2020년도 2학기 컴퓨터공학설계및실험I

8주차 예비 보고서

20170175 김태안

1. 실습 목적

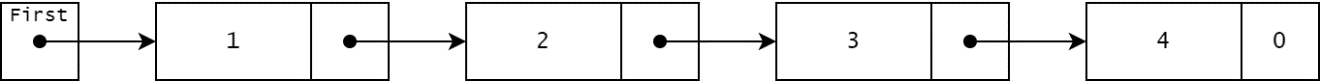
일반적인 게임은 사용자의 기록을 등록하여, 확인할 수 있는 기능을 갖는다. 테트리스 프로젝트에서도 이러한 기능을 하는 랭킹 시스템을 구현한다. 이 시스템은 테트리스 플레이가 종료되면, 사용자의 이름과 점수를 기록하고, 기록된 랭킹 정보들을 확인할 수 있는 기능을 갖는다. 이 랭킹 정보들은 테트리스 프로그램 내에서 효율적인 자료구조를 사용하여 관리되고, 테트리스 프로그램을 종료할 때는 rank.txt 파일에 기록되어, 지속적으로 유지된다.

1. 관련 이론

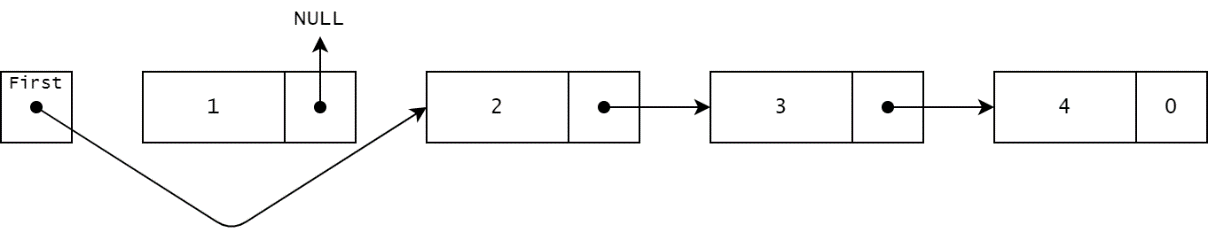
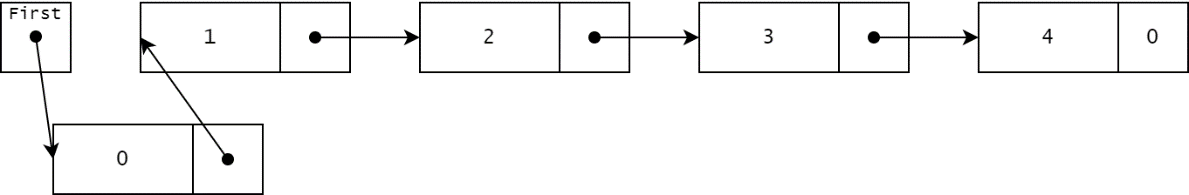
**연결 리스트 (Linked List)**

일반적으로 배열에 데이터를 저장할 경우, 원소들은 일정한 거리만큼 떨어져 메모리에 저장된다. 예를 들어, 배열 L에 정수 1, 2, 3을 저장한다면 각 요소들은 시작 지점에서 0, 4, 8만큼 떨어진 곳에 저장된다. 이러한 방식은 임의의 원소에 접근하기에는 매우 편리하지만, 임의로 배열의 길이를 늘이거나 새 원소를 중간에 삽입하는 경우는 불편하다는 단점이 있다.

이러한 문제는 연결 리스트를 통해 해결할 수 있다. 연결 리스트는 원소를 지정된 위치에 저장하는 대신, 원소가 저장된 주소의 포인터를 저장해 필요할 경우 원소로 이동하는 방식이다. 연결 리스트의 각 노드는 저장할 데이터 필드와 다음 노드를 가리키는 포인터 또는 링크로 구성되어 있다. 포인터를 통해 다음 노드로 이동할 수 있다.



연결 리스트는 일반적인 배열보다 원소의 삽입과 삭제가 쉽다. 중간에 새 원소를 삽입하거나 삭제하면 공백을 다시 채우기 위해 모든 원소를 이동해야 하는 배열과 달리, 연결 리스트는 포인터가 가리키는 원소를 수정하면 쉽게 원소를 삽입/삭제할 수 있기 때문이다. 그리고, 연결 리스트는 데이터의 개수를 미리 정하여 리스트의 크기를 할당하지 않아도 계속해서 데이터를 추가할 수 있다는 장점이 있다. 그러나, 특정 위치의 원소를 확인하기 위해 첫 노드에서부터 해당 노드까지 탐색을 해야 한다는 단점이 있다.



연결 리스트에 새 원소를 삽입할 경우, 먼저 새 노드를 생성해 저장할 데이터를 데이터 필드에 저장한다. 그리고, 삽입할 위치 앞의 노드가 가리키는 포인터를 새 노드가 가리키게 한다. 그리고, 앞의 노드는 새 노드를 가리키게 포인터를 저장한다. 삭제의 경우, 삭제할 노드 앞의 포인터가 삭제할 노드의 다음 노드를 가리키게 한 뒤, 삭제할 노드의 포인터는 NULL을 가리키게 하면 된다.[1]

1. 실습 방법

랭킹 시스템은 테트리스 프로그램을 재시작하는 경우에도 정보가 저장되어 이전의 기록을 불러와야한다. 이를 위해 특정 파일에 점수와 이름을 기록한 뒤, 프로그램이 시작되면 랭킹 파일을 불러오고, 종료되면 이를 갱신하는 방식으로 프로그램을 구성할 수 있다.

