Homework set 3 Lorenzo Esposito M63001417

martedì 13 dicembre 2022 20:47

Problema 3.1

Il palindromo è una sequenza di caratteri che, letta al contrario, rimane invariata. Ad esempio, GAG e MADAM sono palindromi, ma ADAM no. Assumiamo che anche la stringa vuota sia un palindromo. Da qualsiasi stringa non palindromica, si può sempre ottenere una sotto-sequenza palindromica togliendo alcune lettere: ad esempio, data la stringa ADAM, si può rimuovere la lettera M e ottenere ADA. Scrivere un programma per determinare la lunghezza del palindromo più lungo che puoi ottenere da una stringa rimuovendo zero o più caratteri. Il programma deve ricevere una stringa (di lunghezza inferiore a 1000 caratteri) e stampare la lunghezza del palindromo più lungo che si può ottenere rimuovendo zero o più caratteri.

```cpp
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
using namespace std;
// funzione che inizializza la matrice
void init(vector<vector<int>> p)
{
    for (int i = 0; i < p.size(); i++)
    {
        for (int j = 0; j < p.size(); j++)
        {
            p[i][j] = 0;
        }
    }
}
int algorithm(string A, int i, int j, vector<vector<int>> p)
{
    // caso base 1: se la stringa è vuota
    if (i > j)
    {
        return 0;
    }
    // caso base 2: se la stringa ha un solo carattere
    if (i == j)
    {
        return 1;
    }
    // caso in cui la soluzione è già in memoria
    if (p[i][j] != 0)
    {
        return p[i][j];
    }
    // caso nel cui il primo e l'ultimo carattere della stringa sono uguali e la soluzione non è in memoria
    if (A[i] == A[j])
    {
        // caso in cui due lettere sono già uguali
        p[i][j] = algorithm(A, i + 1, j - 1, p) + 2;
        // aggiungo i due caratteri uguali e richiamo la funzione ricorsiva
    }
    else
    {
        // caso in cui due lettere sono diverse
        p[i][j] = max(algorithm(A, i, j - 1, p), algorithm(A, i + 1, j, p));
        // calcolo il massimo tra la soluzione senza il primo carattere e quella senza l'ultimo carattere
    }
    return p[i][j];
}
void testcase1()
{
    string S = "ADAM";
    int n = S.length();
    vector<vector<int>> p(S.length(), vector<int>(S.length(), 0));
    init(p);
    cout << "La lunghezza della sottosequenza palindroma più lunga è "
         << algorithm(S, 0, n - 1, p);
}
void testcase2()
{
    string S = "MADAM";
    int n = S.length();
    vector<vector<int>> p(S.length(), vector<int>(S.length(), 0));
    init(p);
    cout << "La lunghezza della sottosequenza palindroma più lunga è "
         << algorithm(S, 0, n - 1, p);
}
void testcase3()
{
    string S = "MA";
    int n = S.length();
    vector<vector<int>> p(S.length(), vector<int>(S.length(), 0));
    init(p);
    cout << "La lunghezza della sottosequenza palindroma più lunga è "
         << algorithm(S, 0, n - 1, p);
}
int main()
{
    cout << "Testcase 1" << endl;
    // output previsto 3
    testcase1();
    cout << endl
         << "Testcase 2" << endl;
    // output previsto 5
    testcase2();
    cout << endl
         << "Testcase 3" << endl;
    // output previsto 1
    testcase3();
    return 0;
}
```

Spiegazione dell'algoritmo: L'algoritmo è basato sull' approccio top-down della programmazione dinamica, il primo step consiste nel trovare una sottostruttura ottima al problema la quale si basa sull' intervenire sui caratteri eliminabili dalla stringa, per trovare una sequenza palindroma. Abbiamo un caso base che consiste nel caso in cui la stringa è vuota in quel caso ritorniamo zero, mentre se la stringa è composta da un solo carattere ritorniamo uno. Il caso classico invece si costituisce in due sotto casistiche la prima è quando A[0]==A[l], in questo caso semplifichiamo il problema rimuovendo la prima lettera e l'ultima in seguito sommiamo 2 al risultato calcolato sulla rimanente parte della parola, se non ci troviamo in questa casistica calcoliamo il massimo tra la parola privata della prima e dell'ultima lettera e ritorniamo come risultato il massimo tra i due. Per abbassare la complessità rispetto alla soluzione naive è stata aggiunta una struttura dati basata sull'idea di salvare la soluzione in funzione degli indici i e j per costruire lo schema formato da ricorsione+memoization, in questo modo salviamo i risultati calcolati all'interno della matrice e nel caso si ripresenti lo stesso sotto problema estraiamo il risultato dalla struttura dati.

TESTCASE 1
"ADAM" => "ADA" => PALINDROMA => 3

TESTCASE 2
"MADAM" => PALINDROMA => 5

TESTCASE 3
"MA" => "M" oppure "A" => PALINDROMA => 1

Analisi della complessità: La complessità nel caso dell'approccio top down si calcola come #sottoproblemi*(tempo/sottoproblema), applicando la formula sul problema sopra esaminato la complessità sarà di O(n^2) la quale è una complessità polinomiale e quindi più bassa della complessità della soluzione naive la quale è esponenziale.