**7주차 Tetris-2 결과보고서**

전공: 수학과 학년: 4 학번: 20161255 이름: 장원태

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.**

실험시간에 작성한 랭킹 시스템의 자료구조는 링크드 리스트(Linked List)이다. 구현한 구조는 다음과 같다.

|  |
| --- |
| typedef struct \_node \*nodepointer;  typedef struct \_node{  char name[NAMELEN];  int score;  nodepointer link;  } node;  nodepointer head = NULL; |

우선 노드를 가리킬 포인터를 따로 nodepointer 타입으로 선언했다. 그리고, 각 노드는 이름, 점수, 그리고 다음 포인터를 가리킬 nodepointer 타입의 link를 포함하고 있는 구조체 형태이다. 각 노드를 nodepointer 타입의 link를 이용하여 이어주며, 검색, 삽입 및 삭제 작업을 진행하기 위해 각 노드를 탐색해야 할 때 해당 link를 이용하여 이동하게 된다. 이 때, 첫 노드를 가리킬 head 노드를 선언하여 노드 탐색의 시작점을 설정해주었다.

랭킹 시스템에 필요한 각 기능에 대한 알고리즘은 다음과 같다.

1. Void createRankList()

|  |
| --- |
| void createRankList(){  // 목적: Input파일인 "rank.txt"에서 랭킹 정보를 읽어들임, 읽어들인 정보로 랭킹 목록 생성  // 1. "rank.txt" 열기  // 2. 파일에서 랭킹정보 읽어오기  // 3. LinkedList로 저장  // 4. 파일 닫기  FILE \*fp;  int i, j, count, tempscore;  char str[100], tempname[NAMELEN];  head = NULL;  nodepointer temp, last=NULL;  // 1. 파일 열기  fp = fopen("rank.txt", "r");  if (fp == NULL) return;  // 2. 정보 읽어오기  fscanf(fp, "%d", &count);  score\_number = count;  fgetc(fp);  // 3. LinkedList로 저장  for (i=0; i<count; i++){  fgets(str, 100, fp);  sscanf(str, "%s %d", tempname, &tempscore);  temp = (nodepointer)malloc(sizeof(struct \_node));  strcpy(temp->name, tempname);  temp->score = tempscore;  temp->link = NULL;  if (!head) {head=temp; last=temp;}  else {  last->link = temp;  last = temp;  }  }  // 4. 파일 닫기  fclose(fp);  } |

이 함수는 랭킹을 저장할 텍스트 파일인 rank.txt 파일에서 랭킹 정보를 읽어 들인 뒤, 우리가 해당 실험에서 사용할 자료 구조인 링크드 리스트에 해당 정보들을 저장하는 함수이다. 우선 rank.txt 파일을 읽기 전용으로 연 뒤, 총 몇 개의 정보가 있는지를 읽는다. 이는 텍스트 파일의 맨 윗줄에 int형으로 저장되어 있다. 이를 전역 변수인 score\_number에 저장해줌으로써 후에 사용할 기능에 도움이 되도록 해준다.

이제, rank.txt 파일에 남은 데이터는 각 플레이어들과 점수에 대한 데이터뿐이다. 해당 데이터들은 한 줄 당 한 명의 플레이어에 대한 데이터를 가지고 있기 때문에, for문을 통해 한 줄씩 읽어온다. 이 때, 한 줄에서 이름과 점수 각각을 서로 다른 변수에 받아오기 위해 sscanf 함수를 사용했다. 그 후, temp라는 이름의 노드를 할당해준다. 이 노드에 읽어온 이름과 점수를 기록하고, link는 일단 NULL로 설정한다. 이 때, 이름의 경우는 string 변수이기 때문에 단순한 “=”을 사용하는 것이 아닌 strcmp 함수를 이용해야 한다는 점에 유의해야 한다.

