객체지향프로그래밍

ASSN1

20180551

이준석

POVIS ID: ljs9904ljs

명예서약(Honor code)

“나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.”

1. 프로그램 기능에 대한 개요

1-1. 1번 문항에 대하여 (prob_1.cpp)

이 프로그램은 나무 막대기를 일렬로 세워놨을 때 보이는 나무 막대기의 개수를 구하는 프로그램이다. 왼쪽에서 봤을 때의 개수와 오른쪽에서 봤을 때의 개수를 합하여 출력한다.

1-2. 2번 문항에 대하여 (prob_2.cpp)

이 프로그램은 나무 막대기를 일렬로 세워놓고 왼쪽에서 봤을 때 최대로 보이는 나무 막대기의 개수를 구하는 프로그램이다. 중간에서 뒤의 나무 막대기를 가리는 나무 막대기를 원하는 만큼 배제할 수 있다.

1-3. 3번 문항에 대하여 (prob_3.cpp)

이 프로그램은 Y년이 흐른 후 남아 있는 빙하 덩어리의 개수를 구하는 프로그램이다.

2. 프로그램의 구성 및 전체 구조와 알고리즘

2-1. 1번 문항에 대하여 (prob_1.cpp)

총 나무 막대기의 개수, 막대기 각각의 높이를 입력 받는다. 그러고 나서 보이는 나무 막대기의 개수를 구한다. 앞에 있는 막대기보다 뒤에 있는 막대기가 높다면 막대기가 보이는 것으로 생각할 수 있으므로 그것을 이용한다. 가장 앞에 있는 막대기의 높이를 뒤에 있는 막대기와 비교하고 더 높은 막대기가 있다면 temp에 더 높은 막대기의 값을 저장하고 개수를 하나 늘린다. 그것을 왼쪽과 오른쪽에서 각각 반복한 뒤에 그 값들을 합하여 출력하는 것이다.

2-2. 2번 문항에 대하여 (prob_2.cpp)

최대로 보이는 개수를 구하기 위해서 LIS라는 알고리즘을 이용한다. 총 나무 막대기의 개수, 막대기 각각의 높이를 입력 받고 최대 개수를 출력한다. LIS 알고리즘에서는 이분 탐색을 이용한 lower bound라는 것이 사용된다. lower bound는 어떤 값이 주어졌을 때 집합 내에서 그것보다 작거나 같은 값을 찾는 방법이다. LIS를 통해 가장 길이가 긴 수열을 구하는데 여기에서는 나무 막대기 높이들의 수열이다. LIS를 진행하는 배열을 따로 만드는데 거기에 새로운 값이 추가되어 들어올 때마다 보이는 막대기의 개수를 하나씩 더해주는 방식으로 최대로 보이는 막대기의 개수를 구한다.

2-3. 3번 문항에 대하여 (prob_3.cpp)

2차원 배열의 각각의 칸에 몇 년이 흘러야 다 녹아 없어지는지 값을 저장해 놓는다. 예를 들어 arr[0][1] == 7이고 Y == 4라면 7-4=3이므로 Y년이 흐른 후에 빙하가 남아 있는 상태가 된다. 상하좌우로 연결되어 있는 빙하는 한 덩어리로 보기 때문에 DFS라는 알고리즘을 이용하여 남아있는 총 빙하 덩어리의 개수를 구한다. 행과 열과 Y년을 입력받고 행렬의 각각의 칸의 값을 입력받은 후 빙하 덩어리의 개수를 출력한다. DFS는 함수 내에서 자기 자신을 다시 한 번 호출한다. 즉, 재귀 함수이다. 그래서 빙하가 존재하는 곳에 대해서 상하좌우를 탐색함으로써 이어져 있는 빙하에 대해 중복해서 개수를 세지 않고 한 번만 세고 넘어갈 수 있도록 한다. 그리고 대각선으로 이어져 있는 것은 이 문제에서 이어져 있지 않

은 것으로 보기 때문에 상하좌우만 탐색하도록 for문을 구성하였다.

3. 기타 프로그램을 이해하는데 필요한 내용

LIS: Longest Increasing subSequence의 약자이다.
컴퓨터 공학에서, 최장 증가 부분 수열 문제는, 주어진 수열에서 오름차순으로 정렬된 가장 긴 부분수열을 찾는 문제이다. 여기서의 부분 수열은 연속적이거나 유일할 필요는 없다. 최장 증가 부분 수열은 알고리즘을 포함한 수학, 랜덤 행렬 이론, 표현론, 그리고 물리학과 관련된 다양한 분야에서 연구되었다. 최장 증가 부분 수열 문제는, 입력 수열의 길이가 n일 때 O(nlogn) 의 시간에 풀이가 가능하다.
(출처: 위키백과, 최장 증가 부분 수열)

