

void\* HDelete(Heap \* ph);

void HeapDestroy(Heap \* ph);

#endif

\*PQueue.h

#ifndef PQueue\_h

#define PQueue\_h

#include <stdio.h>

#include "Heap.h"

typedef Heap PQueue;

void PQueueInit(PQueue \* ppq, int size, PriorityComp pc);

int PQIsEmpty(PQueue \* ppq);

void PEnqueue(PQueue \* ppq, void\* pData);

void\* PDequeue(PQueue \* ppq);

void PQueueDestroy(PQueue \* ppq);

#endif

\*RBTree.h

#ifndef RBT\_H

#define RBT\_H

#include <stdio.h>

#include <stdlib.h>

typedef struct \_Node{

struct \_Node \*pParent;

struct \_Node \*pLeft;

struct \_Node \*pRight;

enum {Red,Black} Color;

//노드 식별코드와 추가 데이터

int iID;

void \*pData;

} Node;

//레드블랙트리 관련 함수

void RBT\_DestroyTree(Node \*pTree);

Node\* RBT\_CreateNode(int iID,void \*pData);

void RBT\_DestroyNode(Node \*pNode);

Node\* RBT\_SearchNode(Node \*pTree, int iTargetID);

Node\* RBT\_SearchMinNode(Node \*pTree);

void RBT\_InsertNode(Node \*\*pTree, Node \*pNewNode);

void RBT\_InsertNodeHelper(Node \*\*pTree,Node \*pNewNode);

Node\* RBT\_RemoveNode(Node \*\*pRoot, int iID);

void RBT\_RebuildAfterInsert(Node \*\*pTree, Node \*pNewNode);

void RBT\_RebuildAfterRemove(Node \*\*pRoot, Node \*pSuccessor);

void RBT\_PrintTree(Node \*pNode, int iDepth, int iBlackCount);

void RBT\_RotateLeft(Node \*\*Root, Node \*pParent);

void RBT\_RotateRight(Node \*\*pRoot, Node \*pParent);

void RBT\_Inorder( Node \*pNode, void \*pData, void (\*fnConduct)(Node \*pNode,void \*pData) );

#endif

\*Hash.cpp

#include "Hash.h"

HashTable\* OAHT\_CreateHashTable( int TableSize )

{

HashTable\* HT = (HashTable\*)malloc( sizeof(HashTable) );

HT->Table = (ElementType\*)malloc( sizeof(ElementType) \* TableSize);

memset(HT->Table, 0, sizeof(ElementType) \* TableSize );

HT->TableSize = TableSize;

HT->OccupiedCount = 0;

return HT;

}

void OAHT\_Set( HashTable\*\* HT, KeyType Key, ValueType Value )

{

int KeyLen, Address, StepSize;

KeyLen = strlen(Key);

Address = OAHT\_Hash( Key, KeyLen, (\*HT)->TableSize );

StepSize = OAHT\_Hash2( Key, KeyLen, (\*HT)->TableSize );

while ( (\*HT)->Table[Address].Status != EMPTY &&

strcmp((\*HT)->Table[Address].Key, Key) != 0 )

{

Address = (Address + StepSize) % (\*HT)->TableSize;

}

(\*HT)->Table[Address].Key = (char\*)malloc( sizeof(char) \* (KeyLen + 1) );

strcpy( (\*HT)->Table[Address].Key, Key );

(\*HT)->Table[Address].Value = Value;

(\*HT)->Table[Address].Status = OCCUPIED;

(\*HT)->OccupiedCount++;

}

ValueType OAHT\_Get( HashTable\* HT, KeyType Key )

{

int KeyLen = strlen(Key);

int Address = OAHT\_Hash( Key, KeyLen, HT->TableSize );

int StepSize = OAHT\_Hash2( Key, KeyLen, HT->TableSize );

while ( HT->Table[Address].Status != EMPTY &&

strcmp(HT->Table[Address].Key, Key) != 0 )

{

Address = (Address + StepSize) % HT->TableSize;

}

return HT->Table[Address].Value;

}

void OAHT\_ClearElement( ElementType\* Element )

{

if ( Element->Status == EMPTY)

return;

free(Element->Key);

}

void OAHT\_DestroyHashTable( HashTable\* HT)

{

int i = 0;

for ( i=0; i<HT->TableSize; i++ )

{

OAHT\_ClearElement( &(HT->Table[i]) );

}

free ( HT->Table );

free ( HT );

}

int OAHT\_Hash( KeyType Key, int KeyLength, int TableSize )

{

int i=0;

long long HashValue = 0;

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

{

int Abs = (Key[i] > 0) ? Key[i] : -Key[i];

