9주차 결과보고서

전공 : 경영학과 학년 : 4학년 학번 : 20190808 이름 : 방지혁

1. 실험시간에 작성한 랭킹 시스템의 자료구조와 랭킹 시스템의 각 기능에 대한 알고리즘을 요약하여 기술하시오. 본인이 선택한 랭킹 시스템을 구현하기 위한 자료구조가 왜 효율적인지 시간 및 공간 복잡도를 통해 보이고, 설명하시오.

자료구조 및 함수 설명

1) 노드

```
typedef struct Node {
  char name[NAMELEN];
  int score;
  struct Node* next;
} Node;
```

자료구조로 단일 연결 리스트를 사용했습니다. Node *head가 리스트의 헤드를 가리키며, 각 노드는 다음 노드를 가지는 포인터가 있기에 서로 연결될 수 있습니다.

2) createRankList 함수 설명

```
void createRankList() {
  // 목적: Input파일인 "rank.txt"에서 랭킹 정보를 읽어들임, 읽어들인 정
  //1. 파일 열기
  FILE *fp = fopen("rank.txt", "r");
  if (fp == NULL) {
    // 파일 열기 실패
    head = NULL;
    score_number = 0;
    printw("There is no rank.txt in current directory\n");
    return;
  }
  // 2. 정보읽어오기
  int cnt = 0;
  if (fscanf(fp, "%d", &cnt) != 1) {
    fclose(fp);
    head = NULL;
    return;
  }
  score_number = cnt;
```

```
  // 3. LinkedList로 저장
  for (int i = 0; i < cnt; i++) {
    char temp_name[NAMELEN];
    int temp_score;
    if (fscanf(fp, "%s %d", temp_name, &temp_score) != 2) {
      // 메모리 해제
      Node* ptr = head;
      while (ptr != NULL) {
        Node* toFree = ptr;
        ptr = ptr->next;
        free(toFree);
      }
      head = NULL;
      printw("Error reading while rank data\n");
      fclose(fp);
      return;
    }
    Node* tempNode = (Node*)malloc(sizeof(Node));
    if (tempNode == NULL) {
      // 메모리 해제
      Node* ptr = head;
      while (ptr != NULL) {
        Node* toFree = ptr;
        ptr = ptr->next;
        free(toFree);
      }
      head = NULL;
      printw("Memory allocation failed\n");
      fclose(fp);
      return;
    }
    tempNode->next = NULL;
    strcpy(tempNode->name, temp_name);
    tempNode->score = temp_score;
    // LinkedList에 추가
    insertNode(&head, tempNode);
  }
  // 4. 파일닫기
  fclose(fp);
}
```

우선 rank.txt 파일을 읽기 모드로 엽니다. 첫 줄에서 저장된 랭킹의 개수를 읽은 후 순차적으로 읽으며 새로운 노드를 동적 할당해서 이름과 점수 정보를 노드에 저장합니다. 그리고 제가 만든

함수 insertNode()함수를 호출하여 오름차순 정렬 상태를 유지하며 삽입될 수 있도록 합니다.

3) insertNode 함수 설명

```
void insertNode(Node** ref_head, Node* newNode) {
  // user code
  if (*ref_head == NULL || newNode->score > (*ref_head)->score) {
    newNode->next = *ref_head;
    *ref_head = newNode;
    return;
  }
  Node* current = *ref_head;
  while (current->next != NULL && current->next->score >= newNode->score) {
    current = current->next;
  }

  newNode->next = current->next;
  current->next = newNode;
}
```

첫 번째 조건문은 리스트가 비어있거나 현재 삽입하려는 노드가 리스트 상에서 최고 점수일 경우로 삽입 노드의 next 포인터에 기존 head 값을 넣고 head에는 삽입 노드의 주소를 넣고 종료합니다. 두 번째 반복문은 그 외의 경우로 헤드에서부터 삽입 위치를 선형적으로 탐색하며 이전 노드의 next 포인터에 삽입 노드의 주소, 삽입 노드의 next 포인터에 다음 노드의 주소를 넣습니다.

