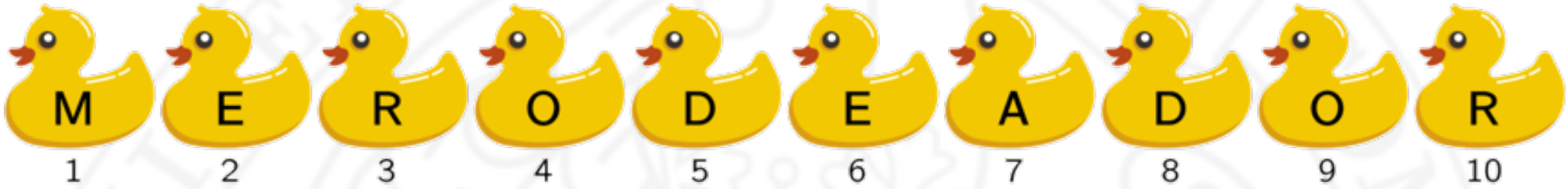
TIRO AL PATÍNDROMO

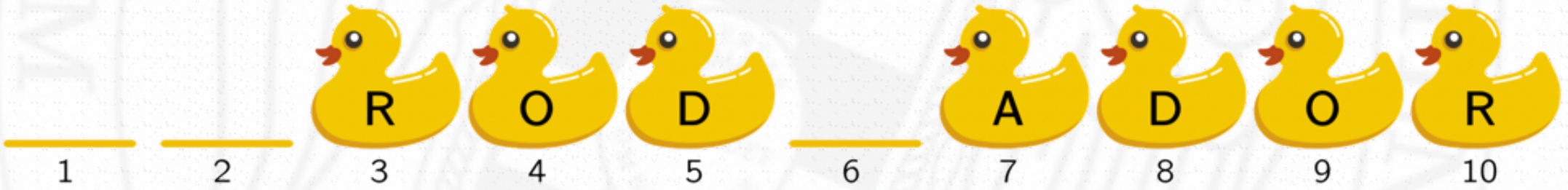
UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