HashValue = (HashValue << 3) + Abs;

HashValue = HashValue % TableSize;

}

return (int)HashValue;

}

int OAHT\_Hash2( KeyType Key, int KeyLength, int TableSize )

{

int i=0;

long long HashValue = 0;

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

{

int Abs = (Key[i] > 0) ? Key[i] : -Key[i];

HashValue = (HashValue << 2) + Abs;

HashValue = HashValue % ( TableSize - 3 );

}

return (int)HashValue + 1;

}

\*Heap.cpp

#include "Heap.h"

void HeapInit(Heap \* ph, int size, PriorityComp pc)

{

ph->numOfData = 0;

ph->comp = pc;

ph->pHeapArr = (void\*\*)malloc(sizeof(void\*) \* size);

}

int HIsEmpty(Heap \* ph)

{

if(ph->numOfData == 0)

return TRUE;

else

return FALSE;

}

int GetParentIDX(int idx)

{

return idx/2;

}

int GetLChildIDX(int idx)

{

return idx\*2;

}

int GetRChildIDX(int idx)

{

return GetLChildIDX(idx)+1;

}

int GetHiPriChildIDX(Heap \* ph, int idx)

{

if(GetLChildIDX(idx) > ph->numOfData)

return 0;

else if(GetLChildIDX(idx) == ph->numOfData)

return GetLChildIDX(idx);

else

{

if(ph->comp(ph->pHeapArr[GetLChildIDX(idx)],

ph->pHeapArr[GetRChildIDX(idx)]) < 0)

return GetRChildIDX(idx);

else

return GetLChildIDX(idx);

}

}

void HInsert(Heap \* ph, void\* pData)

{

int idx = ph->numOfData+1;

while(idx != 1)

{

if(ph->comp(pData, ph->pHeapArr[GetParentIDX(idx)]) > 0)

{

ph->pHeapArr[idx] = ph->pHeapArr[GetParentIDX(idx)];

idx = GetParentIDX(idx);

}

else

{

break;

}

}

ph->pHeapArr[idx] = pData;

ph->numOfData += 1;

}

void\* HDelete(Heap \* ph)

{

void\* retData = ph->pHeapArr[1];

void\* lastElem = ph->pHeapArr[ph->numOfData];

int parentIdx = 1;

int childIdx;

while((childIdx = GetHiPriChildIDX(ph, parentIdx)))

{

if(ph->comp(lastElem, ph->pHeapArr[childIdx]) >= 0)

break;

ph->pHeapArr[parentIdx] = ph->pHeapArr[childIdx];

parentIdx = childIdx;

}

ph->pHeapArr[parentIdx] = lastElem;

ph->numOfData -= 1;

return retData;

}

void HeapDestroy(Heap\* ph){

free(ph->pHeapArr);

}

\*PQueue.cpp

#include "PQueue.h"

void PQueueInit(PQueue \* ppq,int size, PriorityComp pc)

{

HeapInit(ppq, size, pc);

}

int PQIsEmpty(PQueue \* ppq)

{

return HIsEmpty(ppq);

}

void PEnqueue(PQueue \* ppq, void\* pData)

{

HInsert(ppq, pData);

}

void\* PDequeue(PQueue \* ppq)

{

return HDelete(ppq);

}

void PQueueDestroy(PQueue \* ppq){

HeapDestroy(ppq);

}

\*RBTree.cpp

#include "RBTree.h"

extern Node \*Nil;

void RBT\_DestroyTree(Node \*pTree){

if(pTree == NULL)

return;

if( pTree->pRight != Nil )

RBT\_DestroyTree( pTree->pRight );

if( pTree->pLeft != Nil )

RBT\_DestroyTree( pTree->pLeft );

pTree->pLeft = Nil;

pTree->pRight = Nil;

RBT\_DestroyNode( pTree );

}

Node\* RBT\_CreateNode(int iID, void \*pData){

Node \*pNewNode = (Node\*)malloc(sizeof(Node));

pNewNode->pParent = NULL;

pNewNode->pLeft = NULL;

pNewNode->pRight = NULL;

pNewNode->iID = iID;

pNewNode->pData = pData;

pNewNode->Color = \_Node::Black;

return pNewNode;

}

void RBT\_DestroyNode(Node \*pNode){

free(pNode);

}

Node\* RBT\_SearchNode(Node \*pTree, int iTargetID){

if(pTree == Nil)

return NULL;

//데이터 정렬 방식

if( pTree->iID > iTargetID )

return RBT\_SearchNode(pTree->pLeft, iTargetID);

else if(pTree->iID < iTargetID)

return RBT\_SearchNode(pTree->pRight, iTargetID);

else

return pTree;

