Data Structures 2020

HW3

2016726028

이민재 | 자료구조 | 2020/12/04

과제 개요

Homework #3

- A password with size N can be found by searching the text for the most frequent substring with N characters. After finding the password, the password can be used to decode the message.

- Your mission has been simplified as you are only requested to write a program that, given the size of the password and the encoded message, determines the password following the strategy given above.

Input & Output

- Standard/Console Input
  - Input consists of two lines with the size of the password, $3 \le N \le 30$, followed by the text representing the encoded message whose size is less than 5 mega bytes. To simplify things, the input text only includes lower-case alphabet letters.
  - Sample input

```
3
accacacea
```

- Standard/Console Output
  - As output your program should print the password string.
  - Sample output

```
cac
```

과제 수행 환경

Visual studio 2019 Enterprise ver 16.7.7

With 8-core Ryzen cpu 3700x

Language : C++

프로그램 설계

Precond: 프로그램은 사용자 입력(비밀번호 길이, 소문자 알파벳 문자열)을 받음

Postcond: 비밀번호길이 만큼의 val 를 key 값과 함께 저장, 최다 빈출을 출력.

ㅇ사용자 입력

```
21
22  int main()
23  {
24      cin.tie(_Newtie:NULL); // 입출력속도 늘리기
25      ios::sync_with_stdio(_Newsync:false); // 출력속도 빠르게
26      cin >> numberN; // pw 길이
27      cin.ignore();
28
29      cout << decoder([&]cin) << endl;
30  }
```

먼저, 사용자로부터 지정된 패스워드의 길이를 입력 받는다. Cin>>numberN 으로 numberN 이라는 integer 변수에 값을 저장했다. 이후 엔터 키 입력을 무시하도록 cin.ignore() 함수를 호출해주어 바로 알파벳 문자열이 헤더로 오도록 조정해주었다.

이후, 비밀번호 해독을 위해 decoder 함수로 입출력을 넘긴다.

ㅇ DECODER(입력저장)

```
string decoder(istream& ins);
//precond : 입력스트림통해 긴 소문자 알파벳으로 이루어진문자열 입력받음
//postcond : 가장 많이 중복된 패턴을 반환
```

Decoder 함수는 string 형 리턴 타입을 지닌 함수로, 입출력 스트림을 인자로 받고

문자열을 적절한 처리를 통해 최다 빈출 암호를 선정, string 형태로 반환한다.

```
getline([&]cin, [&]buf);
vector<char> inputCode(_First:buf.begin(), _Last:buf.end());

int EOS = inputCode.size() - numberN + 1;
```

먼저, 입출력스트림에 존재하는 소문자 알파벳 문자열을 buf 스트링에 임시로 저장한다. 스트링 자체로 처리하는 방법도 생각해 보았지만, vector 구조가 요소의 탐색, 처리 과정에서 속도의 이득이 있다고 판단하여 buf 로 받은 스트링을 char 형 벡터 컨테이너에 저장하는 방식을 선택하였다. buf.begin 과 buf.end 를 사용하여 입력받은 스트링의 시작부터 끝까지 벡터 컨테이너 생성과 함께 저장하였다.

```
numberN =3 이고 aabbccddeeff 입력시 마지막 문자열은 'eff'
```

탐색의 범위는 EOF 까지로, numberN 길이의 마지막 요소까지 탐색되면 탐색을 종료시키고 싶기 때문에, EOF 의 길이는 전체 사이즈에서 numberN 을 빼고 1 을 더한값이 된다.

```
for(int i = 0; i< EOS; i++) // numberN =3 이고 aabbccddeeff
{
    string pwd; // 페스워드 패턴 저장할 스트링
    for (int j = i; j < i+numberN; j++)
        pwd += inputCode[j];

    m[pwd]++; //없으면 set 생성, 있으면 키값 증가
    //printProgress(i + 1, EOS);
}
```

이후 벡터에 저장되어있는 char 요소들을 i=0 부터 EOS 까지 탐색하여 numberN 길이만큼 추출하고 pwd string 에 차례로 저장한다. 예를들어 '3' 'aaab'가 입력된 경우, numberN 은 3 임으로 inputCode[0] 부터 inputCode[2] 까지인 'aaa' 가 m[aaa] ,key = 1 의 set 로 unordered_map 에 저장된다. For 루프 반복문으로 이후에는 inputCode[1] 부터 inputCode[3] 까지인 'aab'가 m[aab], key =1 의 set 로 저장된다.