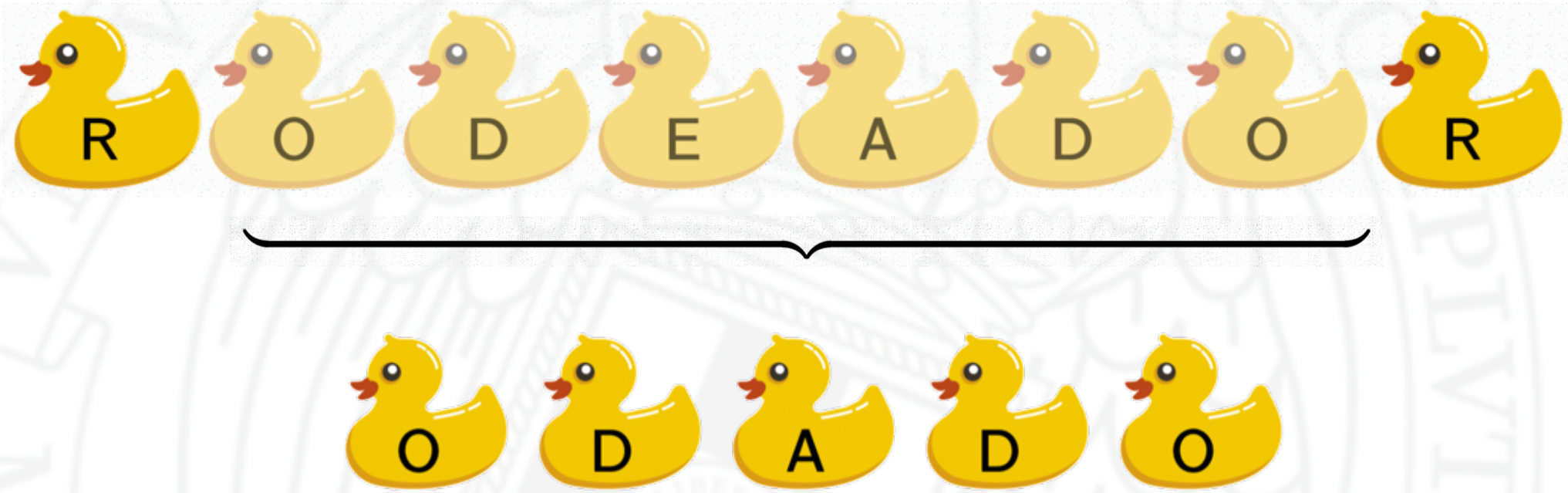
Tiro al patíndromo



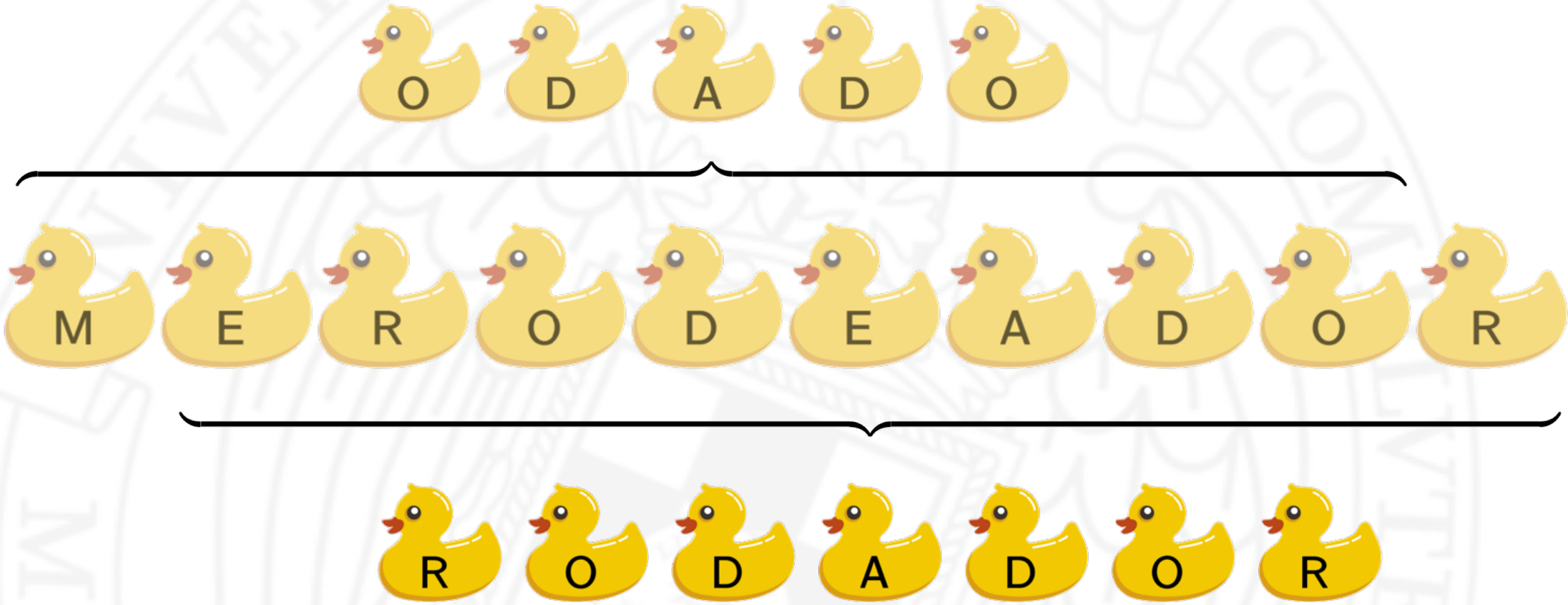
- Conseguir el palíndromo más largo tirando (si es necesario) algunos de los patitos:



Tiro al patíndromo



Tiro al patíndromo



Definición recursiva

patíndromo(i,j) = longitud del palíndromo más largo que podemos obtener con *patitos*$[i..j]$

i ... j

Definición recursiva

- Casos recursivos ($i < j$) :

