**자료구조 과제1**

(ADT 정의/구현 및 시간복잡도 분석)

logo.gif

|  |  |
| --- | --- |
| 과목명 | 자료구조 |
| 담당교수 | 김계영 |
| 학과 | 소프트웨어학부 |
| 학년 | 3 |
| 학번 | 20152994 |
| 성명 | 이진영 |
| 제출일 | 2017.3.22 |



**-목차-**

**1. ADT 정의**

**2. ADT 구현**

1) 원시 코드

2) 수행 결과

**3. 성능 평가**

1) 함수별 실행 step수 산출

2) 시간 복잡도 쎄타(θ) 산출

**ADT 정의/구현 및 시간복잡도 분석**

20152994 이진영

**1. ADT 정의**

ADT ComputerVaccine

objects: 컴퓨터 상에 존재하는 임의의 파일

functions:

임의의 파일 목록을 의미하는 FILE\_LIST, 그리고 감염된 파일 목록을 의미하는 INFECTED\_FILE\_LIST,

바이러스 감염율 ∈ RealNumber, 감염된 파일 개수 ∈ PositiveInteger

RealNumber inspectFile(FILE\_LIST) ::= return 바이러스 감염율 = 파일이 가진 패턴이 알려진 바이러스 패턴에 부합하는 정도

PositiveInteger countInfected(FILE\_LIST) ::= return 임의의 파일 목록 중 감염된 파일 개수

isolateFile(FILE\_LIST) ::= 파일위치를 격리폴더로 이동

deleteFile(FILE\_LIST) ::= 파일위치를 휴지통으로 이동

INFECTED\_FILE\_LIST createInfectedList(FILE\_LIST) ::= inspectFile(), isolateFile(), deleteFile()함수를 이용해 실제로