}

Node\* RBT\_SearchMinNode(Node \*pTree){

if(pTree == Nil)

return Nil;

if(pTree->pLeft == Nil)

return pTree;

else

return RBT\_SearchMinNode(pTree->pLeft);

}

void RBT\_InsertNode(Node \*\*pTree, Node \*pNewNode){

RBT\_InsertNodeHelper(pTree, pNewNode);

pNewNode->Color = \_Node::Red;

pNewNode->pLeft = Nil;

pNewNode->pRight = Nil;

RBT\_RebuildAfterInsert(pTree, pNewNode);

}

void RBT\_InsertNodeHelper(Node \*\*pTree,Node \*pNewNode){

if( (\*pTree) == NULL )

(\*pTree) = pNewNode;

//데이터 정렬 방식

if( (\*pTree)->iID < pNewNode->iID ){

if( (\*pTree)->pRight == Nil ){

(\*pTree)->pRight = pNewNode;

pNewNode->pParent = (\*pTree);

}else

RBT\_InsertNodeHelper( &(\*pTree)->pRight, pNewNode );

}else if( (\*pTree)->iID > pNewNode->iID ){

if( (\*pTree)->pLeft == Nil ){

(\*pTree)->pLeft = pNewNode;

pNewNode->pParent = (\*pTree);

}else

RBT\_InsertNodeHelper( &(\*pTree)->pLeft, pNewNode );

}

}

Node\* RBT\_RemoveNode(Node \*\*pRoot, int iID){

Node \*pRemoved = NULL;

Node \*pSuccessor = NULL;

Node \*pTarget = RBT\_SearchNode( (\*pRoot), iID );

if( pTarget == NULL )

return NULL;

if( pTarget->pLeft == Nil || pTarget->pRight == Nil )

pRemoved = pTarget;

else{

pRemoved = RBT\_SearchMinNode( pTarget->pRight );

pTarget->pData = pRemoved->pData;

pTarget->iID = pRemoved->iID;

}

if( pRemoved->pLeft != Nil )

pSuccessor = pRemoved->pLeft;

else

pSuccessor = pRemoved->pRight;

pSuccessor->pParent = pRemoved->pParent;

if(pRemoved->pParent == NULL){

(\*pRoot) = pSuccessor;

}else{

if( pRemoved == pRemoved->pParent->pLeft )

pRemoved->pParent->pLeft = pSuccessor;

else

pRemoved->pParent->pRight = pSuccessor;

}

if(pRemoved->Color == \_Node::Black)

RBT\_RebuildAfterRemove( pRoot, pSuccessor );

return pRemoved;

}

void RBT\_RebuildAfterInsert(Node \*\*pTree, Node \*pNewNode){

while( pNewNode != (\*pTree) && pNewNode->pParent->Color == \_Node::Red){

if(pNewNode->pParent == pNewNode->pParent->pParent->pLeft){

Node\* pUncle = pNewNode->pParent->pParent->pRight;

if(pUncle->Color == \_Node::Red){

pNewNode->pParent->Color = \_Node::Black;

pUncle->Color = \_Node::Black;

pNewNode->pParent->pParent->Color = \_Node::Red;

pNewNode = pNewNode->pParent->pParent;

}else{

if(pNewNode == pNewNode->pParent->pRight){

pNewNode = pNewNode->pParent;

RBT\_RotateLeft( pTree, pNewNode );

}

pNewNode->pParent->Color = \_Node::Black;

pNewNode->pParent->pParent->Color = \_Node::Red;

RBT\_RotateRight( pTree , pNewNode->pParent->pParent );

}

}else{

Node \*pUncle = pNewNode->pParent->pParent->pLeft;

if(pUncle->Color == \_Node::Red){

pNewNode->pParent->Color = \_Node::Black;

pUncle->Color = \_Node::Black;

pNewNode->pParent->pParent->Color = \_Node::Red;

pNewNode = pNewNode->pParent->pParent;

}else{

if( pNewNode == pNewNode->pParent->pLeft ){

pNewNode = pNewNode->pParent;

RBT\_RotateRight( pTree , pNewNode );

}

pNewNode->pParent->Color = \_Node::Black;

pNewNode->pParent->pParent->Color = \_Node::Red;

RBT\_RotateLeft( pTree, pNewNode->pParent->pParent );

}

}

}

(\*pTree)->Color = \_Node::Black;

}

void RBT\_RebuildAfterRemove(Node \*\*pRoot, Node \*pSuccessor){

Node \*pSibling = NULL;

while( pSuccessor->pParent != NULL && pSuccessor->Color == \_Node::Black){

if(pSuccessor == pSuccessor->pParent->pLeft){

pSibling = pSuccessor->pParent->pRight;

if(pSibling->Color == \_Node::Red){

pSibling->Color = \_Node::Black;

pSuccessor->pParent->Color = \_Node::Red;

