2020 Spring OOP Assignment Report

과제 번호 : 1
학번 : 20190084
이름 : 권민재
Povis ID : mzg00

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

Problem 1. 온전한 사각형

1. 프로그램 개요

□ 모눈종이의 가로 세로 크기를 입력받아, 모눈종이의 한 대각선을 지나지 않는 사각형의 개수를 구하는 프로그램이다.

□ 이 프로그램은 STDIO (Standard Input Output)를 이용하여 가로 세로 크기를 각각 입력받으며, prob_1 폴더 내의 prob_1.cpp에 C++11로 코드가 작성되어 있다.

2. 프로그램의 구조 및 알고리즘

□ **구조**

■ **int main()**

◆ 가로 세로 칸 수를 입력받아서 온전한 사각형의 개수를 출력한다.

■ **int gcd(int a, int b)**

◆ a와 b를 입력받아서 w와 h의 최대공약수를 반환한다.

□ **알고리즘** (대각선이 지나지 않는 사각형 구하기)

■ 우선 파이썬으로 편리하게 대각선이 지나는 사각형의 개수를 구해보았다.

```
import math
w = int(input("w >> "))  # 가로 길이 입력
h = int(input("h >> "))  # 세로 길이 입력
total = 0  # 대각선을 지나는 사각형 개수
f = lambda x: (h / w) * x  # 대각선의 1차 함수 꼴
for i in range(1, w + 1):  # 주어진 범위에서,
```

```
# (f(i) 올림) -(f(i-1) 내림)=(i 번째 열에서 대각선이 지나는 사각형의 개수)
    total += (math.ceil(f(i)) - math.floor(f(i - 1)))
print(total)
```

- 그 결과를 정리하면 아래와 같다.

(가로)X(세로)	2X2	2X3	2X4	3X3	3X4	3X6
개수	2	4	4	3	6	6

- 이때, 대각선이 지나는 사각형의 개수 N은 (가로) + (세로) – (가로와 세로의 최대공약수)개이고, 대각선이 지나지 않는 사각형은 (전체 칸 수) - N임을 알 수 있다.
- 최대공약수는 유클리드 호제법을 이용해 구할 수 있다. 유클리드 호제법은 어떤 정수 a, b의 최대공약수가 b와 a%b의 최대공약수와 같다는 정리로, b가 0이 될 때까지 gcd(b, a%b)를 재귀적으로 호출하면 최대공약수를 구할 수 있다.

□ 변수

- **int w, int h** 각각 모눈종이의 가로 칸 수와 세로 칸 수를 저장한다.
- **int a, int b** 최대공약수를 구할 두 수를 입력받는다.

3. 토론 및 개선

□ 좌표평면 위에서의 알고리즘도 생각해보며, 조금만 더 생각하면 GCD를 쓰는 더 좋은 알고리즘을 설계할 수 있음을 알았다.

□ 스택 오버플로우를 대비해 반복문으로 gcd를 구현할 필요성이 있다.

4. 참고 문헌

□ 유클리드 호제법 (개념 및 코드) https://ko.wikipedia.org/wiki/유클리드_호제법

□ 대각선을 지나는 사각형의 개수 https://m.blog.naver.com/orbis1020/220664563768

Problem 2. 리코더

1. 프로그램 개요

□ 여분 리코더를 가진 친구들이 도난 당한 친구들에게 리코더를 빌려줄 수 있을 때, 리코더를 가져갈 수 있는 최대 학생 수를 출력하는 프로그램이다.

□ 전체 학생 수, 리코더를 도난당한 학생 수, 여분 리코더를 가지고 있는 학생 수, 리

코더를 도난당한 학생과 여분이 있는 학생들의 번호를 STDIO로 입력받는다.

2. 프로그램의 구조 및 알고리즘

□ **구조**

■ **int main()**

◆ 데이터들을 입력받고, 학생 수 크기의 벡터를 만든다. recorder_max로 리코더를 가져갈 수 있는 최대 학생 수를 구하고 출력한다.

■ **int recorder_max(std::vector<int>& student_v)**

◆ 리코더를 가져갈 수 있는 최대의 학생 수를 반환하며, main의 student_v 벡터를 참조한다. 도난당한 친구에게 여분 리코더를 빌려주는 일을 수행한다.

□ **알고리즘** (Greedy algorithm으로 리코더를 가져갈 수 있는 최대 학생 수 구하기)

■ Greedy algorithm은 매 순간마다 부분적으로 최적인 선택을 통해 최적의 전체 결과 값을 근사하는 알고리즘이다.

■ student_v 벡터는 기본적으로 0으로 초기화 되어 있고, 리코더를 도난 당한 경우에는 -1을, 여분의 리코더가 있는 경우엔 +1이 학생의 값에 더해진다.

■ student_v 벡터 순회 중 0보다 작은 값을 만났을 때, 우선 앞 번호 친구에게 여분 리코더 빌리기를 시도해보고, 없다면 뒷 번호 친구에게서 빌리기를 시도한다. 이미 탐색이 끝난 앞 번호 친구에게 먼저 시도하는 것이 합리적이기 때문이다.

