Writeup

20140475 임범혁

I did the project 2 alone.

A. Description of your data structure and your main algorithm

1. checker.cpp : checker라는 함수를 따로 만들었는데 input을 string으로 받고 checker함수에 입력해주면 괄호가 잘 parenthesis correctness를 확인하기 시작한다. count를 0으로 설정해 '('가 들어오면 count를 +1하고, ')'가 들어오면 count를 -1하였다. 여기서 count가 0보다 작아진다는 것은 괄호가 모두 닫혔는데 ')'가 들어와 count가 음수가 되었다는 소리이므로 FALSE를 return하게 하였다. 모든 '('와 ')'를 확인하였을 때 count가 0이라는 것은 모든 괄호가 닫혔다는 소리이므로 TRUE를 return하게 하였고, 그렇지 않으면 모든 괄호가 닫히지 않았다는 소리이므로 FALSE를 return하게 된다.

2. generator.cpp : 먼저 input이 command line argument로 argv[]로 들어오게 된다. argv[]의 첫번째 element는 name of program 일 거고, 두번째 element가 우리가 원하는 n값일 것이다. atoi함수를 통해 integer로 변환에 n에 입력해준 다음 parenthesis를 만드는 함수 generator에 넣어주었다. 여기서 아무것도 입력되어 있지 않은 string S를 같이 generator함수에 입력해주게 되는데 함수 안에서 S에 '(', ')'를 추가하면서 원하는 parenthesis를 만들고, 출력하게 만들어 주었다. 함수에서 기본적으로 recursive function 을 이용하였다. 함수에서 n은 string에 추가할 수 있는 '('의 개수이고 S에 '('가 추가되면 n은 하나씩 줄어들게 된다. opened는 '(' 로 괄호가 얼마나 열려 있는가 하는 것인데 만약 string '()('면 1개의 괄호가 열려 있으므로 opened가 1이 되고, '(((('이면 3개의 괄호가 열려 있으므로 opened가 3이 된다. parenthesis가 만들어지는 과정은 binary tree와 비슷하게 생겼는데 n이 0보다 크면(추가할 수 있는 '('가 있다는 소리) '('를 추가해주고, 새로운 generator함수를 부르고, opened가 0보다 크면('('가 아직 열려있다는 소리) ')'를 추가해주고 새로운 generator를 부르면서 모든 가능한 parenthesis를 만들게 된다. 마지막으로 n과 opened가 모두 0 이면 '('를 모두 사용하였고, 모두 잘 닫혀있다는 소리이므로 출력을 해주게 된다.

3. merger.cpp : 여기서는 sorted linked list를 이용하여 코드를 짜주었다. main 함수에서 line마다 string으로 getline함수를 통해 input을 받고, stringstream segment를 통해 띄어쓰기를 구분하여 한 줄에 쓰여있는 두 숫자를 input[0], input[1]에 저장할 것이다. 입력된 것이 두 숫자인지, 첫번째 숫자는 0이상이고, 두번째 숫자는 0보다 큰지 확인해준 다음 input[0]을 start_offest으로 input[1]을 length로 AddNewNode함수에 입력하여 기존의 linked list에 새로운 Node를 추가하는 구조이다. 먼저 start_offset과 length의 integer값, 다음 Node를 가르키는 next 포인터를 내재하고 있는 Node class를 만들어 주었다. 그리고 start_offset, length pair가 입력으로 들어오면 AddNewNode를 통해 새로운 Node를 생성하고 linked list에 추가해주었는데 start_offset크기 순서대로 linked list가 이루어져 있을 수 있게 추가해주었다. 새로 들어올 Node의 start_offset을 linked listd의 첫번째 start_offset부터 비교해주었고, 새로 들어올 start_offset보다 큰 Node가 나타나면 그 사이에 새로운 Node를 추가해 주었다. Node가 추가되면서 동시에 sorted가 되는 것이다. 입력이 끝나고 sorted linked list가 만들어준 다음 규칙에 맞게 merge를 해주는 Merger 함수를 통해 원하는 결과를 얻는다. Merger 함수는 만약 n번째 Node와 n+1번째 노드를 합쳐야 한다면 n번째 length를 계산에 맞게 바꿔준 다음 next 포인터가 n+2번째 Node를 향하게 하는 구조이다.

B. Explanation of why you think your algorithm is correct, and how you tested your program.

1. checker.cpp : count가 0보다 작아지지만 않는다면 parenthesis가 맞다는 것이므로 내가 짠 코드가 맞다고 생각하고, counting만 하는 구조이기 때문에 가장 빠른 algorithm이라고 생각한다. 많은 input cases를 입력해보면서 program이 맞는지 확인해보았다.

2. generator.cpp : recursive 함수를 이용하면 모든 가능한 parenthesis를 만들 수 있다. n을 1부터 10까지 입력하며 output이 맞는지 확인해보았고, parenthesis 수가 2nCn/(n+1) 인지 체크하였다.

3. merger.cpp : 새로운 start_offset, length가 들어옴에 따라 Node를 생성하여 sorted linked list를 만들어준 다음 merging를 해주는게 더 효율적이라고 생각하였고, 여러 input cases를 입력해보면서 program이 맞는지 확인하였다.

C. Time and space complexity of your algorithm and timing measurement of your program with various input values - you can use 'time' command on linux for the measurement.

1. checker.cpp : time complexity O(n), space complexity O(1)

```
corret: 4, wrong: 4

real    0m22.082s
user    0m0.002s
sys     0m0.000s
```

```
corret: 8, wrong: 7

real    0m34.715s
user    0m0.002s
sys     0m0.000s
```

```
corret: 10, wrong: 13

real    0m50.850s
user    0m0.002s
sys     0m0.000s
```

2. generator.cpp : time complexity O(log(n)), space complexity O(n)

```
real    0m0.001s
user    0m0.001s
sys     0m0.000s
```
n = 1

```
real    0m0.002s
user    0m0.002s
sys     0m0.000s
```
n = 3

```
real    0m0.002s
user    0m0.002s
sys     0m0.000s
```
n = 5

```
real    0m0.035s
user    0m0.013s
sys     0m0.022s
```
n = 10

3. merger.cpp : time complexity O(n), space complexity O(n)

```
[5 1
5 1

real    0m18.606s
user    0m0.002s
sys     0m0.000s
```
input : <5, 1>

```
[5 1
[6 2
[8 3
5 6

real    0m7.242s
user    0m0.002s
sys     0m0.000s
```
input : <5, 1>, <6, 2>, <8, 3>

```
[4 5
[4 2
[0 1
[0 3
[9 3
[7 3
[23 3
0 3
4 8
23 3

real    0m26.861s
user    0m0.000s
sys     0m0.002s
```
input : <4, 5>, <4, 2>, <0, 1>, <0, 3>, <9, 3>, <7, 3>, <23, 3>