RBT\_RotateLeft(pRoot, pSuccessor->pParent);

//pSibling = pSuccessor->pParent->pRight;

}else{

if(pSibling->pLeft->Color == \_Node::Black && pSibling->pRight->Color == \_Node::Black){

pSibling->Color = \_Node::Red;

pSuccessor = pSuccessor->pParent;

}else{

if(pSibling->pLeft->Color == \_Node::Red){

pSibling->pLeft->Color = \_Node::Black;

pSibling->Color = \_Node::Red;

RBT\_RotateRight(pRoot, pSibling);

pSibling = pSuccessor->pParent->pRight;

}

pSibling->Color = pSuccessor->pParent->Color;

pSuccessor->pParent->Color = \_Node::Black;

pSibling->pRight->Color = \_Node::Black;

RBT\_RotateLeft(pRoot, pSuccessor->pParent);

pSuccessor = (\*pRoot);

}

}

}else{

pSibling = pSuccessor->pParent->pLeft;

if(pSibling->Color == \_Node::Red){

pSibling->Color = \_Node::Black;

pSuccessor->pParent->Color = \_Node::Red;

RBT\_RotateRight(pRoot, pSuccessor->pParent);

//pSibling = pSuccessor->pParent->pLeft;

}else{

if( pSibling->pRight->Color == \_Node::Black && pSibling->pLeft->Color == \_Node::Black){

pSibling->Color = \_Node::Red;

pSuccessor = pSuccessor->pParent;

}else{

if(pSibling->pRight->Color == \_Node::Red){

pSibling->pRight->Color = \_Node::Black;

pSibling->Color = \_Node::Red;

RBT\_RotateLeft(pRoot, pSibling);

pSibling = pSuccessor->pParent->pLeft;

}

pSibling->Color = pSuccessor->pParent->Color;

pSuccessor->pParent->Color = \_Node::Black;

pSibling->pLeft->Color = \_Node::Black;

RBT\_RotateRight(pRoot, pSuccessor->pParent);

pSuccessor = (\*pRoot);

}

}

}

}

pSuccessor->Color = \_Node::Black;

}

void RBT\_RotateLeft(Node \*\*pRoot, Node \*pParent){

Node \*pRightChild = pParent->pRight;

pParent->pRight = pRightChild->pLeft;

if(pRightChild->pLeft != Nil)

pRightChild->pLeft->pParent = pParent;

pRightChild->pParent = pParent->pParent;

if(pParent->pParent == NULL)

(\*pRoot) = pRightChild;

else{

if(pParent == pParent->pParent->pLeft)

pParent->pParent->pLeft = pRightChild;

else

pParent->pParent->pRight = pRightChild;

}

pRightChild->pLeft = pParent;

pParent->pParent = pRightChild;

}

void RBT\_RotateRight(Node \*\*pRoot, Node \*pParent){

Node \*pLeftChild = pParent->pLeft;

pParent->pLeft = pLeftChild->pRight;

if(pLeftChild->pRight != Nil)

pLeftChild->pRight->pParent = pParent;

pLeftChild->pParent = pParent->pParent;

if(pParent->pParent == NULL)

(\*pRoot) = pLeftChild;

else{

if(pParent == pParent->pParent->pLeft)

pParent->pParent->pLeft = pLeftChild;

else

pParent->pParent->pRight = pLeftChild;

}

pLeftChild->pRight = pParent;

pParent->pParent = pLeftChild;

}

void RBT\_PrintTree(Node \*pNode, int iDepth, int iBlackCount){

int i=0;

char c = 'X';

int v = -1;

char cnt[100];

if(pNode == NULL || pNode == Nil)

return;

if(pNode->Color == \_Node::Black)

iBlackCount++;

if(pNode->pParent != NULL){

v = pNode->pParent->iID;

if(pNode->pParent->pLeft == pNode)

c = 'L';

else

c = 'R';

}

if(pNode->pLeft == Nil && pNode->pRight == Nil)

sprintf(cnt,"--------- %d",iBlackCount);

else

sprintf(cnt,"");

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

printf(" ");

printf("%d %s [%c, %d] %s\n", pNode->iID,

(pNode->Color == \_Node::Red)?"Red":"Black",c,v,cnt);

RBT\_PrintTree( pNode->pLeft, iDepth+1, iBlackCount);

RBT\_PrintTree( pNode->pRight, iDepth+1, iBlackCount);