4) newRank 함수 설명

```
void newRank(int score) {
  // user code
  // 목적: GameOver시 호출되어 사용자 이름을 입력받고 sc
  char str[NAMELEN+1];
  int i, j;
  clear();
  //1. 사용자 이름을 입력받음
  printw("Your name: ");
  echo();
  scanw("%16s", str);
  noecho();

  Node *newNode = (Node*)malloc(sizeof(Node));
  if (newNode == NULL) {
    printw("Memory allocation failed\n");
    return;
  }

  strcpy(newNode->name, str);
  newNode->score = score;
  newNode->next = NULL;
  insertNode(&head, newNode);
  score_number++;
  writeRankFile();
}
```

게임이 종료될 경우 사용자 이름을 받아 새로운 랭킹을 리스트에 추가하는 함수입니다. 우선 clear()를 통해 화면을 지우고 사용자가 입력을 하도록 프롬프트를 출력합니다. scanw()로 이름을 입력 받고, 입력받은 이름 및 점수를 새롭게 동적할당한 노드에 저장합니다. 마찬가지로

insertNode함수를 호출하여 정렬되어 삽입될 수 있도록 합니다. 그리고 랭킹 개수를 저장하는 전역변수인 score를 1 증가시킵니다. 그리고 writeRankFile()로 즉시 txt파일에 저장합니다.

5) Rank 함수 설명

```
void rank() {
  // user code
  //목적: rank 메뉴를 출력하고 점수 순으로 X부터~Y까지 출력함
  //1. 문자열 초기화
  int X = 1, Y = score_number, ch;
  clear();

  //2. printw()로 3개의 메뉴출력
  printw("1. list ranks from X to Y\n");
  printw("2. list ranks by a specific name\n");
  printw("3. delete a specific rank\n");

  //3. wgetch()를 사용하여 변수 ch에 입력받은 메뉴번호 저장
  ch = wgetch(stdscr);
  //4. 각 메뉴에 따라 입력받을 값을 변수에 저장
  //4-1. 메뉴1: X, Y를 입력받고 적절한 input인지 확인 후(X<=Y), X와 Y사이의 rank 출력
  if (ch == '1') {
    echo();
    printw("X: ");
    scanw("%d", &X);
    printw("Y: ");
    scanw("%d", &Y);
    noecho();

    printw("        name        |    score\n");
    printw("----------------------------------------\n");

    if (X > Y || X < 1 || Y > score_number) {
      printw("search failure: no rank in the list.\n");
      getch();
      return;
    }


    // X와 Y 모두 입력: X부터 Y까지 출력
    Node* ptr = head;
    for (int i = 1; i < X && ptr != NULL; i++) {
      ptr = ptr->next;
    }

    for (int j = X; j <= Y && ptr != NULL; j++) {
      printw("%-19s | %d\n", ptr->name, ptr->score);
      ptr = ptr->next;
    }
  }
```

```
  //4-2. 메뉴2: 문자열을 받아 저장된 이름과 비교하고 이름에 해당하는 리스트를 출력
  else if (ch == '2') {
    char str[NAMELEN+1];
    int check = 0;
    echo();
    printw("input the name: ");
    scanw("%16s", str);
    noecho();

    printw("        name        |    score\n");
    printw("----------------------------------------\n");

    Node* ptr = head;
    while (ptr != NULL) {
      if (strcmp(ptr->name, str) == 0) {
        printw("%-19s | %d\n", ptr->name, ptr->score);
        check = 1;
      }
      ptr = ptr->next;
    }
    if (check == 0) {
      printw("search failure: no name in the list.\n");
      getch();
      return;
    }
  }
```

```
  //4-3. 메뉴3: rank번호를 입력받아 리스트에서 삭제
  else if (ch == '3') {
    int num;
    echo();
    printw("input the rank number: ");
    scanw("%d", &num);
    noecho();

    if (num < 1 || num > score_number) {
      printw("search failure: the rank not in the list.\n");
      getch();
      return;
    }

    Node* ptr = head;
    Node* prev = NULL;
    score_number--;

    for (int i = 1; i < num; i++) {
      prev = ptr;
      ptr = ptr->next;
    }

    if (prev == NULL) {
      // 첫 번째 노드를 삭제하는 경우
      head = ptr->next;
    }
    else {
      prev->next = ptr->next;
    }
    free(ptr);

    printw("result: the rank deleted.\n");
  }
  getch();
}
```

