Task PALINDROMES

Explanation to the solutions

Slow solution: recursion

The program reads the given string into `s`. The helper function

```
bool isPalindrome(int i, int j)
```

returns `true`, when the substring of `s` with indices `i` through `j` is a palindrome.

The recursive function `rec(i,j)` finds how many cuts are necessary, so that the substring of `s` with indices `i` to `j` can be optimally split into palindromes. In the body of this function, all splits into two substrings are checked, and in `r` the minimum count is calculated:

```
int r = 1e9, c;
for (int k = i; k < j; k++)
  {
    c = 1 + rec(i, k) + rec(k + 1, j);
    r = min(r, c);
  }
```

Faster solution: recursion with memoization

The recursion from the previous program is used, adding a two-dimensional array `memo[i][j]` to store already calculated values from the function `rec[i][j]`. Initially, the array is loaded with values equal to the number −1 via `fill_n(&memo[0][0],N*N,-1);`

Faster solution: dynamic programming with bottom-up table filling

Two two-dimensional arrays `d[][]` and `isPalin[][]` are used:

– in `dp[i][j]` the number of cuts of the substring of `s` with indices from `i` to `j` is calculated, with which it is optimally divided into palindromes.

– `isPalin[i][j]` is evaluated to `true` or `false` depending on whether the substring of `s` with indices `i` through `j` is either a palindrome or not.

The calculation starts for indices `i` and `j`, for which the difference `j-i` is initially small (equal to `len-1`, where `len` in the outer loop gradually increases from 2 to `n`). First we calculate `isPalin[i][j]` and then, when `isPalin[i][j]==true` we set `dp[i][j]=0`, and in the opposite case we go through all partitions into two substrings and we calculate in `dp[i][j]` the optimal value:

```
dp[i][j] = 1e9;
 for (int k = i; k <= j - 1; k++)
  dp[i][j] =
    min(dp[i][j], 1 + dp[i][k] + dp[k+1][j]);
```

Finally we print `dp[0][n - 1] + 1`.

Fast solution: dynamic programming with precomputing

The idea of the previous program is used, but with precomputing in the Boolean array p[i][j] whether the substring of `s` with indices from `i` to `j` is a palindrome. This is done by the function `genp` using a double loop:

```
 void genp()
{
  int n = s.size();
  for (int i = 0; i < n; i++) p[i][i] = true;

  for (int len = 2; len <= n; len++)
  for (int i = 0, j = i + len - 1; j < n ; i++, j++)
   if (s[i] == s[j] && (len == 2 || p[i + 1][j - 1]))
    p[i][j] = true;
}
```

Емил Келеведжиев