}

void RBT\_Inorder( Node \*pNode, void \*pData, void (\*fnConduct)(Node \*pNode,void \*pData) ){

if(pNode == NULL || pNode == Nil)

return;

RBT\_Inorder(pNode->pLeft, pData, fnConduct);

fnConduct(pNode, pData);

RBT\_Inorder(pNode->pRight, pData, fnConduct);

}

\*def.h

#ifndef \_\_D\_LINKED\_LIST\_H\_\_

#define \_\_D\_LINKED\_LIST\_H\_\_

#include <string.h>

#define FILENAME "/Users/SuGyeong/Desktop/test/usersample.txt"

#define TRUE 1

#define FALSE 0

#define WHITE 0

#define GRAY 1

#define BLACK 2

//#define RED 0

//#define BLACK 1

typedef struct Adj {

char id[20];

Adj\* friendnext;

} Adj;

//초기화

void Adj\_init(Adj\* self);

typedef struct LData

{

char id[20];

char date[40];

char name[40];

char word[100][150];// word[100] char \*로?

int color;

int parent;

}LData;

typedef struct Vertex

{

LData data;

Adj \* first;

Vertex \* next;

}Vertex;

void Vertex\_init(Vertex\* self);

void Vertex\_add(Vertex\* self, Vertex\* v);

typedef struct \_likedlist

{

Vertex \* head;

Vertex \* cur;

Vertex \* before;

int(\*comp)(LData d1, LData d2);

int numOfData;

} LinkedList;

//연결리스트

typedef LinkedList List;

//void LDataInit(LData \* pdata);

//LData LDatavalue(char \* str1, char \*str2, char \* str3);

//LData에 값 추가(id, data, name)

//파일 입력받아서 LData에 값 넣고 추가.

//세로 부분 -

//void NodeInit(Node \* puser);

//void Adj\_init(Adj\* self);

//void Friend\_add(Node \* self, Node \* v);

void LDataInit(LData \* pdata);

void FInsert(List \* plist, LData data);

//더미노드 다음 노드에 추가.

void ListInit(List \* plist);

//List 초기화

void LInsert(List \* plist, LData data);

//List에 데이터 삽입. (정렬조건없으면 FInsert)

int LFirst(List \* plist, LData \* pdata);

int LNext(List \* plist, LData \* pdata);

void VertexRemove(List \*plist);

LData LRemove(List\* plist);

int LCount(List\* plist);

//void SetSortRule(List \* plist, int(\*comp)(LData d1, LData d2));

//LData Uservalue(char \* str1, char \*str2, char \* str3);

#endif // !\_\_D\_LINKED\_LIST\_H\_\_#pragma once

\*linked.cpp

#include <stdio.h>

#include <stdlib.h>

#include "def.h"

void ListInit(List \* plist)

{

plist->head = (Vertex\*)malloc(sizeof(Vertex));

plist->head->next = NULL;

plist->numOfData = 0;

}

void FInsert(List \* plist, LData data)

{

Vertex \* vertex = (Vertex\*)malloc(sizeof(Vertex));

vertex->data = data;

vertex->next = plist->head->next;

plist->head->next = vertex;

(plist->numOfData)++;

}//vertex에 삽입.

void LInsert(List \* plist, LData data)

{

if (plist->comp == NULL)

FInsert(plist, data);

/\* else

SInsert(plist, data);\*/

}

int LFirst(List \* plist, LData \* pdata)

{

if (plist->head->next == NULL)

return FALSE;

plist->before = plist->head;

plist->cur = plist->head->next;

\*pdata = plist->cur->data;

return TRUE;

}

int LNext(List \* plist, LData \* pdata)

{

if (plist->cur->next == NULL)

return FALSE;

plist->before = plist->cur;

plist->cur = plist->cur->next;

\*pdata = plist->cur->data;

return TRUE;

}

LData LRemove(List \* plist)

{

Vertex \* rpos = plist->cur;

LData rdata = rpos->data;

plist->before->next = plist->cur->next;

plist->cur = plist->before;

free(rpos);

(plist->numOfData)--;

return rdata;

}

void VertexRemove(List \*plist)

{

Vertex \* rpos = plist->cur;

plist->before->next = plist->cur->next;

plist->cur = plist->before;

free(rpos);

(plist->numOfData)--;

}

int LCount(List \* plist)

{

return plist->numOfData;

}

void Adj\_init(Adj\* self) {

strcpy\_s(self->id, sizeof(self->id), "(none)");

self->friendnext = NULL;