파일 검사,격리,삭제 행위를 수행한 다음

return INFECTED\_FILE\_LIST

listInfected(INFECTED\_FILE\_LIST) ::= 감염된 파일 목록을 전시

infoFile(FILE, INFECTED\_FILE\_LIST) ::= 특정 파일의 감염정보를 알 수 있도록 검색기능 제공

end ComputerVaccine

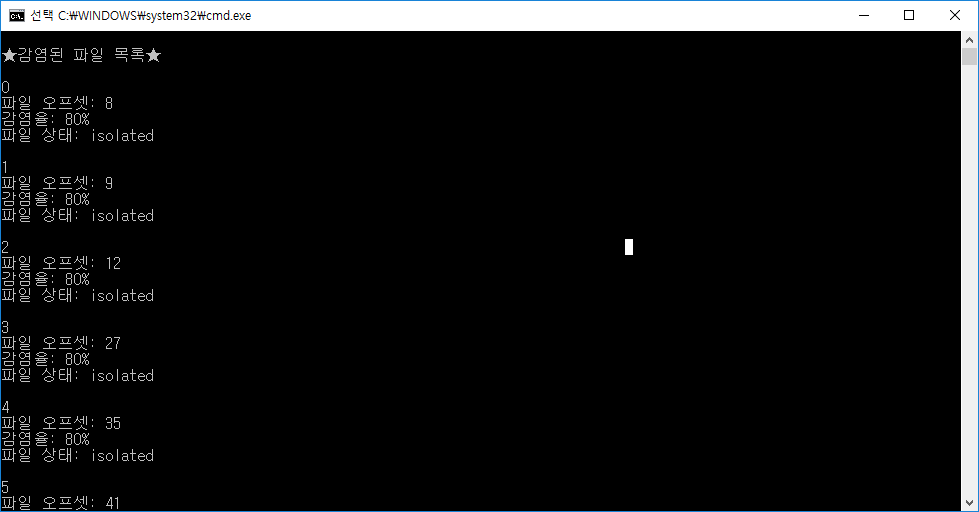
**2. ADT 구현**

1) 원시 코드

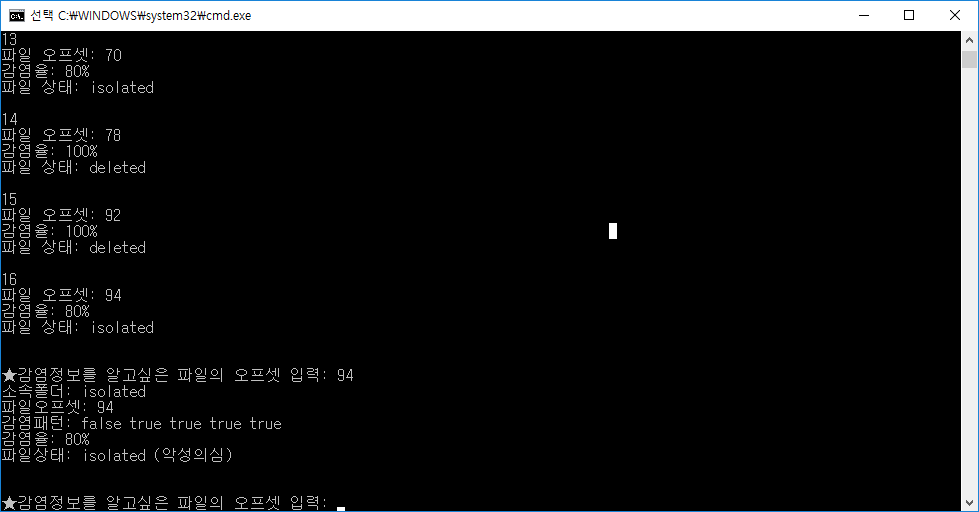
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  #define NUM\_FILE 100  #define NUM\_PATTERN 5  #define RATE\_SUSPECTED 0.8  #define RATE\_INFETCTED 1  typedef struct {  int offset;  char\* location = "default";  bool pattern[NUM\_PATTERN] = { false,false,false,false,false };  }ABSTRACT\_FILE;  typedef struct {  int offset;  float infectedRate;  char\* status;  }INFECTED\_FILE;  float inspectFile(ABSTRACT\_FILE\* file);  int countInfected(ABSTRACT\_FILE\* file);  void isolateFile(ABSTRACT\_FILE\* file);  void deleteFile(ABSTRACT\_FILE\* file);  void createInfectedList(ABSTRACT\_FILE\* file, INFECTED\_FILE\* inf\_file);  void listInfected(INFECTED\_FILE\* inf\_file, int inf\_num);  void infoFile(ABSTRACT\_FILE\* file, INFECTED\_FILE\* inf\_file, int inf\_num);  void main()  {  ABSTRACT\_FILE file[NUM\_FILE];  INFECTED\_FILE\* inf\_file;  srand((unsigned)time(NULL));  for (int i = 0; i < NUM\_FILE; i++) {  file[i].offset = i;  for (int j = 0; j < NUM\_PATTERN; j++){  if (rand()%2) {  file[i].pattern[j] = true;  }  else {  file[i].pattern[j] = false;  }  }  } /\* 테스트를 위해 각 파일 구조체에 임의의 바이러스 패턴을 입력하는 부분 \*/  int inf\_num = countInfected(file);  inf\_file = (INFECTED\_FILE\*)malloc(sizeof(INFECTED\_FILE\*)\*inf\_num);  createInfectedList(file, inf\_file);  listInfected(inf\_file, inf\_num);  while (1) {  infoFile(file, inf\_file, inf\_num);  }  free(inf\_file); /\*해당 ADT 구현체의 동작을 구현하는 부분 \*/  }  float inspectFile(ABSTRACT\_FILE\* file) {  float infectedRate;  int num\_true=0;  for (int j=0; j < NUM\_PATTERN; j++) {  if (file->pattern[j] == true) {  num\_true++;  }  }  return infectedRate = (float)num\_true / NUM\_PATTERN;  }  int countInfected(ABSTRACT\_FILE\* file) {  float infRate;  int num = 0;  for (int i = 0; i < NUM\_FILE; i++) {  infRate = inspectFile(&file[i]);  if (infRate > RATE\_SUSPECTED)  num++;  }  return num;  }  void isolateFile(ABSTRACT\_FILE\* file) {  file->location = "isolated";  }  void deleteFile(ABSTRACT\_FILE\* file) {  file->location = "deleted";  }  void createInfectedList(ABSTRACT\_FILE\* file, INFECTED\_FILE\* inf\_file) {  float infRate;  int j = 0;  for (int i = 0; i < NUM\_FILE; i++) {  infRate = inspectFile(&file[i]);  if (infRate >= RATE\_INFETCTED) {  inf\_file[j].offset = i;  inf\_file[j].infectedRate = infRate;  inf\_file[j].status = "deleted";  deleteFile(&file[i]);  j++;  }  else if (infRate >= RATE\_SUSPECTED) {  inf\_file[j].offset = i;  inf\_file[j].infectedRate = infRate;  inf\_file[j].status = "isolated";  isolateFile(&file[i]);  j++;  }  }  }  void listInfected(INFECTED\_FILE\* inf\_file, int inf\_num) {  printf("\n★감염된 파일 목록★\n");  for (int i = 0; i <inf\_num; i++) {  printf("\n%d\n", i);  printf("파일 오프셋: %d\n",inf\_file[i].offset);  printf("감염율: %0.f%%\n", inf\_file[i].infectedRate\*100);  printf("파일 상태: %s\n", inf\_file[i].status);  }  }  void infoFile(ABSTRACT\_FILE\* file, INFECTED\_FILE\* inf\_file, int inf\_num) {  int i;  printf("\n\n★감염정보를 알고싶은 파일의 오프셋 입력: ");  scanf("%d", &i);  printf("소속폴더: %s\n", file[i].location);  printf("파일오프셋: %d\n", file[i].offset);  printf("감염패턴:");  for (int j = 0; j < NUM\_PATTERN; j++) {  if (file[i].pattern[j] == true)  printf(" true");  if (file[i].pattern[j] == false)  printf(" false");  }  printf("\n");  for (int k = 0; k < inf\_num; k++){  if (i == inf\_file[k].offset) {  printf("감염율: %0.f%%\n", inf\_file[k].infectedRate\*100);  if (strcmp(inf\_file[k].status, "isolated") == 0) {  printf("파일상태: %s (악성의심)\n", inf\_file[k].status);  }  else if (strcmp(inf\_file[k].status, "deleted")==0) {  printf("파일상태: %s (악성확진)\n", inf\_file[k].status);  }  else {  break;  }  }  }  } |