우선 메뉴에 대한 입력을 받습니다. 메뉴 1번의 경우 X부터 Y까지 순위를 출력하는 것으로 X와 Y를 입력 받아, 유효 범위에서 벗어나면 에러메세지를 출력할 수 있도록 했습니다. 범위가 유효한 경우 헤드부터 X-1번 이동하여 시작위치인 X를 찾고 Y번째까지 순회하며 출력합니다. 메뉴 2번의 경우 특정 이름으로 검색하는 경우로 검색할 이름을 입력 받아 리스트를 전체 순회하며 strcmp()로 노드 내에 저장되어 있는 이름과 비교합니다. 그리고 이름이 일치하는 모든 랭킹에 대해 출력합니다. 단 일치하는 노드가 하나도 존재하지 않는다면 check = 0이 되어 에러메세지를 출력하고 종료하게 됩니다. 메뉴 3은 특정 순위를 삭제하는 경우입니다. 마찬가지로 삭제할 순위가 1이상이고 score_number이하인지 유효성 검사를 합니다. 유효하지 않다면 에러 메세지를 출력합니다. 유효한 경우는 또 2가지로 나뉘는데 첫번째 노드를 삭제하는 경우 head 주소만 두번째 노드 주소로 바꿔줍니다. 중간 노드의 경우 prev->next에 ptr->next 값을 넣어주어 삭제 노드의 이전 노드와 다음 노드를 연결시켜줍니다.

6) writeRankFile 함수 설명

```
void writeRankFile() {
  // user code
  // 목적: 추가된 랭킹 정보가 있으면 새로운 정보를 "rank.txt"에 쓰고 없
  int sn, i;
  //1. "rank.txt" 연다
  FILE *fp = fopen("rank.txt", "w");
  if (fp == NULL) {
    // 파일 열기 실패
    printw("Rank.txt open failure while writeRankfile\n");
    return;
  }
  //2. 랭킹 정보들의 수를 "rank.txt"에 기록
  fprintf(fp, "%d\n", score_number);
  Node* ptr = head;
  while (ptr != NULL) {
    fprintf(fp, "%s %d\n", ptr->name, ptr->score);
    ptr = ptr->next;;
  }
  fclose(fp);
}
```

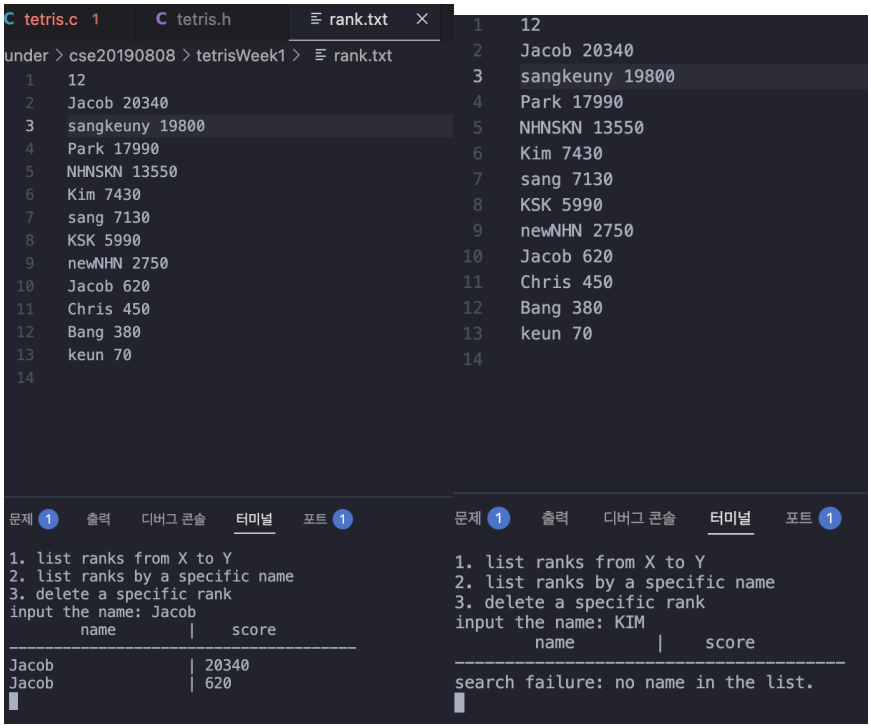
Rank.txt 파일을 쓰기모드로 열어서 첫 줄에 전체 rank 수인 score_number을 기록하고 리스트의 첫 노드부터 순회하며 이름과 점수를 기록합니다 그리고 파일을 닫습니다.

