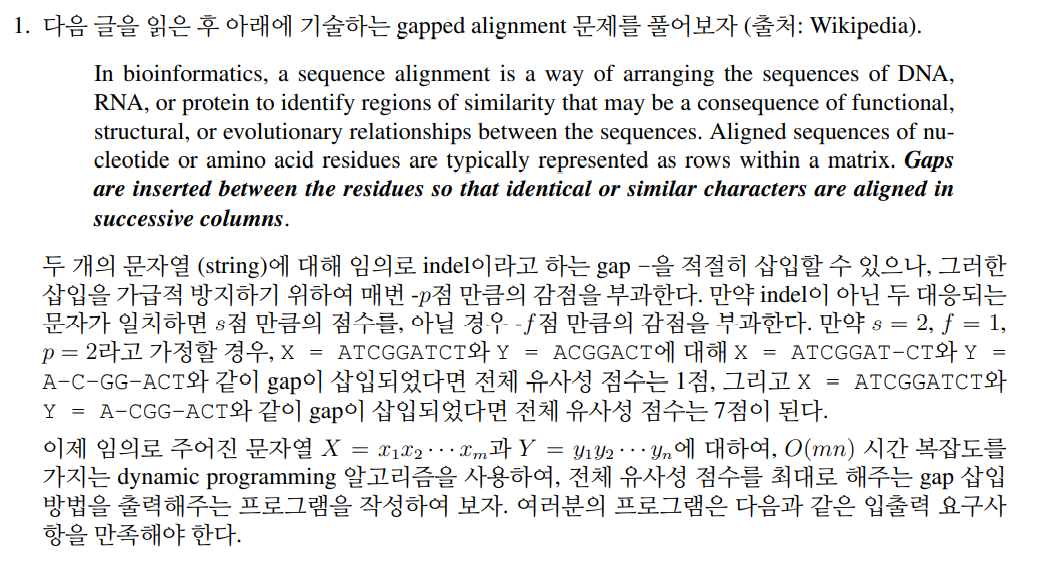
알고리즘 설계와 분석 과제2 보고서

# 컴퓨터공학과 20192134 김은원

본인이 해결한 각 문제에 대하여 HW1 S2021\*\*\*\*.{hwp,doc,txt}와 같은 이름의 보고서에 optimal substructure를 간략히 기술한 후, 프로그램 및 데이터와 함께 제출하라 (파일명에 학번을 기술 또는 조교가 공지하는 방식으로 제출).

1번 문제)



Optimal substructure)

mat(i,j)는 X의 처음부터 X의 i까지, Y의 처음부터 Y의 j까지 비교했을 때 최대 점수를 뜻한다.

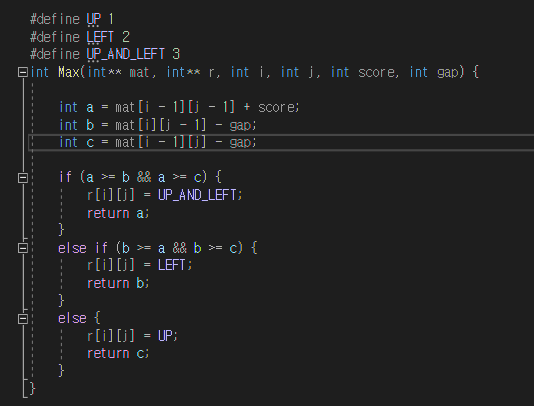
mat (i,j) = { if i=0, mat (i,j) = -2j

j=0, mat (i,j)=-2i

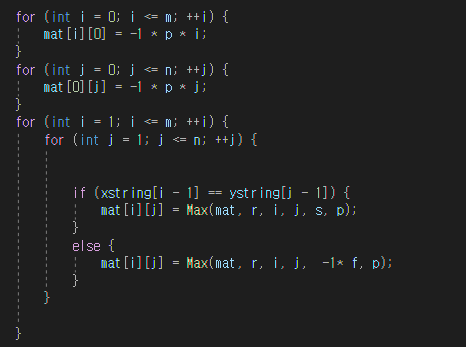
If i>0 and j>0 max{ mat(i-1,j-1) + s(ai,bi) , mat (i,j-1) – gap, mat(i,-1,j) – gap }

\*s(ai,bi)는 ai === bi 이면 match 점수인 s점 같지 않으면 -f 이다.