}

void LDataInit(LData \* pdata) {

pdata->color = 0;

pdata->parent = -1;

strcpy\_s(pdata->id, sizeof(pdata->id), "(none)");

strcpy\_s(pdata->date, sizeof(pdata->date), "(none)");

strcpy\_s(pdata->name, sizeof(pdata->name), "(none)");

}

void Vertex\_init(Vertex\* self) {

self->data.color = 0;

self->data.parent = -1;

strcpy\_s(self->data.id, sizeof(self->data.id), "(none)");

strcpy\_s(self->data.date, sizeof(self->data.date), "(none)");

strcpy\_s(self->data.name, sizeof(self->data.name), "(none)");

self->first = NULL;

for (int i = 0; i<100; i++)

{

{

strcpy\_s(self->data.word[i], sizeof(self->data.word[i]), "(none)");

}

}

}

void Vertex\_add(Vertex\* self, Vertex\* v) {

Adj\* a = (Adj \*)malloc(sizeof(Adj));

strcpy\_s(a->id, sizeof(a->id), v->data.id);

a->friendnext = self->first;

self->first = a;

}

\*main

#include <stdio.h>

#include <stdlib.h>

#include "def.h"

int GetMenu();

// 현재 내용을 출력

//void Add(User s[], int \*count);

// 현재 파일 내용을 읽어 옴

//void Save(char\* file, User s[], int count);

void Make\_Userinf(char\* file, List \* plist)

{

FILE\* pFile;

char str[40];

int ch = 0;

fopen\_s(&pFile, FILENAME, "rt");

//파일 오픈

if (pFile == NULL)

{

perror("Error opening file");

return;

}

while (!feof(pFile)){ //LD 포인터 생성.

LData \* newdata = (LData\*)malloc(sizeof(LData));

newdata->color = 0;

newdata->parent = -1;

for (int i = 0; i < 100; i++)

{

{

strcpy\_s(newdata->word[i], sizeof(newdata->word[i]), "(none)");

}

}

fgets(newdata->id, sizeof(newdata->id), pFile);

fgets(newdata->date, sizeof(newdata->date), pFile);

fgets(newdata->name, sizeof(newdata->name), pFile);

FInsert(plist, \*newdata);// 노드에 집어 넣었음.

fgets(str, sizeof(str), pFile);

}

VertexRemove(plist);

fclose(pFile);

}

int WordInput(char\* file, List \* plist)

{

FILE\* pFile;

char str[40];

int ch = 0;

LData data;

int wordcount = 0;

int i = 0;

fopen\_s(&pFile, file, "rt");

//파일 오픈

if (pFile == NULL)

{

perror("Error opening file");

return -1;

}

while (!feof(pFile)) { //LD 포인터 생성.

LData \* newdata = (LData\*)malloc(sizeof(LData));

fgets(newdata->id, sizeof(newdata->id), pFile);

fgets(str, sizeof(str), pFile);//날짜 삭제

fgets(newdata->word[i++], sizeof(newdata->word[i++]), pFile);

printf("%d", i);

if (LFirst(plist, &data))

{

if (data.id == newdata->id)

{

FInsert(plist, \*newdata);

wordcount++;

}

while (LNext(plist, &data))

{

if (data.id == newdata->id)

{

FInsert(plist, \*newdata);

wordcount++;

}

}

}

}

fclose(pFile);

return wordcount;

}

int main(void)

{

List list;

ListInit(&list);

LData data;

int menu = 0;

char file2[100] = "/Users/SuGyeong/Desktop/test/wordsample.txt";

char file3[100] = "friend";

while (menu != 99)

{

menu = GetMenu();

switch (menu)

{

case 0:

Make\_Userinf(FILENAME, &list);

printf("Total user : %d\n", LCount(&list)-1);//쓰레기값이 하나 들어감.\*해결해야함.

printf("Total tweets : %d\n", WordInput(file2, &list));

if (LFirst(&list, &data))

{

printf("%s %s %s \n", data.id, data.date, data.name);

while (LNext(&list, &data))

{

printf("%s %s %s \n", data.id, data.date, data.name);

}

}

break;

case 1:

// Add(user, &count);

break;

case 2:

//Add(user, &count);

break;

case 3:

//Add(user, &count);

break;

case 4:

//Add(user, &count);

break;

case 5:

//Add(user, &count);

break;

case 6:

//Add(user, &count);

break;

case 7:

// Add(user, &count);

break;

case 8:

// Add(user, &count);

break;

case 9:

// Add(user, &count);

break;

case 99:

default:

break;

}

}

// 결과 저장

//Save(FILENAME, users, count);

getchar();

return 0;

}

int GetMenu()