DFS: Depth First Search의 약자이다.
깊이 우선 탐색(depth-first search: DFS)은 맹목적 탐색방법의 하나로 탐색트리의 최근에 첨가된 노드를 선택하고, 이 노드에 적용 가능한 동작자 중 하나를 적용하여 트리에 다음 수준(level)의 한 개의 자식노드를 첨가하며, 첨가된 자식 노드가 목표노드일 때까지 앞의 자식 노드의 첨가 과정을 반복해 가는 방식이다.
(출처: 위키백과, 깊이 우선 탐색)

4. 프로그램 실행 방법

4-1. 1번 문항에 대하여 (prob_1)

prob_1 폴더에 있는 prob_1.cpp과 Makefile을 이용해서 make하여 prob_1.exe파일을 만들어서 ./prob_1.exe를 입력하여 프로그램을 실행한다. 첫 줄에 몇 개의 막대기인지 입력하고 두 번째 줄에 각각의 막대기들의 높이를 입력한다. 그러면 결과값이 출력된다.

4-2. 2번 문항에 대하여 (prob_2)

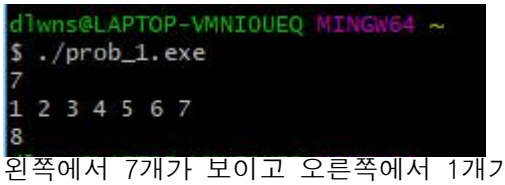
prob_2 폴더에 있는 prob_2.cpp과 Makefile을 이용해서 make하여 prob_2.exe파일을 만들어서 ./prob_2.exe를 입력하여 프로그램을 실행한다. 첫 줄에 몇 개의 막대기인지 입력하고 두 번째 줄에 각각의 막대기들의 높이를 입력한다. 그러면 결과값이 출력된다.

4-3. 3번 문항에 대하여 (prob_3)

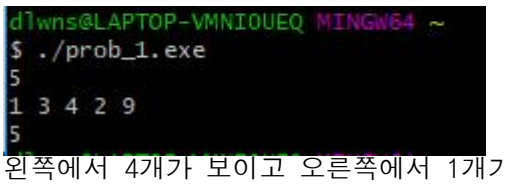
prob_3 폴더에 있는 prob_3.cpp과 Makefile을 이용해서 make하여 prob_3.exe파일을 만들어서 ./prob_3.exe를 입력하여 프로그램을 실행한다. 첫 줄에 가로(W)와 세로(H)와 년(y)을 순서대로 입력하고 두 번째 줄부터 행렬의 각 칸의 값을 입력한다. 그러면 결과값이 출력된다.

5. 예제

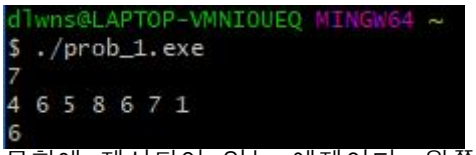
5-1. 1번 문항에 대하여



왼쪽에서 7개가 보이고 오른쪽에서 1개가 보이므로 합인 8을 출력한다.



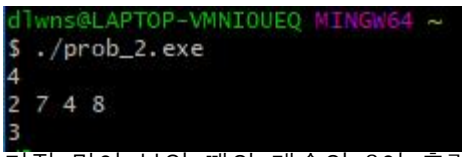
왼쪽에서 4개가 보이고 오른쪽에서 1개가 보이므로 합인 5를 출력한다.



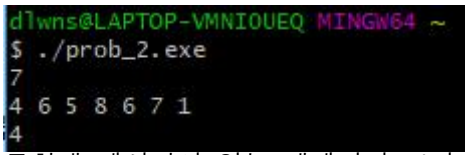
문항에 제시되어 있는 예제이다. 왼쪽에서 3개가 보이고 오른쪽에서 3개가 보이므로 합인 6을 출력한다.

5-2. 2번 문항에 대하여

가장 많이 보일 때는 1 2 5 6 9 일 때이므로 5가 출력된다. (물론 LIS 알고리즘 특성 상 최장 증가 부분 수열의 각각의 값을 알 수는 없고 1 4 5 7 9와 같은 구성도 있을 수 있다. 하지만 결과적으로 최장 증가 부분 수열의 길이는 5이다.)

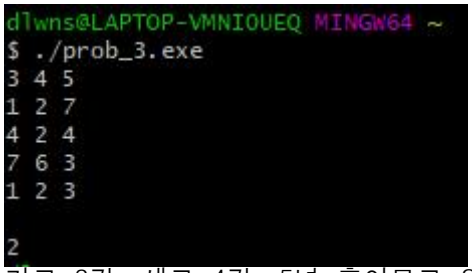


가장 많이 보일 때의 개수인 3이 출력된다. 이것도 위와 마찬가지로 그 구성 요소가 무엇인지 하나로 특정할 수는 없다.