2) 수행결과

①



②



**3. 성능 평가**

1) 함수별 실행 step 수 산출

① inspectFile()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| float inspectFile(ABSTRACT\_FILE\* file) {  float infectedRate;  int num\_true=0;  for (int j=0; j < NUM\_PATTERN; j++) {  if (file->pattern[j] == true) {  num\_true++;  }  }  return infectedRate = (float)num\_true / NUM\_PATTERN;  } | 0 0 0  0 0 0  1 1 1  1 NUM\_PATTERN+1 NUM\_PATTERN+1  1 NUM\_PATTERN NUM\_PATTERN  1 NUM\_PATTERN NUM\_PATTERN  0 0 0  0 0 0  1 1 1  0 0 0 |
| 합계 | 3NUM\_PATTERN+3 |

② countInfected()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| int countInfected(ABSTRACT\_FILE\* file) {  float infRate;  int num = 0;  for (int i = 0; i < NUM\_FILE; i++) {  infRate = inspectFile(&file[i]);  if (infRate > RATE\_SUSPECTED)  num++;  }  return num;  } | 0 0 0  0 0 0  1 1 1  1 NUM\_FILE+1 NUM\_FILE+1  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  0 0 0  1 1 1  0 0 0 |
| 합계 | 4NUM\_FILE+3 |

③ isolateFile()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void isolateFile(ABSTRACT\_FILE\* file) {  file->location = "isolated";  } | 0 0 0  1 1 1  0 0 0 |
| 합계 | 1 |

④ deleteFile()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void deleteFile(ABSTRACT\_FILE\* file) {  file->location = "deleted";  } | 0 0 0  1 1 1  0 0 0 |
| 합계 | 1 |

⑤ createInfectedList()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void createInfectedList(ABSTRACT\_FILE\* file, INFECTED\_FILE\* inf\_file) {  float infRate;  int j = 0;  for (int i = 0; i < NUM\_FILE; i++) {  infRate = inspectFile(&file[i]);  if (infRate >= RATE\_INFETCTED) {  inf\_file[j].offset = i;  inf\_file[j].infectedRate = infRate;  inf\_file[j].status = "deleted";  deleteFile(&file[i]);  j++;  }  else if (infRate >= RATE\_SUSPECTED) {  inf\_file[j].offset = i;  inf\_file[j].infectedRate = infRate;  inf\_file[j].status = "isolated";  isolateFile(&file[i]);  j++;  }  }  } | 0 0 0  0 0 0  0 0 0  1 1 1  1 NUM\_FILE+1 NUM\_FILE+1  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  0 0 0  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  1 NUM\_FILE NUM\_FILE  0 0 0  0 0 0  0 0 0 |
| 합계 | 14NUM\_FILE+2 |

