알고리즘 설계와 분석(CSE3081-02)

HW3

20191286 김나현

1. Project 1

첫 번째 프로젝트는 동적 계획법(dynamic programming)을 이용하여 문자열이 주어졌을 때, 가장 긴 회문(palindrome)을 만들 수 있는 부분 문자열, 즉 Longest palindromic subsequence(LPS)를 찾는 것이다. 이때, 부분 문자열, subsequence는 string과 달리 연속되지 않은 문자열도 포함하므로 이를 고려해야 한다. LPS를 찾는 코드를 구현하기 위해 파일을 열어 X라는 문자열 배열에 주어진 문자열을 저장하고, LPS 함수를 호출하였다. LPS 함수는 void LPS(const int m, const char\* X, int \*LPS\_length, char \*\*LPS\_string);과 같이 선언되므로 void LPS 함수 내부에서 LPS\_length 변수의 주소가 가리키는 공간에 함수에서 구한 LPS 길이를 저장하고 LPS 부분 수열을 따로 배열에 저장하여 LPS\_string이 가리키는 공간에 이 배열의 포인터를 저장하여 반환한다. 동적 계획법을 사용하기 위해서는 dp table을 만들어야 하므로 전역 변수로 int dp[1024][1024];라고 선언해주고 입력 받은 문자열의 크기를 저장하기 위해 int len;라고 선언해주며 LPS를 구성하는 문자를 저장하기 위한 문자열 포인터 lps를 char\* lps;라고 선언해준다. LPS 함수 안에서는 lpslen라는 정수형 변수를 0으로 선언과 동시에 초기화하고, 이중 for loop를 돌면서 i가 0부터 len까지 변해갈 때, j가 0부터 len까지 바뀌면서 dp[i][j]를 모두 0으로 초기화해준다. 그런 다음, 주어진 문자열 X의 inverse된 형태의 문자열 inverseX 포인터를 선언하고 동적 할당을 이용해 len 크기의 메모리를 갖도록 한다. LPS 함수에서 가장 중요한 부분은 dp table을 채우는 것인데 dp[][]의 모든 원소를 채워야 하므로 시간 복잡도 O(n^2)가 걸리고 마찬가지로 이중 for loop로 구현이 가능하다. i가 1부터 len까지 변해갈 때, j 역시 1부터 len까지 변해가며 dp table을 채우는데 X[i-1]와 inverseX[j-1]가 서로 같으면 dp[i][j]를 dp[i-1][j-1]에 1을 더한, dp[i-1][j-1]+1로 설정해주고 두 값이 같지 않으면 dp[i][j]를 dp[i-1][j]와 dp[i][j-1] 중 큰 값으로 설정한다. 이중 for loop를 도는 와중에 dp[i][j]가 lpslen보다 크면 lpslen를 dp[i][j]로 바꾸어 LPS의 길이를 저장한다. 이중 for loop를 다 돌면 LPS의 길이를 알 수 있지만 LPS를 구성하는 문자를 알 수 없으므로 이중 for loop를 또 한 번 돌면서 LPS를 구성하는 문자를 lps라는 문자열 배열 포인터에 idx 정수형 변수를 이용하여 배열처럼 저장한다. 지역 변수로 int idx=lpslen, column = len;라고 선언하고 i가 len부터 1까지 변해갈 때, j는 column부터 1까지 변해가며 이중 for loop를 돈다. 이때, dp table이 lpslen에서 lpslen-1, lpslen-1에서 lpslen-2…로 변해가는 지점을 찾아서 lps 배열에 저장할 것이므로 dp[i][j]가 idx이고, dp[i][j]를 기준으로 왼쪽, 위쪽, 대각선 앞이 모두 idx-1이면 lps[--idx]에 X[i-1]을 저장하고, column을 j-1로 바꿔주고 break를 통해 for loop을 빠져나간다. 만약, 그렇지 않고 dp[i-1][j], 즉 dp table에서 dp[i][j]와 위쪽이 같으면 i--; j++;을 해줘서 열은 변하지 않되, 행을 줄여주며 탐색을 한다. idx가 0이면 이중 for loop을 break를 통해 빠져나오고 \*LPS\_string=lps; \*LPS\_length=lpslen;를 통해 LPS 함수를 통해 반환해야 할 값들을 저장해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

config\_LPS.txt 파일을 위와 같이 만들어 저장하고, 해당 파일이 들어있는 디렉토리에 테스트 케이스로 주어진 input1.bin과 intput2.bin 파일을 저장한 후 코드를 실행시키면 다음과 같은 결과가 나온다. 이는 input1.bin의 NURSESAFRUN의 LPS인 NURSESRUN과 intput2.bin의 BBABCBCAB의 LPS인 BABCBAB가 올바르게 나오고 있음을 보여준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Project 2