ㅇ DECODER(암호 해독)

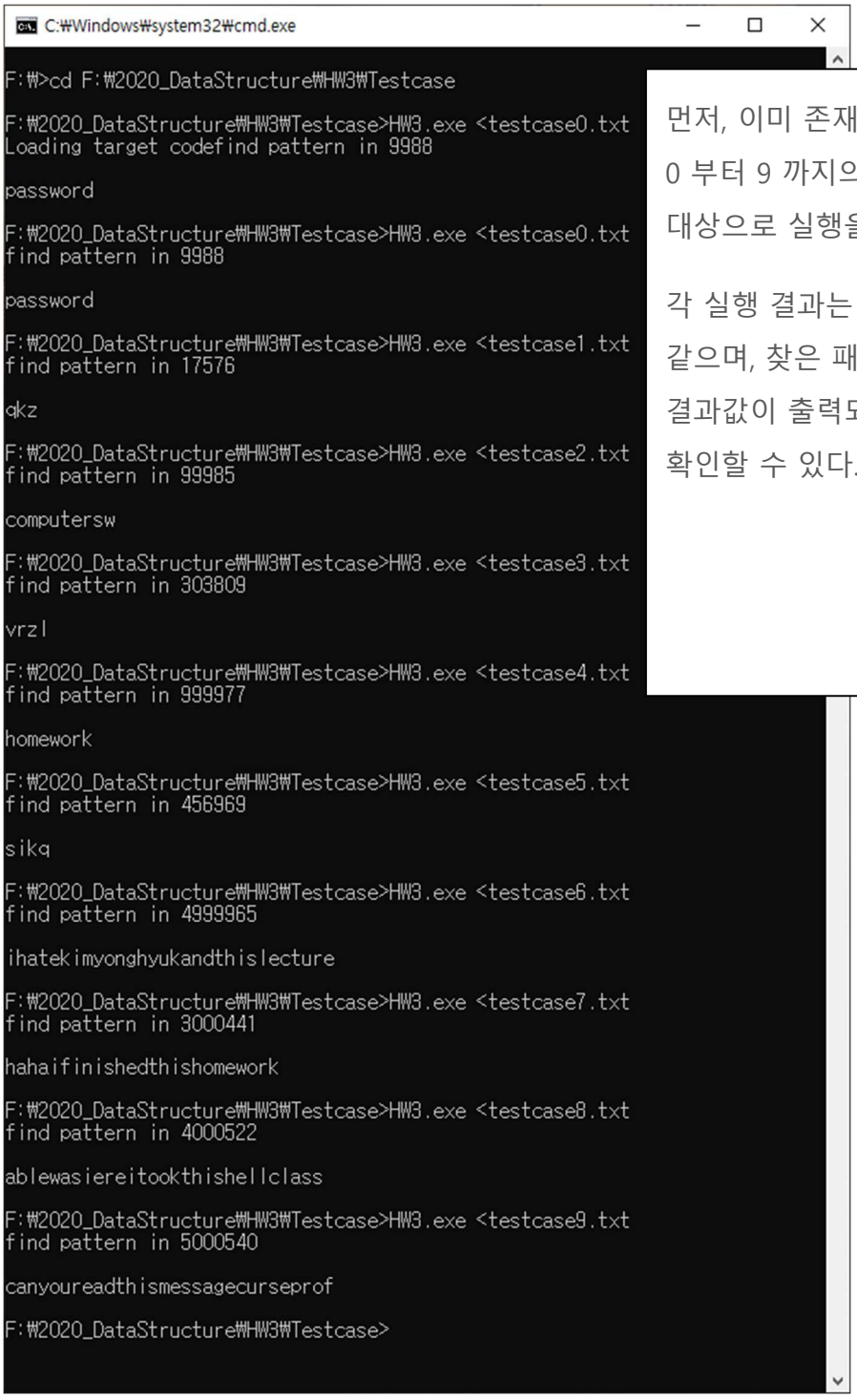
```
int index;
int max_index = std::distance( _First: m.begin(), _Last: m.end());
cout << "₩r" <<"find pattern in "<< max_index << endl;;
```

저장을 마친 이후에는, 사용자에게 암호 해독모드로 진입한 것을 알리기 위해 저장된 set 의 총 개수를 출력한다(std::distance 를 사용하여 처음과 끝을 전달, 반복자가 탐색해야 할 범위를 사용자에게 알려주기 위해 출력한다.)

```
for(it = m.begin(); it !=m.end();it++)
{
    //index = std::distance(m.begin(), it);
    if((it->second) > counter) // iterator 탐색
    {
        counter = it->second;
        ans = it->first; // 해당키값의 문자열
    }
    //cout <<"₩r"<< "[" << index+1 << "/" << max
}
cout << endl;
return ans;
```

이후, unordered_map 의 iterator , it 를 통해 암호탐색을 루프를 돌며 실행한다. Counter 는 해당 암호가 반복한 횟수로 set 의 second 값(키값)과 비교 연산을 실행한다. 만일 다른 set 의 최다 빈도수가 더 높게 갱신된다면, (it->second > counter) 새로운 정답 ans 를 갱신하고 다시 루프를 돈다. 기본적으로 최대값, 최소값을 갱신하는 예제의 알고리즘을 사용하였으며, it 를 사용하여 탐색속도에 큰 저하가 없을것으로 예상되어 해당 구조를 채택하였다. 루프를 다 돈 이후에는 ans 로 저장되어 있는 first 값(string)을 반환한다.