이제 이 노드를 다른 노드와 연결시켜주는 역할을 해야 한다. 만약 head 노드가 아직 없는 경우는, 만든 노드가 최초의 노드가 될 것이므로 그 노드를 head로 선언해준다. 또한, 앞으로 할 일련의 과정에서 이전 노드인 last 노드를 따로 선언해주는 것이 필요하며, 지금의 경우는 이 노드가 last 노드가 되도록 선언해준다. 이후부터는 이제 head가 존재하는, 혹은 하나의 노드라도 존재하는 상태가 되므로, 이후에 노드를 추가할 때는 last의 link가 추가할 노드를 가리키게 하고, 이렇게 추가된 노드가 이제 last 노드가 되도록 해주어야 한다.

이와 같은 과정을 모두 마무리했으면, 열었던 파일을 다시 닫아주며 마무리한다.

1. Void rank()

|  |
| --- |
| void rank(){  // 목적: rank 메뉴를 출력하고 점수 순으로 X부터 Y까지 출력함  // 1. 문자열 초기화  int X=1, Y=score\_number, ch, i, j;  nodepointer ptr;  clear();  // 2. printw()로 3개의 메뉴 출력  move(0, 0);  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 Y 사이의 rank 출력  if (ch == '1'){  createRankList();  echo();  printw("X: ");  scanw("%d", &X);  printw("Y: ");  scanw("%d", &Y);  if (X == NULL) X=1;  if (Y == NULL) Y=score\_number;  printw(" name | score \n");  printw("------------------------------\n");  if (X>Y){  printw("search failure: no rank in the list\n");  }  else{  ptr=head;  for (i=1; i<X && ptr!=NULL; i++) ptr=ptr->link;  for (j=0; j<=Y-i && ptr!=NULL; j++){  printw(" %-17s| %-8d\n", ptr->name, ptr->score);  ptr=ptr->link;  }    }  noecho();  }  // 4-2. 메뉴2: 문자열을 받아 저장된 이름과 비교하고 이름에 해당하는 리스트를 출력  else if (ch == '2'){  char str[NAMELEN+1];  int check = 0;  createRankList();  echo();  printw("input the name: ");  scanw("%s", &str);  printw(" name | score \n");  printw("------------------------------\n");  for (ptr=head; ptr!=NULL; ptr=ptr->link){  if (!strcmp(str, ptr->name)){  check = 1;  printw(" %-17s| %-8d\n", ptr->name, ptr->score);  }  }  if (!check) printw("search failure: no name in the list\n");  noecho();  }  // 4-3. 메뉴3: rank번호를 입력받아 리스트에서 삭제  else if (ch == '3'){  int num, count=1;  int flag = 1;  nodepointer prev=NULL;  createRankList();  echo();  printw("input the rank: ");  scanw("%d", &num);  for (ptr=head; ptr!=NULL && count<num; count++){  prev = ptr;  ptr = ptr->link;  }  if (num<=0 || count!=num){  printw("search failure: the rank not in the list\n");  flag = 0;  }  else if (!prev){  head = head->link;  free(ptr);  }  else {  prev->link = ptr->link;  free(ptr);  }  if (flag){  score\_number--;  writeRankFile();  printw("result: the rank deleted\n");  }  noecho();  }  getch();  } |

이 함수가 사실상 랭킹 시스템을 총괄하는 함수라고 볼 수 있다. 메뉴에서 “2.rank”를 선택했을 경우에 대한 함수이며, 사용자로 하여금 랭킹 시스템에 관여할 수 있도록 해주는 함수이다.

Rank 메뉴에서는 총 세 가지의 작업을 할 수 있다. 이는 입력받은 X등부터 Y등까지의 랭킹 출력하기, 입력받은 이름의 랭킹 출력하기, 그리고 특정 등수의 랭킹 삭제하기이다. 해당 메뉴들을 각각 1, 2, 3이라는 숫자를 통해 선택하도록 하며, 각 숫자에 대해 다음과 같은 작업을 수행한다.