Max를 구하는 함수는 다음과 같이 구현했다.

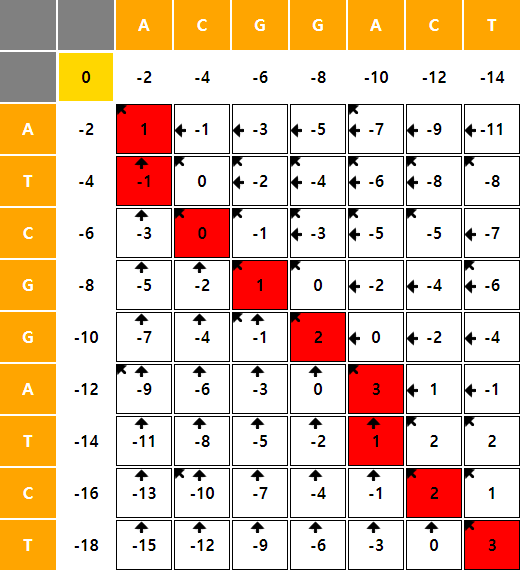


optimal substructure를 구현한 코드는 다음과 같다. 시간 복잡도는 O(n2)이다.



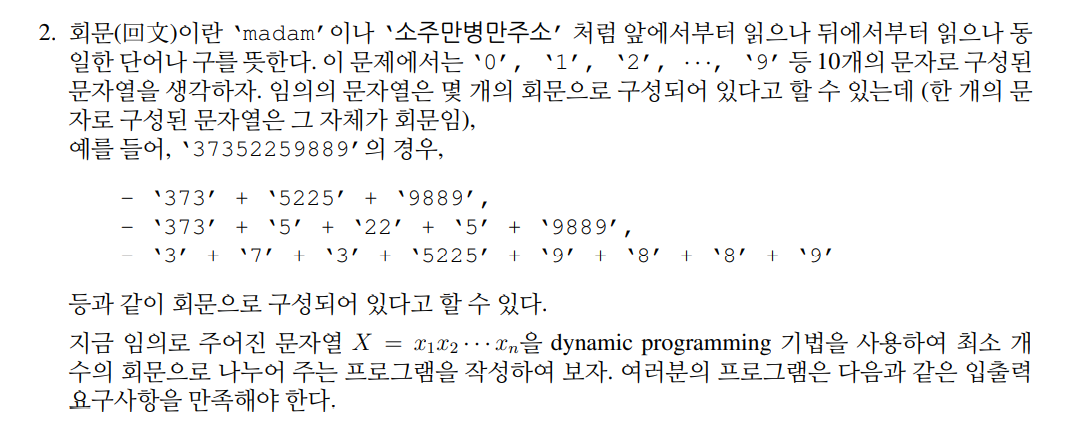
r[i][j]는 i행 j열에서 가야하는 방향을 의미한다.

ATCGGATCT 와 ACGGACT를 비교하는 r[i][j]와 mat[i][j]를 나타낸 표이다.



점수는 S[n][n] 값이 된다.

2번 문제)



Optimal substructure)

주어진 문자열의 개수가 n개라면 2차원 배열크기가 n x n인 DP 배열을 만들어 DP[시작][끝]에는 시작 인덱스부터 끝 인덱스까지 회문이면 1을, 아니면 0을 넣는다.

입력문자열은 str이라고 한다.