⑥ listInfected()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void listInfected(INFECTED\_FILE\* inf\_file, int inf\_num) {  printf("\n★감염된 파일 목록★\n");  for (int i = 0; i <inf\_num; i++) {  printf("\n%d\n", i);  printf("파일 오프셋: %d\n",inf\_file[i].offset);  printf("감염율: %0.f%%\n", inf\_file[i].infectedRate\*100);  printf("파일 상태: %s\n", inf\_file[i].status);  }  } | 0 0 0  1 1 1  1 inf\_num+1 inf\_num+1  1 inf\_num inf\_num  1 inf\_num inf\_num  1 inf\_num inf\_num  1 inf\_num inf\_num  0 0 0  0 0 0 |
| 합계 | 5inf\_num+2 |

⑦ infoFile()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void infoFile(ABSTRACT\_FILE\* file, INFECTED\_FILE\* inf\_file, int inf\_num) {  int i;  printf("\n\n★감염정보를 알고싶은 파일의 오프셋 입력: ");  scanf("%d", &i);  printf("소속폴더: %s\n", file[i].location);  printf("파일오프셋: %d\n", file[i].offset);  printf("감염패턴:");  for (int j = 0; j < NUM\_PATTERN; j++) {  if (file[i].pattern[j] == true)  printf(" true");  if (file[i].pattern[j] == false)  printf(" false");  }  printf("\n");  for (int k = 0; k < inf\_num; k++){  if (i == inf\_file[k].offset) {  printf("감염율: %0.f%%\n", inf\_file[k].infectedRate\*100);  if (strcmp(inf\_file[k].status, "isolated") == 0) {  printf("파일상태: %s (악성의심)\n", inf\_file[k].status);  }  else if (strcmp(inf\_file[k].status, "deleted")==0) {  printf("파일상태: %s (악성확진)\n", inf\_file[k].status);  }  else {  break;  }  }  }  } | 0 0 0  0 0 0  0 0 0  1 1 1  1 1 1  1 1 1  1 1 1  1 1 1  1 NUM\_PATTERN+1 NUM\_PATTERN+1  1 NUM\_PATTERN NUM\_PATTERN  1 NUM\_PATTERN NUM\_PATTERN  1 NUM\_PATTERN NUM\_PATTERN  1 NUM\_PATTERN NUM\_PATTERN  0 0 0  1 1 1  1 inf\_num+1 inf\_num+1  1 inf\_num inf\_num  1 inf\_num inf\_num  1 inf\_num inf\_num  1 inf\_num inf\_num  0 0 0  1 inf\_num inf\_num  1 1 1  0 0 0  0 0 0  1 1 1  0 0 0  0 0 0  0 0 0  0 0 0 |
| 합계 | 5NUM\_PATTERN+6inf\_num+10 |

2) 시간 복잡도 쎄타(θ) 산출

① 해당 ADT의 총 실행 step 수(연산 횟수) = 8NUM\_PATTERN + 18NUM\_FILE + 11inf\_num + 22

② n = 8NUM\_PATTERN + 18NUM\_FILE + 11inf\_num 이라 가정하면,

\* 해당 ADT의 연산 횟수 f(n) = n+22

※정의[쎄타(Theta, θ)] : 두개의 함수 f(n), g(n)이 주어졌을 때 모든 n>=n0 에 대하여 c1\*|g(n)|<=f(n)<=c2\*|g(n)|을 만족하는 3개의 상수 c1, c2, n0 가 존재하면 f(n)은 θ(g(n))이다.

Proof) 두개의 함수 f(n)=n+22, g(n)=n 이 주어졌을때, 모든 n>=22 에 대하여,

n <= n+22 <= 2n 을 만족하는 상수 c1=1, c2=2가 존재하므로,

**∴** f(n) = n+22 = θ(g(n)) = θ(n) = θ(8NUM\_PATTERN + 18NUM\_FILE + 11inf\_num)