우선, 1을 선택한 경우이다. 사전에 링크드 리스트를 생성해둬야 하며, 이는 createRankList 함수를 통해 구현한다. 그 후 우선, X와 Y에 대한 정보를 사용자로부터 입력받는다. 그 후, X와 Y에 대해 컨디션 체크를 진행한다. 만약 X를 입력받을 때 사용자가 아무 것도 입력하지 않고 엔터를 쳤다면, X는 자동으로 1이 되게 한다. 만약 Y를 입력받을 때 사용자가 아무 것도 입력하지 않고 엔터를 쳤다면, Y는 자동으로 랭킹에 등록된 데이터의 개수인 score\_number가 되게 한다. 또한, 당연히 X는 Y보다 작거나 같아야 한다. 그래야 그 사이에 있는 데이터를 출력할 수 있게 된다. 그러므로, 만약 X>Y라는 컨디션을 만족하게 되면, 이는 검색할 수 없는 상태이므로 검색 실패 메시지를 출력한다.

이러한 컨디션 체크에서 걸러지지 않은 경우, 정상적으로 랭킹 출력을 수행한다. 물론, 검색 과정은 링크드 리스트를 통해 진행된다. X등부터 Y등까지의 데이터를 출력해야 하며, 이는 곧 X번째 노드부터 Y번째 노드까지의 데이터를 출력하는 것이다. 따라서, 우선 head 노드에서부터 X번째 노드까지 이동시켜야 하며, 이 때 이동 횟수는 (X-1)번이 된다. 이제 해당 노드부터 Y번째 노드까지 이동시키되, 이동시키며 각 노드의 데이터를 출력한다. 이동 횟수는 (Y-X)번이 된다.

다음으로 2를 선택한 경우이다. 이번에도 역시 링크드 리스트를 중점으로 진행할 것이기 때문에, createRankList 함수를 통해 링크드 리스트를 생성한다. 그 후 우선 사용자에게 이름을 입력받는다. 이 이름을 str이라는 string 변수에 저장한다. 이제, str이라는 이름과 같은 이름이 저장되어 있는 노드를 찾으면 된다. Head 노드에서 시작하여 더 이상 노드가 없을 때까지 for문을 돌려, 그 노드의 이름 데이터와 입력받은 str이라는 이름이 서로 일치하면 해당 노드에 대한 정보를 출력한다. 이 때, 이름을 찾았음을 확인해주기 위해, 0이라고 선언되어 있던 check를 1로 설정해준다. 이 과정을 모두 마친 뒤, 만약 check가 0이라면 이름을 찾지 못한 것이므로 검색 실패 메시지를 출력한다.

마지막으로 3을 입력한 경우이다. 이 경우에도 링크드 리스트를 사전에 createRankList 함수를 통해 만들어준다. 그 후, 사용자에게 등수를 입력받는다. 이제 그 등수의 노드를 찾기 위해 head 노드에서부터 이동해야 하며, 필요한 이동 횟수는 (num-1)이 된다. 이 때, 각각이 몇 번째 노드인지 알려주는 count 변수를 선언해주며, 노드의 이동에 따라 count 변수를 1씩 증가시키는 쪽으로 진행한다. 그리고, 삭제 과정에서 이전 노드를 이용해야 하므로, 이전 노드도 prev라는 변수를 이용하여 선언해준다.

이 for문을 종료한 뒤, 만약 num 자체가 0보다 작거나 같아 유효한 숫자가 아니거나, 도착한 노드가 입력받은 등수가 아닌 경우(즉, 노드의 개수보다 더 큰 등수의 숫자를 입력받은 경우) 해당 랭킹이 없는 것이므로 검색 실패 메시지를 출력하며, 과정 성공을 의미하는 flag 변수를 0으로 설정한다. 그렇지는 않더라도 만약 “1등”의 랭킹을 삭제하려고 한다면, head 노드의 재정의가 필요하다. 이는 head의 링크를 타 2번째 노드가 head 노드가 되도록 하면 된다. 그렇지 않다면, 이전 노드인 prev 노드의 링크가 현재 노드가 아닌 현재 노드의 다음 노드를 가리키게 한다. 이런 식으로 노드의 재정의가 진행된 뒤에는, 현재 가리키고 있는 노드를 free시켜 삭제하면 된다. 성공적으로 과정이 끝나 flag가 1인 상태로 유지되어 있다면, 한 노드가 삭제되었으므로 전역 변수인 score\_number를 1 감소시켜줘야 하며, 이에 따라 rank.txt 파일을 다시 작성해주어야 한다. 그 후, 삭제 성공 메시지를 출력한다.