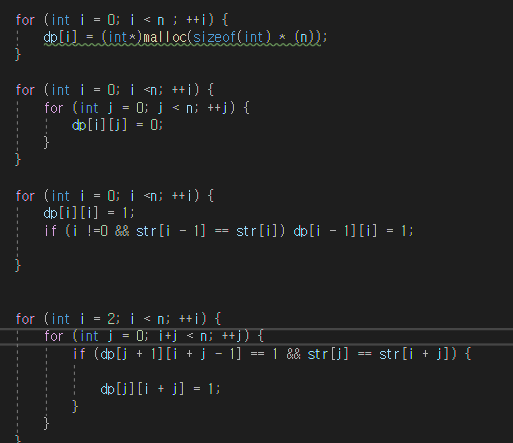
DP = { if i=j , DP[i][j] = 1

if i !=1 and str[i-1] =str[i] , DP[i-1][i] =1

if str[i]=str[j] and DP[i+1][j-1]=1, DP [i][j] =1

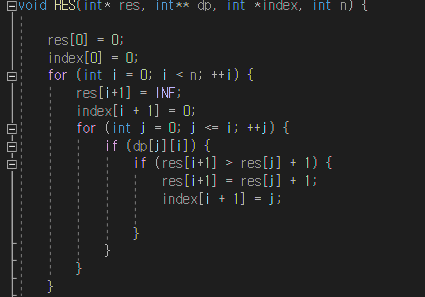
}

위 최적해를 구하는 공식을 프로그래밍한 코드이다.



위 최적해를 구하기 위해 수행한 프로그램의 시간 복잡도는 O(n2)이다.

RES함수는 최소 회문 개수를 구하는 함수이다.

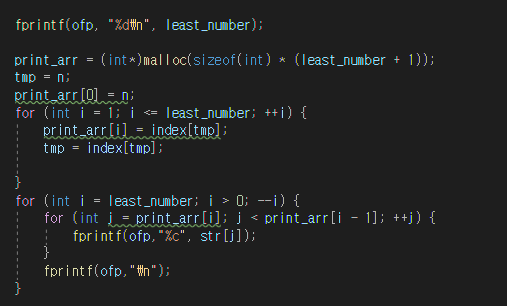
처음 res 값들을 모두 INF로 초기화한다. res[0]=0으로 한다. res 값은 다음과 같다.

res = { if i<n and j<=i , dp[j][i] ==1 이면 res[i+1] = max(res[i+1], res[j]+1) }

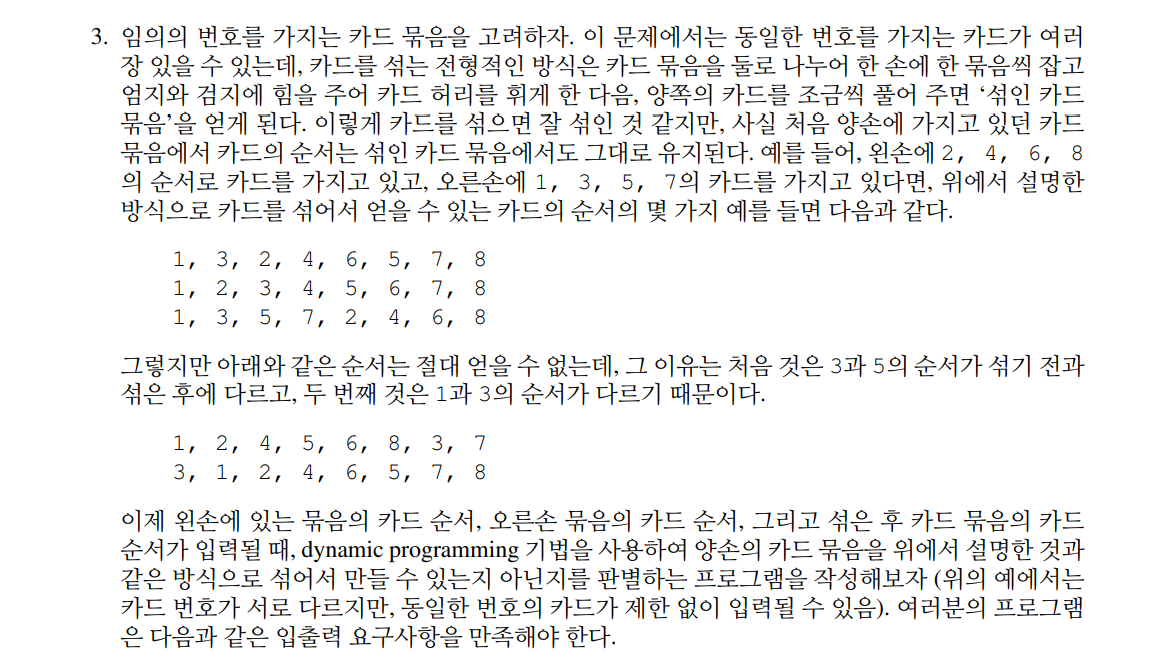
즉 j부터 i까지의 문자열이 회문이면 j-1 까지의 점수 + 1 이 i까지의 점수보다 작을 때 갱신하는 것이다.