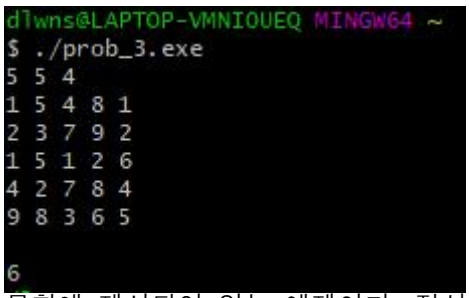


문항에 제시되어 있는 예제이다. 4가 정상적으로 출력된다.

5-3. 3번 문항에 대하여



가로 3칸, 세로 4칸, 5년 후이므로 위와 같은 값을 입력했을 때 첫 줄의 7과 셋 째 줄의 7-6이 녹지 않고 남아 있다. 따라서 빙하는 2덩이 남아있게 되고 정상적으로 2가 출력된다.



문항에 제시되어 있는 예제이다. 정상적으로 6이 출력된다.

6. 토론

6-1. 1번 문항에 대하여

이 문제를 해결할 때, 어떻게 해야 보이는 막대기의 수를 셀 수 있을지 고민하였다. 그 결과 깨달은 것이 앞서 나온 막대기들보다 높을 때만 보인다는 점이었다. 그것을 이용해서 temp에 현재 막대기를 저장해두고 다음으로 높은 막대기가 나올 때마다 temp의 값을 갱신하면서 그 뒤의 있는 막대기들과 높이를 비교하는 방식으로 코드를 구성하게 되었다.

6-2. 2번 문항에 대하여

맨 처음에는 1번 문항을 해결할 때처럼 일일이 비교하려고 했으나 그런 방식으로 하게 되면 시간 복잡도가 n*n이 되어버리는 문제가 발생하였다. 그 방식으로는 100만 개 일 때 10초 안에 해결할 수 없으므로 문제가 되었다. 그래서 LIS 알고리즘에 대해 조사하였고 nLog(n) 방식의 알고리즘을 찾게 되었다. 그것을 이용하여 2번 문항을 해결하였다.

6-3. 3번 문항에 대하여

상하좌우로 연결되어 있는 빙하를 어떻게 같은 것으로 취급할지 고민하다가 DFS 알고리즘을 이용하게 되었다. DFS를 통해 맨 처음 녹지 않은 빙하를 발견하면 그 빙하의 상하좌우를 탐색하여 녹지 않은 빙하가 이어져 있을 때 동일한 빙하로 취급하도록 코드를 구성하였다.

7. 결론

이번 과제를 통하여 같은 결과에 도달하더라도 효율적인 알고리즘을 이용한다면 훨씬 빠른 시간 내에 문제를 해결할 수 있음을 알게 되었다. 2번 문항에서 본인이 생각했던 방식으로 했다면 10초 안에 결과를 출력할 수 없었을 텐데 이분 탐색을 이용한 LIS 알고리즘을 사용함으로써 효과적인 풀이를 할 수 있었다. 그리고 이 과정에서 완벽하게는 아니더라도 어느 정도 LIS 알고리즘을 배울 수 있었던 것도 좋은 배움이었다고 생각한다. 마찬가지로 3번 문항을 해결할 때 이용한 DFS도 훌륭한 배움이었다. 완벽하다고는 할 수 없지만 이런 방식의 탐색 방법도 있다는 것을 배움으로써 앞으로 다른 문제를 직면했을 때 효과적으로 문제를 푸는 데 도움을 받을 수 있을 것이다.

8. 개선 방향

프로그램의 기능을 다양화하는 법에 대해 생각해 본다면 2번 문항에서 가장 많이 보일 때 나무 막대기 높이들의 구성이 무엇들로 이루어져 있는지 출력하도록 만들 수도 있다. 여러 가지 방식이 존재한다면 그 방식들을 전부 찾아내는 것이다. 3번 문항에서 대각선으로 이어져 있는 것도 하나로 보는 조건을 생각할 수 있다. 그리고 3번 문항에서 DFS 알고리즘을 이용했는데, 이것 말고도 BFS 알고리즘을 이용해보고 어떤 방식이 더 빠른지 비교하여 그것을 채택한다면 프로그램의 속도를 증가시킬 수 있을 것이다. 혹은 또 다른 알고리즘을 찾아내서 적용하는 방법도 있을 수 있겠다.

9. 참고 문헌

1번 문제에서, https://kldp.org/node/110745
2번 문제에서(LIS 관련), https://www.crocus.co.kr/583
3번 문제에서(DFS 관련), https://jun-itworld.tistory.com/21