1. Void writeRankFile()

|  |
| --- |
| void writeRankFile(){  // 목적: 추가된 랭킹 정보가 있으면 새로운 정보를 "rank.txt"에 쓰고 없으면 종료  int sn, i;  nodepointer temp;  // 1. "rank.txt" 연다  FILE \*fp = fopen("rank.txt", "r");  if (fp==NULL) return;  // 2. 랭킹 정보들의 수를 "rank.txt"에 기록  fscanf(fp, "%d", &sn);  fclose(fp);  fp = fopen("rank.txt", "w");  // 3. 탐색할 노드가 더 있는지 체크하고 있으면 다음 노드로 이동, 없으면 종료  if (sn == score\_number) return;  else {  fprintf(fp, "%d\n", score\_number);  temp = head;  while (temp){  fprintf(fp, "%s %d\n", temp->name, temp->score);  temp = temp->link;  }  }  fclose(fp);  } |

이 함수는 랭킹 정보를 갱신하는 함수로, 변경된 랭킹 정보가 있으면 그 정보를 rank.txt 파일에 저장하는 함수이다. 우선, rank.txt 파일을 연다. 이제 변경된 랭킹 정보가 있는지를 확인해야 한다. 이는 전역 변수인 score\_number를 이용하여 확인한다. 텍스트 파일에 있는 데이터의 개수는 변하지 않지만, 전역 변수인 score\_number는 삽입 혹은 삭제의 과정을 거쳐 변경될 수 있다. 그래서, 텍스트 파일의 첫 줄에 있는, 데이터의 총 개수와 score\_number를 비교했을 때 만약 같다면 다시 return하고, 다르다면 파일에 입력하기 시작한다. 우선, 기존 텍스트 파일의 서식과 일치하게 현재 데이터의 총 개수인 score\_number를 먼저 출력해준다. 그 후, head 노드에서부터 링크드 리스트를 탐색하여, 각 노드의 이름과 점수를 출력해준다. 마지막으로, 열었던 파일을 다시 닫아주며 마무리한다.

1. Void newRank(int score)

|  |
| --- |
| void newRank(int score){  // 목적: GameOver시 호출되어 사용자 이름을 입력받고 score와 함께 리스트의 적절한 위치에 저장  char str[NAMELEN+1];  int i, j;  nodepointer new, ptr=head;  clear();  // 1. 사용자 이름을 입력받음  printw("your name: ");  echo();  getstr(str);  noecho();  // 2. 새로운 노드를 생성해 이름과 점수를 저장, score\_number가  new = (nodepointer)malloc(sizeof(struct \_node));  strcpy(new->name, str); new->score=score; new->link=NULL;  if (score\_number == 0) head=new;  else {  if (head->score <= new->score){  new->link = head;  head = new;  }  else{  while (ptr->link != NULL && ptr->link->score > score)  ptr=ptr->link;  new->link = ptr->link;  ptr->link = new;  }  }  score\_number++;  writeRankFile();  } |

이 함수는 게임 오버 시 호출되어, 사용자에게 이름을 입력받고, 그 사용자가 얻은 점수와 입력한 이름을 링크드 리스트 및 rank.txt 파일에 저장하는 함수이다. 게임 오버 시 호출되는 함수이므로, 스크린을 clear한 뒤 사용자의 이름을 str이라는 string 변수에 입력받는다. 그 후, 새로운 노드를 생성하여 그 정보들을 저장하고, link는 NULL로 설정한다. 만약 기존에 아무 데이터가 없었다면, 해당 노드를 head 노드로 저장해준다.