자료구조 및 알고리즘 효율성 평가 & 결과

첫 번째 평가기준인 Game over가 되고, 새로운 랭킹 정보(사용자 이름, 점수)가 등록될 때, 시간 및 공간 복잡도에 대해서 이야기 해보겠습니다. 사용자 이름 입력, 노드 메모리 할당, insertNode, writeRankeFile 호출의 과정을 거칩니다. 여기서 나머지는 다 상수 시간 복잡도이고 inserNode와 writeRankFile만이 O(n)의 시간 복잡도를 가집니다. insertNode의 경우 새 점수가 계속 최고 점수이면 헤드에 삽입만 하면 되기에 O(1)의 시간 복잡도이지만 평균적으로 봤을 때는 리스트 절반까지 순회하는 것으로 이경우 O(n/2)의 시간 복잡이며 최악의 경우 다 순회해야하기 때문에 O(n)이 됩니다. writeRankFile의 경우 모든 노드를 무조건 다 순회해야하기 때문에 O(n)이 될 수 밖에 없습니다. 공간복잡도의 경우 호출된 함수의 스택, 추가적인 변수들을 제외하고 고려하자면 노드 할당을 위해 항상 같은 크기의 메모리만 필요하기에 O(1)이라고 볼 수 있습니다. 실시간으로 정렬이 유지되며 O(n)의 시간 복잡도로 삽입하기에 배열이나 정렬되지 않은 리스트보다 우수하다고 생각됩니다.

두 번째 평가기준인 원하는 랭킹 범위를 입력 받고, 랭킹을 추출하기 위한 과정에서 자료구조를 탐색 및 랭킹 추출에서의 시간 및 공간 복잡도에 대해 이야기 해보겠습니다. X부터 Y까지 출력하기는 기능으로 O(X + (Y – X + 1))이라고 볼 수 있습니다. 최선의 경우 최고 순위만 출력하기에 O(1)의 시간 복잡도일 것입니다. 평균적인 경우 절반을 출력하는 경우로 O(n/2)가 될 것입니다. 공간복잡도는 반복문 및 포인터 변수외에 추가적인 메모리 할당이 없기에 O(1)이라고 볼 수 있습니다.

사용자의 이름을 입력하고 해당하는 랭킹 정보 출력 화면 첨부

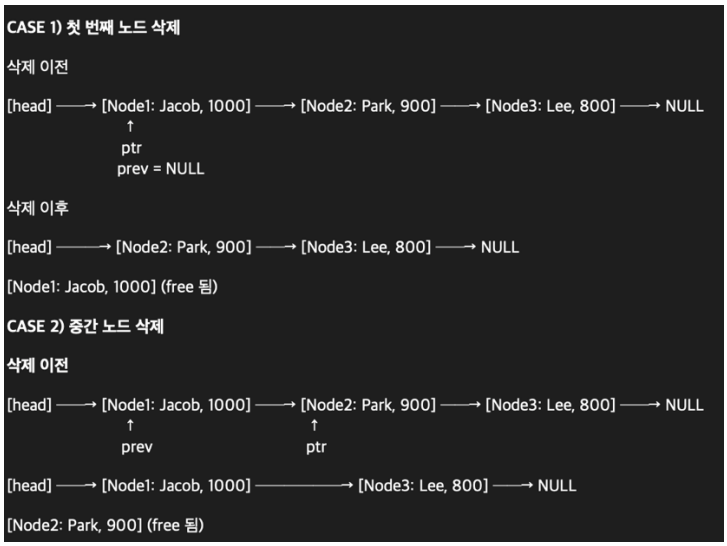


좌측이 이름이 있는 경우, 우측이 없는 경우입니다. 편의 상 rank.txt도 보여지도록 같이 캡쳐했습니다.

사용자 이름으로 원하는 사용자의 이름을 검색할 때의, 시간 및 공간 복잡도

모든 동명이인 처리를 위해 전체 리스트를 끝까지 순회해야 합니다. 그렇기에 O(n/2)입니다. 공간 복잡도의 경우 임시 변수, 임시 저장 문자열 외에는 메모리 할당이 필요 없기에 O(1)입니다.

선택한 자료구조에 맞춰 삭제 알고리즘을 그림으로 표현



2. 본 실험 및 숙제를 통해 습득한 내용을 기술하시오.

본 실험 및 숙제를 통해서 많은 것을 배웠습니다. 처음으로는 정렬된 연결 리스트의 구현입니다. 구현을 하면서 삽입 및 삭제 시 노드의 포인터 조작에 대해 더 익숙해지게 되었습니다. 또한, 동적 메모리 관리입니다. 코드를 작성하며 에러 발생 시 혹은 종료 시 연결 리스트의 메모리 해제 과정에 대해 숙지하게 되었습니다. 또한. 파일 입출력을 다룰 때 fscanf, fprintf등 관련 함수 및 에러처리에 대해 다시 복습하게 되었습니다.