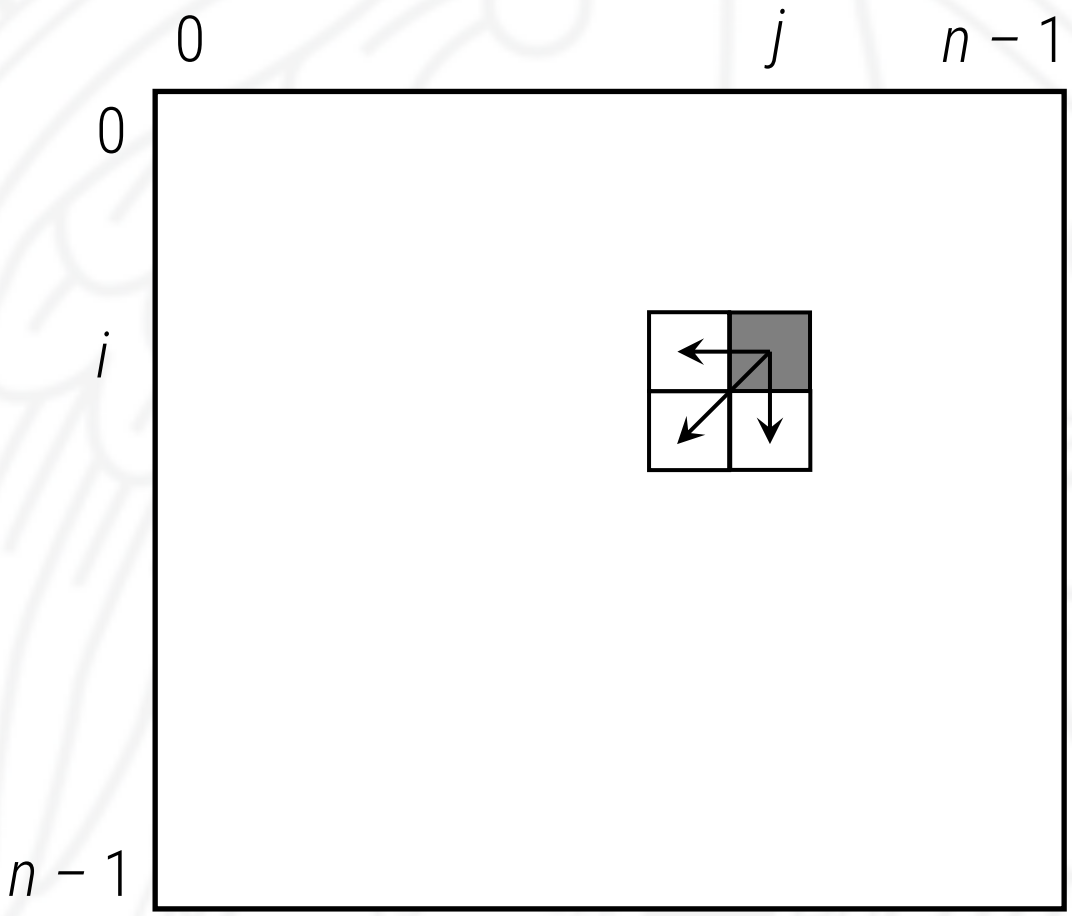
$$patíndromo(i,j) = \begin{cases} patíndromo(i+1,j-1)+2 & \text{si } patitos[i] = patitos[j] \\ \text{máx}(\ patíndromo(i+1,j), & \\ \qquad patíndromo(i,j-1)\ ) & \text{si } patitos[i] \neq patitos[j] \end{cases}$$

- Casos básicos:

$$\begin{aligned} patíndromo(i,i) &= 1 \\ patíndromo(i,j) &= 0 \quad \text{si } i > j \end{aligned}$$

- Llamada inicial: $patíndromo(0, n-1)$

Tabla



Implementación

```
// calcula la longitud del mayor palíndromo en patitos[i..j]
int patin_rec(string const& patitos, int i, int j, Matriz<int> & patin) {
    int & res = patin[i][j];
    if (res == -1) {
        if (i > j) res = 0;
        else if (i == j) res = 1;
        else if (patitos[i] == patitos[j])
            res = patin_rec(patitos, i+1, j-1, patin) + 2;
        else
            res = max(patin_rec(patitos, i+1, j, patin),
                      patin_rec(patitos, i, j-1, patin));
    }
    return res;
}
```

Implementación

```
// devuelve el palíndromo más largo en patitos[i..j]
string reconstruir(string const& patitos, Matriz<int> const& patin,
                   int i, int j) {
  if (i > j) return {};
  if (i == j) return { patitos[i] };
  if (patitos[i] == patitos[j])
    return patitos[i] + reconstruir(patitos, patin, i+1, j-1) + patitos[j];
  else if (patin[i][j] == patin[i+1][j])
    return reconstruir(patitos, patin, i+1, j);
  else
    return reconstruir(patitos, patin, i, j-1);
}
```

Implementación

```
// añade al final de sol el palíndromo más largo en patitos[i..j]
void reconstruir(string const& patitos, Matriz<int> const& patin,
                 int i, int j, string & sol) {
   if (i > j) return;
   if (i == j) sol.push_back(patitos[i]);
   else if (patitos[i] == patitos[j]) {
      sol.push_back(patitos[i]);
      reconstruir(patitos, patin, i+1, j-1, sol);
      sol.push_back(patitos[j]);
   } else if (patin[i][j] == patin[i+1][j])
      reconstruir(patitos, patin, i+1, j, sol);
   else
      reconstruir(patitos, patin, i, j-1, sol);
}
```

Implementación

```
string patitos;
cin >> patitos;
int n = patitos.length();

Matriz<int> patindromo(n, n, -1);
cout << patin_rec(patitos, 0, n-1, patindromo) << '\n';

cout << reconstruir(patitos, patindromo, 0, n-1) << '\n';

string sol;
reconstruir(patitos, patindromo, 0, n-1, sol);
cout << sol << '\n';
```