이때 index라는 문자열은 회문 시작위치를 저장한다.

최소 회문 개수는 res[n]이 된다. index를 이용하여 최소 회문이 시작하는 문자열을 출력한다.



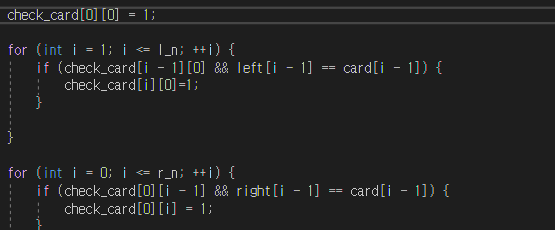
3번문제)



왼쪽 카드를 left, 오른쪽 카드를 right, 카드 순서가 맞는지 확인하기 위한 카드 문자열을 card, 왼쪽 카드의 문자열 개수를 l\_n, 오른쪽 카드 문자열 개수를 r\_n, 확인 카드 문자열 개수를 card\_n이라고 하자.

check\_card는 카드 순서가 되는지 확인하기 위한 2차원 배열이며 크기는 (l\_n +1) \* (r\_n + 1) 이다. check\_card[i][j]는 Left i까지, Right j까지 비교했을 때 순서가 가능한 지를 의미한다. 최종적으로 check\_card[l\_n][r\_n]=1 이면 1을 출력하고 0이면 0을 출력한다.

우선 card와 left, right가 처음부터 같을 때까지 1을 채운다.



Left=2,4,6,8

Right=1,3,5,7

card = 1,3,2,4,6,5,7,8 이면

card의 처음 1,3은 right의 1,3과 같으므로

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 |  |  |  |  |
| 2 | 0 |  |  |  |  |
| 3 | 0 |  |  |  |  |
| 4 | 0 |  |  |  |  |

위처럼 채워진다.

optimal substructure)

check\_card = { if check\_card[i-1][j]=1 and card[i+j-1] = left[i-1] , check\_card[i][j]=1

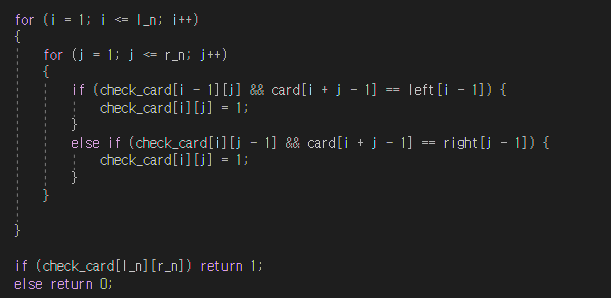
if check\_card[i][j - 1]=1 and card[i + j - 1] == right[j – 1], check\_card[i][j]=1

} (i=1~l\_n and j=1~r\_n일 때)

위 공식을 대입한 check\_card는 최종적으로

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 |

이 된다. 다음은 위 공식을 프로그래밍한 코드이다.



dp 함수의 시간복잡도는 O(N2)이다.