ㅇ test case!



먼저, 이미 존재하는 testcase 0 부터 9 까지의 txt 파일을 대상으로 실행을 해 보았다.

각 실행 결과는 왼쪽 그림과 같으며, 찾은 패턴 수와 결과값이 출력되는 것을 확인할 수 있다.

```
2
aabvjiiwlvdilvaiviaabiebvibvilbviabaabiqeofbqyasvhjfajaamcxzbvjbiequaabjvibebjajbvjkbkaawtbcbmzbiaahiwfhjqlbdkjbv,abdvkj
aauehdcjbzbvofehabaajiefhjolvnkldsvblaaaajbvjkebvjkab,hyfebjkaaafhiweohesbaakdshzncxzb,vbyfioqaaxjchcqhaaahhfhlaaaaaaaac
jlbjbbvlknkaa
find pattern in 135

aa
```

주어진 조건은 N 이 최소 3 인 경우이지만, 간략한 테스트를 위해 임의로 의미 없이 친 문자열에서 의도적으로 ‘aa’를 여러군데 삽입하여 ‘aa’의 결과가 나오도록 유도해 보았다. 역시, 정상적으로 ‘aa’값이 출력되는 것을 확인 가능 하였다.

```
3
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccccccccc
cccccccccccccccccccccccccccccccccdddddddddddddddddddddddddddddddddddddddeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
find pattern in 13e

bbb
```

이번에는 find pattern 이 정상적으로 동작하는 것을 확인하기 위해 한 종류의 알파벳을 연속해서 입력하는 것으로 패턴의 수를 적게 유지하여 13 개가 나오는지 검증해 보았다. (aaa, aab, abb,bbb,bbc,bcc,ccc....dee,eee 의 13 가지) 그 결과 13 은 정상적으로 출력되었지만, 현재 find pattern in 문장을 출력하는 문장이 “₩r”을 통해 헤드를 콘솔 처음 부분으로 돌리기 때문에, `cout << "₩r"<< "Loading target code";` 문장이 너무 짧아, 상단의 로딩 문구의 마지막 e 알파벳이 덧씌워지지 않아 ‘13e’라는 의도치 않은 출력이 발생하였다. 일반적인 사용처에서는 pattern 의 개수가 100 단위는 넘을것으로 예상이 되어 특별한 조치를 취하지는 않았다.

ㅇ 고찰

이번과제는 연산이 복잡하지는 않지만, 입력되는 string 의 길이가 길어질수록, 실행시간이 크게 늘어나 과제 초반에는, 빈 콘솔창을 계속 바라보며 기다리는 것이 힘들었다. 그래서 사용자가 진행 상황을 알 수 있도록 로깅 함수를 나름대로 추가해 보는 시도를 하였다.

```
void printProgress(int count , int EOS)
{ //count 현재 진행된 작업 // EOS 최대값
    char bar = '=';
    char blank = ' ';
    int LEN = 20; //프로그래스 바 길이

    int bar_count; // 바 갯수 저장
    float percent; // 퍼센트 저장

    float tick = (float)100 / LEN; //1틱의 백분율

    if (count <= EOS)
    {
        cout << "₩r" << count << "/" << EOS << "[";
        percent = (float)count / EOS * 100; // 퍼센트 연산
        bar_count = percent / tick; // 프로그래스 바 갯수 계
        for(int i=0; i<LEN; i++)
        {
            if (bar_count > i) cout << bar;
            else               cout << blank;
        }
        cout << "]";
        cout.flush();
    }
```

처음에는 간단한 함수 형태로 i 의 값과 EOS 값을 인자로 받아와서 프로그래스 바의 형태로 출력하는 함수 printProgress 를 만들어 실행해 보았다. 하지만 문자열을 불러오거나 , 최다 빈출 패스워드를 찾는 연산보다 해당 프로그래스 바를 출력하는데 더 시간이 오래걸려 프로그램 시간이 매우 느려져 포기하였다.

```
}
//cout <<"₩r"<< "[" << index+1 << "/" << max_index << "]";
```

두번째 시도로는 decoder 함수 내부에서 로깅이 가능하도록 iterator 가 작동할 때 출력을 갱신하려고 시도해 보았다. 이 역시 콘솔 출력에 더 많은 시간이 걸려 프로그램이 매우 느려졌다. 조사 결과 cout 과 cin 에 걸리는 시간은 일반적인 printf 와 scanf 보다 크고, 이를 해결하기 위해선 멀티프로세싱을 하거나 cin.tie(null)처럼 cout 과 cin 이 연결되어있는 것을 풀어주는 것이 필요하다는 것을 알았다. 하지만 이런 시도해도 불구하고 콘솔 출력속도가 크게 해결되지는 않아 현재의 구조인 로딩/ 서칭 의 분기마다 콘솔에 한번씩만 출력해 주는 구조로 변경하였다.