{

int menu;

do

{

printf("----------------------------------\n");

printf("Select Menu:\n");

printf("0. Read data files\n");

printf("1. display statistics\n");

printf("2. Top 5 most tweeted words\n");

printf("3. Top 5 most tweeted users\n");

printf("4. Find users who tweeted a word\n");

printf("5. Find all people who are friends of the above users\n");

printf("6. Delete all mentions of a word\n");

printf("7. Delete all users who mentioned a word\n");

printf("8. Find strongly connected components\n");

printf("9. Find shortest path from a given user\n");

printf("99. Quit\n");

printf("----------------------------------\n");

printf(">>> "); scanf\_s("%d", &menu);

} while ((menu < 0 || menu > 9) && (menu != 99));

return menu;

}

/\*

void display(Vertex vertices[], int count)

{

int i;

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

printf("%s ", vertices[i].id);

printf("%s ", vertices[i].date);

printf("%s ", vertices[i].name);

printf("\n\n");

}

}

\*/

/\*

//LData에 값넣어주는 과정. id랑

void UInsert(char\* file)

{

FILE\* pFile;

char str[40];

int ch = 0;

LData \* newdata = (LData\*)malloc(sizeof(LData));

fgets(vertex[\*count].id, sizeof(vertex[\*count].id), pFile);

fgets(vertex[\*count].date, sizeof(vertex[\*count].date), pFile);

fgets(vertex[\*count].name, sizeof(vertex[\*count].name), pFile);

fgets(str, sizeof(str), pFile);

newdata = Make\_Userinf(char\* id, char\* date, char\* name);

while (!feof(pFile)) {

++(\*count);

fgets(vertex[\*count].id, sizeof(vertex[\*count].id), pFile);

fgets(vertex[\*count].date, sizeof(vertex[\*count].date), pFile);

fgets(vertex[\*count].name, sizeof(vertex[\*count].name), pFile);

fgets(str, sizeof(str), pFile);

}

fclose(pFile);

}

\*/

/\*

void ReadDataFile(User\* user, int count)

{

//int i;

int total\_user = count;

printf("Total user : %d\n", total\_user);

}

void Add(User s[], int \*count)

{

++(\*count);

if ( \*count >= COUNT)

{

printf("더 이상 추가할 수 없습니다.\n");

return;

}

printf("이름 : "); scanf("%s", s[\*count-1].id);

printf("학교 : "); scanf("%s", s[\*count-1].date);

printf("학과 : "); scanf("%s", s[\*count-1].name);

}\*/

/\*void Read(char\* file, User s[], int \*count)

{

FILE\* pFile;

char str[40];

int i;

pFile = fopen(FILENAME, "rt");

fgets(s[0].id, sizeof(s[0].id), pFile);

fgets(s[0].date, sizeof(s[0].date), pFile);

fgets(s[0].name, sizeof(s[0].name), pFile);

fgets(str, sizeof(str), pFile);

for (i = 1; i<8; i++) {

fgets(s[i].id, sizeof(s[i].id), pFile);

fgets(s[i].date, sizeof(s[i].date), pFile);

fgets(s[i].name, sizeof(s[i].name), pFile);

fgets(str, sizeof(str), pFile);

}

fclose(pFile);

}

void Save(char\* file, User s[], int count)

{

FILE\* pFile;

int i;

pFile = fopen(file, "wt");

for (i = 0; i < 8; ++i)

fprintf(pFile, "%10s%10s%10s\n", s[i].id, s[i].date, s[i].name);

fclose(pFile);

}

\*/

-manual

menu창이 뜨고 메뉴에 맞는 번호를 입력하면, 각 번호에 따른 메뉴가 실행이 됨.

각 실행에 해당하는 함수가 설정되어있음.

-report

∗ What data structure you chose and why

-menu 관련 함수 : 스위치문과 반복문을 이용해서 설정.

-연결리스트를 사용하여, 동적 할당을 이용해 일정한 크기의 데이터만을 받을 수 있는 배열보다 활용성이 더 높다. User에 관한 변수를 담는 구조체를 형성하여 연결리스트의 data값으로 사용한다. 수업시간에 배운 인접리스트를 활용하여 친구관계를 나타낸다. 이때 ID값을 이용해 연결한다. 탐색 과정을 통해 트윗한 단어를 보낸 사람(ID를 이용해 찾음)의 구조체 안에 집어넣는다.