이제 기존에 데이터가 있었을 경우, 점수가 높은 순서에 맞추어 해당 노드를 삽입하는 일이 남았다. 만약 1등인 head 노드보다도 점수가 높다면, 기존 head 노드가 새 노드의 link가 가리키는 노드가 되게 하고, 해당 노드를 head 노드로 설정한다. 그렇지 않다면, 이미 점수가 높은 순서대로 정렬되어 있는 링크드 리스트를 탐색하기 시작한다. 이 탐색 과정은 link를 통해 일어나며, 다음 노드의 점수가 삽입할 노드의 점수보다 낮거나 같아지면 그 위치에 노드를 삽입한다. 삽입 과정에서, 현재 노드의 다음 노드를 새 노드의 link가 가리키게 하고, 현재 노드의 link는 새 노드를 가리키게 하면 정상적으로 삽입이 완료된다.

이와 같은 과정이 모두 끝나면 이제 파일을 갱신해야 한다. 이 때, 노드 하나가 추가된 것이므로 전역 변수인 score\_number를 1 증가시켜주고, writeRankFile 함수를 호출하여 rank.txt 파일을 갱신해주며 마무리한다.

이러한 링크드 리스트는 시간 복잡도와 공간 복잡도 면에서 효율적임을 보인다. 예비 보고서에서 생각했던 다른 자료 구조인 배열과 비교해보자. 두 자료 구조의 효율성의 차이는 원소 삽입 및 삭제의 과정에서 두드러진다. 배열의 경우, 원소를 특정 위치에 삽입하고자 하면, 원래 그 인덱스에 있던 데이터부터 그 후의 인덱스들에 있던 모든 데이터를 하나씩 밀어내야 한다. 삭제의 경우도, 그 후의 인덱스들에 있던 모든 데이터를 하나씩 끌어와야 한다. 그러므로, 배열은 시간 복잡도가 각 과정에 대해 모두 O(n)으로 나타난다. 반면에 링크드 리스트의 경우는 특별히 다른 데이터를 밀거나 끄는 일 없이, 단순히 링크의 수정만으로 이루어질 수 있다. 그러므로, 같은 과정들에 대해 링크드 리스트는 O(1)의 시간 복잡도를 가지게 되어 시간을 크게 단축할 수 있다. 공간 복잡도의 측면에서도, 두 구조 모두 O(n)으로 같지만, 랭킹 갱신 때마다 인덱스의 문제 때문에 배열을 다시 선언해야 하는 배열과는 달리 링크드 리스트는 기존의 구조에서 추가 및 삭제만 진행하면 되므로 메모리도 효율적으로 사용할 수 있다.

1. **본 실험 및 숙제를 통해 습득한 내용을 기술하시오.**

해당 실험 및 숙제를 통해 링크드 리스트를 구현해봄을 통해, 링크드 리스트의 사용에 더 익숙해지고 삽입 및 삭제와 같이 리스트를 수정해야 하는 부분에 대해서도 이전보다 더 숙련되게 진행할 수 있었다. 뿐만 아니라, 시간 복잡도 및 공간 복잡도를 고려할 수 있게 되었고, 그에 따라 적당한 자료 구조를 찾을 줄도 알게 되었다. 이전에는 링크드 리스트를 구현하기 어려워하여 배열을 더 선호하였으나, 배열이 링크드 리스트에 비해 시간 복잡도가 더 커 시간 효율이 떨어지고, 공간 효율도 떨어진다는 것을 알게 되어 이제 링크드 리스트를 사용하는 것을 더 선호하게 되었다. 또한, 기존에 많이 사용하지는 않았던 파일에 대해서도 다루는 것에 좀 더 능숙해지게 되었다.