두 번째 프로젝트는 동적 계획법을 이용하여 두 개의 카드 묶음 배열과 카드를 섞었을 때의 배열이 주어졌을 때, 두 개의 카드 묶음을 가지고 같은 카드 묶음에 있는 순서 배열은 깨뜨리지 않으면서 두 개의 카드 묶음을 섞어서 만들 수 있는지를 확인하는 코드를 구현하는 것이다. 두 개의 카드 묶음은 1000장을 넘지 않는다고 하였으므로 다음과 같이 매크로와 전역 변수를 선언해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이때, whatToCheck 배열은 commans\_3\_2.txt 파일 안에 적혀 있는 입력 파일(input.txt)을 열었을 때 두 개의 카드 묶음을 섞어서 만들 수 있는지 판단할 배열이 저장하기 위한 배열이다.

다음과 같이 input.txt 파일의 파일 스트림을 이용하여 한 줄씩 정수형 포인터 intArr에 저장하면 되고 input.txt 각 줄의 첫 번째 정수는 해당 줄에 적힌 글자 수이므로 \*num에 저장한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

입력 파일에서 첫 번째, 두 번째, 세 번째 줄에 대해 initialize 함수를 호출하여 r, l, whatToCheck 배열을 초기화고 이중 for loop을 이용해 dp table을 0으로 초기화하고 checkPossibility라는 함수를 호출하여 whatToCheck 배열이 두 개의 카드 묶음을 섞어서 가능한 조합인지 확인한다. checkPossibility 함수는 다음과 같이 구현할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

#define BAD 1라고 선언하였으므로 BAD가 1이었기 때문에 초기 상태에서는 dp table에서 모든 원소가 GOOD으로 초기화된 것과 동일하므로 whatToCheck[currIdx]가 r[idx1]과 같으면 checkPossibility(currIdx+1, idx1+1, idx2) 함수를 재귀적으로 호출하여 GOOD이 반환되면 dp[idx1][idx2]가 GOOD으로 세팅되고, BAD가 반환되면 마찬가지로 dp[idx1][idx2]을 BAD로 세팅한다. 마찬가지로 whatToCheck[currIdx]가 l[idx2]와 같으면 checkPossibility(currIdx+1, idx1, idx2+1) 함수를 재귀적으로 호출하여 GOOD이 반환되면 dp[idx1][idx2]가 GOOD으로 세팅되고, BAD가 반환되면 dp[idx1][idx2]를 BAD로 세팅한다. checkPossibility 함수에 젤 위에 있는 if문 두 개는 basis로, dp[idx1][idx2]가 BAD이면 이미 dp[idx1][idx2]을 확인해본 것이므로 바로 BAD를 반환하고 currIdx가 inputSize와 같으면 whatToCheck 배열의 inputSize-1번째 원소까지 확인한 것이므로 GOOD을 반환한다.

따라서, checkPossibility 함수에서 GOOD을 반환하면 가능하다는 의미이지만 값으로는 0을 의미하므로 checkPossibility에서 반환한 값에 느낌표(!)를 붙여 0이면 1을, 1이면 0을 출력해야 된다.

다음은 서로 다른 두 개의 input98.txt, input99.txt에 대한 실행 결과이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Project 3

세 번째 프로젝트는 그리디 알고리즘인 허프만 코딩(Huffman coding) 방법을 이용하여 주어진 파일 안에 있는 모든 문자들을 이용하여 가장 적은 비트로 표현할 수 있는 binary code를 찾는 것이다. 우선 구조체 Node를 다음과 같이 선언하여 파일로부터 얻은 문자에 대한 정보를 저장하기 위한 변수형으로 사용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