menu 0 : user.txt 파일을 받아서 Total user은 연결리스트를 생성할 때마다 numOfData 값을 1씩 증가시켜 구한다. Total friendship records는 friend.txt 파일을 받아서 라인의 수 %3 ==1인 ID(1)를 입력받아서 탐색과정을 통해 찾은 후 이후 입력받는 라인의 수 %3==2인 ID(2)를 갖는 user를 팔로우하는 것으로 표기한다. 즉, 인접리스트와 함수 Add\_Adj를 이용해 ID(1)에 ID(2)를 연결시킨다. 마찬가지로 변수 friendcout를 이용해서 이 횟수를 저장한다. Total tweets의 값을 찾기 위해서 일단, word.txt를 받는다. ID를 입력받은 후 ID를 바탕으로 탐색을 한다. ID가 일치하는 User을 찾은 후 word변수 안에 txt파일로부터 받아온 데이터를 저장한다. word의 길이는 140자 이내이다.

menu 1 : Average number of friends은 menu0을 바탕으로 나온 Total friend ship records를 Total user으로 나눈다. Minimum friends와 Maximum number of friends는 User vertex를 탐색해서 인접연결리스트에 연결된 노드의 수를 대소 비교한다.

Average tweets per user는 menu0에서 얻은 Total tweets을 Total user로 나눈다. User Vertex를 탐색해서 Minium tweets per user와 Maximu tweets per user을 대소 비교를 통해서 구한다.

menu 2 : 각 vertex에서 word 데이터를 새로운 연결리스트를 생성하고 여기에 저장한다. 첫 단어를 가지고 비교를 시작한다.

menu 3 : 탐색과정을 통해 각 User에 저장된 word num을 비교하여 Top 5 most tweet users을 구한다.

menu 4 : 특정 단어를 받은 후, 각 User Vertex를 탐색하여 특정 단어를 트윗한 유저를 매치시켜 찾아낸다.

menu 5 : menu4의 결과를 바탕으로 찾아진 User의 인접리스트를 전부 탐색하여 출력한다. menu 4가 실행되지 않은 경우 “다른 menu를 선택하시오”를 띄운다.

menu 6 : 특정 단어를 받은 후 특정 단어를 갖는 User Vertex를 탐색하여 그 단어를 포함하고 있는 데이터를 삭제를 진행한다.

menu 7 : 특정 단어를 언급한 User Vertex를 탐색하여 찾은 후 삭제를 진행한다.

menu 8 : 수업시간에 배운 strongly connected components를 찾아내는 방식을 이용해서 찾아낸다. -DFS와 DFS\_Transpose를 이용해서 -

menu 9 : dijkstra algorithm을 이용해서 특정 User Vertex로부터의 최단 경로에 있는 User Vertex를 출력한다.

∗ What is your expected performance

위의 ADT를 바탕으로 C언어로 구현한다. Visual studio 2015 사용.

∗ How would you improve the system in the future

아이디어를 구상하였으나, C언어로 파일 받는 것이 미숙하여 제대로 구현하지 못했기에 C언어를 좀 더 공부하여, 연결리스트 뿐만 아니라 다양한 트리를 이용한다면 더 발전시킬 수 있을 것이다.

-self evaluation form

• Submit a github account (10)

- 여러 개의 파일을 제출하여 변화하는 과정을 보여주었으나, 동적할당에서 배열로, 다시 동적할당을 사용하는 것을 볼 수 있다. 함수도 제대로 구현하지 못했다. - 8점. : 제출에 의의.

• Commit source code displaying menu (10)

해당메뉴에 관련한 함수 등을 구현하지 못하였지만, menu display는 잘 구현되었다. - 10

• Commit the first draft of manual (10)

-위의 report를 통해서 구현하고자 했던 방향성을 제시하였으나, 다양한 자료구조를 활용하지 못했다.

• Read data files (20)

-user.txt파일은 입력받아서 각 User Vertex의 데이터 값에 데이터를 입력시켰다. 하지만, word.txt파일을 받았을 때 탐색이 이루어지지 않아 각 User vertex의 word값에 입력이 되지 않았다. friend.txt는 관련 함수를 만들지 못했다. - 10

• Statistics (20)

-total user의 값은 받았지만, 다른 data txt를 받지 못했기 때문에 통계값을 내지 못했다.

: 0

• Top 5 most tweeted words (10)

• Top 5 most tweeted users (5)

• Find all users who mentioned a word (10)

• Find all users who are friend of the above user (5)

• Top 5 strongly connected components (10)

• Find shortest path from a user (id) (10)

->위의 항목들은 아이디어는 구체적으로 생각해보았으나, C언어로 구현하지 못하였으므로 0점 처리.

C언어에서 파일 입출력과 관련한 함수를 제대로 이용하지 못해서 함수 등을 구현하지 못했다. 한편, 기본적인 User에 관해 구조체 정의를 하였고, 연결리스트와 인접리스트를 이용하였고 user.txt 파일을 입력받아서 구조체에 포함되는 변수들에 값을 입력하였다.