■ 위 과정 이후, 학생의 값이 0 이상인지 확인한다. 학생의 값이 0 이상이면 수업에 리코더를 가져갈 수 있으므로, 리코더를 가져갈 학생 수로 추가한다.

□ 변수

■ **int student_num, int lost, int extra, int result**

◆ 각각 전체 학생 수, 리코더를 잃어버린 학생 수, 여분의 리코더를 가지고 있는 학생 수, 수업시간에 리코더를 가져갈 수 있는 학생 수를 저장한다.

■ **vector<int> student_v**

◆ 전체 학생 수 만큼의 크기를 가지는 벡터이다. 각 학생의 초기화된 값은 0이며, 0보다 작으면 도난당했음을, 0보다 크면 여분 리코더가 있음을 뜻한다.

3. 토론 및 개선

□ 도난 당한 학생 벡터 **V**에 학생 번호를 저장하고, 여분을 가진 학생 번호가 입력될 때마다 **V**를 탐색한다면 메모리를 많이 차지하는 문제를 개선할 수 있을 것이다.

□ Greedy algorithm을 통해서도 충분히 의미있는 결과값을 근사할 수 있음을 알았다.

4. 참고 문헌

□ Greedy algorithm의 정의 (개념) https://en.wikipedia.org/wiki/Greedy_algorithm

Problem 3. 위스키 시음

1. 프로그램 개요

□ 이 프로그램은 연속으로 3개 이상의 위스키를 마실 수 없는 상황에서 최대한 마실 수 있는 위스키의 양을 구하는 프로그램이다.

□ 이 프로그램은 사용자로부터 위스키 잔의 수와 위스키 잔에 들어있는 위스키의 양들을 입력받으며, prob_3 폴더 내의 prob_3.cpp에 C++11로 코드가 작성되어 있다.

2. 프로그램의 구조 및 알고리즘

□ **구조**

■ **전처리기**

◆ **CACHE_INIT** 이 프로그램에서 쓰이는 캐시 벡터의 초기화 값을 지정한다.

◆ **MAX(X, Y)** X와 Y 중 큰 수를 반환하는 매크로이다.

■ **int main()**

◆ 위스키에 대한 정보를 담는 벡터 whiskey와 cache를 만든다. 벡터 whiskey에 각 위스키 양을 저장하고, 함수 whiskey_max의 결과 값을 출력한다.

■ **int whiskey_max(const int idx, const vector<int>& whiskey, vector<int>& cache)**

◆ 이 함수는 whiskey 벡터의 idx 부터 끝까지의 구간에 대해 알고리즘에 따라

서 최대로 먹을 수 있는 위스키의 양을 반환하는 함수이다.

□ **알고리즘** (Dynamic Programming으로 최대의 위스키 양 구하기)

■ Dynamic programming이란 큰 문제를 간단한 여러 개의 문제로 쪼개어 푸는 방법을 일컫는다. 우리는 이 문제를 점화식처럼 풀어 볼 것이다.

■ k번째 위스키 잔의 위스키 양을 a_k, n번째 이후 먹을 수 있는 최대의 위스키 양을 S_k이라고 할 때, $S_k = \max(a_k + a_{k+1} + S_{k+3}, a_k + S_{k+2}, S_{k+1})$라고 할 수 있다.

■ 위의 식을 계산하는 과정에서 함수의 반복된 호출으로 계산이 느려지기에, 벡터 cache의 cache[k]에 S_k를 저장하여 memoization을 구현했다.

□ 변수

■ **int size, int input**

◆ size는 위스키 잔의 개수를 저장하며, input은 입력을 위한 임시변수이다.

■ **vector<int> whiskey, vector<int> cache**

◆ whiskey는 위스키 잔에 담긴 위스키의 양을 저장하는 벡터이며, 벡터 cache에서 cache[idx]는 whiskey_max(idx, whiskey, cache)의 값을 가진다.

■ **int case_list[3]**

◆ 알고리즘 문단에서 설명한 각 3가지 케이스의 결과 값을 저장하는 배열이다.

■ **int idx, int partial_sum, int partial_max**

◆ idx로 계산할 벡터 구간의 시작점을 설정하고, partial_max에 이 구간에서의 최대 위스키 양을 저장하며, partial_length는 이 구간의 길이를 저장한다.

3. 토론 및 개선

□ 이번 문제를 통해 memoization으로 효율적인 계산을 할 수 있음을 알았다.

□ 이 프로그램은 함수 whiskey_max를 재귀적으로 호출하는데, 스택 오버플로우를 대비해 반복문을 이용하여 재귀적인 호출을 없애도록 설계해볼 필요가 있을 것 같다.

4. 참고 문헌

□ Dynamic Programming의 정의 (개념) https://ko.wikipedia.org/wiki/동적_계획법