메인 함수에서 for loop를 이용해 지역 변수 Node arr[1000];와 전역 변수 Node arrNode[128];를 초기화해주고 P3\_input\_ASCII.txt 파일을 열어 while문 안에서 fgetc 함수를 이용하여 문자 하나씩을 읽어 P3\_input\_ASCII.txt 파일 안에 있는 모든 문자를 순서대로 저장하는 배열인 orig\_char에 저장하고, arr[ch].freq가 0이면 문자열에서 사용한 전체 문자의 개수를 의미하는 n을 하나 증가시키고 arr[ch].freq과 arrNode[ch].freq를 하나씩 증가시킨다. fgetc로 얻은 문자가 EOF이면 while문을 break로 빠져나오고 P3\_input\_ASCII.txt 파일을 닫는다. 그런 다음, n을 이용하여 전역 변수 Node의 이중 포인터 heap(Node \*\*heap;)에 (n+1) 크기의 메모리를 동적 할당해주고 for loop를 돌면서 arr[i].freq가 0이 아니면 index를 1부터 하나씩 증가(heap이기 때문에 인덱스 0은 사용하지 않음)시키면서 heap을 채워준다. Freq를 기준으로 minheap을 만들어야 하는데 지금은 minheap이 아니므로 for loop를 i가 n/2부터 0까지 돌면서 adjust\_minheap(i, n) 함수를 이용하여 minheap을 만들어준다. 그럼 n개의 Node\*형의 원소로 구성된 minheap이 생성된 것이니까 n-1번의 for loop를 돌면서 Node\* u와 \*v에 delete\_minheap(&n) 함수로부터 반환된 가장 작은 원소를 저장하고 Node\* w를 w=&arr[index++]라고 만든 후, w의 freq를 u와 v의 freq 합으로, w의 left를 u, w의 right를 v로 설정한 뒤 다시 minheap에 insert\_minheap(w, &n) 함수를 이용하여 삽입한다. For loop이 종료되면 모든 원소들을 노드로 갖는 하나의 트리의 root node만이 남을 것이므로 이 root node를 Node \*p에 delete\_minheap(&n) 함수를 이용해 저장한다. 지금까지 사용한 adjust, delete, insert 함수는 heap을 구현할 때 사용하는 일반적인 함수지만 이 알고리즘에서는 이 외에도 encoding을 위한 특별한 함수가 필요하다. encoding라고 이름 지은 이 함수는 함수 호출 시 Node \*ptr를 매개 변수로 받아 ptr가 NULL이 아니고 ptr->left가 NULL이 아니면 for loop을 이용해 ptr->left->ch에 ptr->ch에 있는 모든 문자를 복사하고 ptr->left->ch에 0을 저장한다. 그런 다음 ptr->left를 이용해 재귀적으로 encoding 함수를 다시 호출하고, ptr->right가 NULL이 아니면 마찬가지로 for loop를 이용해 ptr->right->ch에 ptr->ch에 있는 모든 문자를 복사하고 ptr->right->ch에 1을 저장한다. 그런 다음에 ptr->right을 이용해 재귀적으로 encoding 함수를 다시 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 함수를 통해 arr[]의 원소에 encoding할 codeword가 저장되므로 ASCII 코드 순서대로codeword를 출력하기 위해 P3\_output\_codewords.txt 파일을 out이라는 이름의 파일 스트림으로 열어 for loop(변수 i)를 이용해 arr[i].freq가 0이 아닐 때, out을 통해 arrNode[arr[i].symbol].symbol을 출력하고 arr[i].ch.size()번의 for loop(변수 j)를 돌면서 arrNode[arr[i].symbol].ch에 저장하고 out을 통해 arr[i].ch[j]를 출력한다. 그런 다음 for loop를 돌면서 arrNode[i].freq가 0이 아닐 때 해당 문자의 빈도와 codeword의 곱을 sum이라는 변수에 더해간다. 마지막으로 파일 스트림 File \*bin=fopen(“P3\_output\_encoded.bin”,”wb”);를 이용하여 출력 파일2에 들어가야 할 내용을 채워넣는다.

다음 결과를 확인하면 빈도에 따라 입력 데이터를 최소 개수의 비트로 압축할 수 있는 codeword가 잘 나타난 것을 알 수 있고, P3\_input\_ASCII.txt 파일로부터 얻은 텍스트를 0과 1만을 이용한 binary code로 encoding된 모습 또한 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명