프로그램이 실행되는 동안 랭킹을 보여주거나, 새 기록을 기록하기 위해서, 프로그램이 시작되면 랭킹 파일을 읽어 이를 저장해야 한다. 이때, 배열이나 연결 리스트를 사용할 수 있다.

배열을 사용하는 경우, 이름과 점수를 원소로 하는 구조체의 배열을 사용한다. 랭킹 파일을 읽어 줄바꿈의 개수를 계산한 뒤, 개수보다 큰 배열을 선언한다. 게임이 끝날 때마다 배열에 새 기록을 기록한 뒤, 점수 순서로 정렬하고, 랭킹 파일을 쓴 다음, 다시 랭킹 파일을 불러와 배열을 선언한다. 이 경우, 배열의 공간 복잡도는 기록한 랭킹 N+1이다. 원소를 삽입하는 경우, 배열의 맨 끝에 새 원소를 삽입하므로 시간 복잡도는 O(1)이고, 이를 정렬하기 위한 Selection Sort 알고리즘은 O(N2)의 시간 복잡도를 가진다. i번째 원소를 삭제할 경우, 이를 탐색하기 위해 i번의 탐색이 필요하고, 삭제한 뒤, i 뒤의 모든 원소 j에 대해 a[j] = a[j+1]를 실행해야 하므로 N-i의 시간이 소요된다. 즉, O(N)의 시간 복잡도를 가진다. 그러나 이와 같은 구현은 랭킹 파일을 계속해서 읽고 써야하기 때문에 복잡하며, 시간이 많이 소요된다.

연결 리스트를 사용해 랭킹 파일의 내용을 저장할 경우, 존재하는 기록의 개수만큼의 노드가 필요하다. 즉, N만큼의 공간 복잡도를 가진다. 연결 리스트는 배열에 비해 노드의 삽입/삭제가 자유롭기 때문에 삽입과 동시에 정렬을 할 수 있다.

|  |
| --- |
| linkedlist temp, front, back, new;  새 기록을 new에 저장  temp=\*first;  while(!temp):  if (temp<=새 기록<=temp->link)  front=temp;  back=temp->link;    front->link=new;  new->link=back;  break;  else if (temp->link == NULL) // 맨 마지막  temp->link=new;  break;  temp = temp->link; |

처음부터 노드가 삽입되어야 할 위치를 탐색한다. 탐색 중 삽입되어야 할 위치를 찾으면 현재 위치를 front, 그 다음 원소를 back에 저장하고 front와 back 사이에 새 기록 new를 삽입한다. first가 NULL이거나 새 기록이 제일 낮은 기록이라면 맨 뒤에 새 원소를 추가한다. 이때 삽입할 위치 i만큼의 시간이 걸리므로 시간 복잡도는 O(N)이다.

|  |
| --- |
| linkedlist temp, front, back, new;  새 기록을 new에 저장  temp=\*first;  while(!temp):  if (삭제할 기록 == temp->link)  front=temp;  back=temp->link->link;  temp->link->link=NULL;  front->link=back;  break;  else if (삭제할 기록 == first) // 맨 첫 기록이 삭제할 기록  first = temp->link;  break;  temp = temp->link; |

삭제의 경우도 마찬가지로 처음부터 삭제할 노드를 탐색한다. 삭제할 노드를 찾은 경우, front에 현재 노드를, back에 삭제할 노드의 다음 노드를 저장하고, 이를 연결한다. 위 수도코드와 같은 방식의 경우, temp 다음 노드를 기준으로 탐색하기 때문에 삭제할 노드가 첫 노드면 탐색할 수 없다. 이를 방지하기 위해 삭제할 기록이 첫 기록인 경우를 조건문으로 따로 처리하도록 한다. 이 경우 시간 복잡도는 삽입과 마찬가지로 O(N)이며, 파일의 읽고 쓰기는 프로그램이 종료될 때만 이루어지므로 배열보다 단순하다고 할 수 있다.

1. 기타

**Selection Sort**

Selection Sort는 정렬되지 않은 원소 중 가장 작은(큰) 값을 찾아 정렬된 배열 다음 자리에 놓는 정렬 방식이다. 배열을 모두 탐색해 가장 작은(큰) 값을 찾아 맨 첫 원소로 둔다. 그 다음, 두번째 원소부터 배열을 모두 탐색해 가장 작은(큰) 값을 찾아 두번째 원소로 둔다. 이를 반복해 배열의 모든 원소를 정렬하는 알고리즘을 Selection Sort라고 한다. 원소의 개수를 n개라고 할 때, Selection Sort는 O(n2)의 시간 복잡도를 가진다.[2]

|  |
| --- |
| for (i =0; i < n; i++) {  list[i]에서부터 list[n-1]까지 정수 값을 검사한 결과  list[min]이 가장 작은 정수 값이라고 하자;  list[i]와 list[min]을 서로 교환;  } |

1. 참고 문헌
2. “4. List.” *Fundamentals of Data Structures in C*, by Ellis Horowitz et al., University Press, 2008.
3. “1. Basic Concepts.” *Fundamentals of Data Structures in C*, by Ellis Horowitz et al., University Press, 2008.