Lab 4

```
M = {{a}, {b, b}, {a, a, a}, {a, a, b}, {a, b, b},
    {a, a, a, b}, {a, a, b, a}, {a, a, b, a, b}, {a, a, b, b, b}};
```

```
Prefixes[M_] := Module[{list, i, aux},
                 módulo
    list = {{}}; aux = M;
    For[i = 1, i ≤ Length[aux], i++,
    para cada      longitud
     While[Length[aux[[i]]] > 0, AppendTo[list, aux[[i]]];
     mient··· longitud              añade al final
       aux[[i]] = Drop[aux[[i]], -1];];];
                  elimina
    Return[Union[list]];];
    retorna unión
```

Exercise 1 - Longest suffix of u contained in M

```
LongestSuffix[u_, M_] := Module[{i, word},
                         módulo
    word = u;
    While[Length[word] > 0 && ! MemberQ[M, word], word = Rest[word];];
    mient··· longitud          ¿contenido en?         todos excepto el prin
    If[MemberQ[M, word], Return[word], Return[False]];
    si ¿contenido en?    retorna       retorna falso
   ];
```

```
LongestSuffix[{}, M]
```

```
False
```

Exercise 2 - Generate a dictionary automaton for M

```
DictionaryAutomaton[M_] := Module[{A, s, i, j},
                           módulo
   (*Automaton: {Q, ∑, δ, q0, F}*)
   A = {Prefixes[M], Union[Flatten[M]], {}, {}, M};
                     unión  aplana
   (*Generate list of transitions: for all states try to add each letter*)
   For[i = 1, i ≤ Length[A[[1]]], i++,
   para cada      longitud
    For[j = 1, j ≤ Length[A[[2]]], j++,
    para cada      longitud
     AppendTo[A[[3]],
     añade al final
       {A[[1, i]], A[[2, j]], LongestSuffix[Append[A[[1, i]], A[[2, j]]], A[[1]]]}
                                           añade
      ];
    ];
    (*Add to F any state of Q whose suffix is included in F*)
    If[i > 1 && ! LongestSuffix[Rest[A[[1, i]]], A[[5]]] ≠ False,
    si                          todos excepto el primero    falso
     (*TRUE*), (*FALSE*), (*SCHRÖDINGER*) AppendTo[A[[5]], A[[1, i]]];];
                                          añade al final
   ];
   A[[5]] = Union[A[[5]]];
            unión
   Return[A];
   retorna
  ];
```

```
DictionaryAutomaton[M]
```

```
{{{}, {a}, {b}, {a, a}, {a, b}, {b, b}, {a, a, a}, {a, a, b}, {a, b, b},
   {a, a, a, b}, {a, a, b, a}, {a, a, b, b}, {a, a, b, a, b}, {a, a, b, b, b}}, {a, b},
  {{{}, a, {a}}, {{}, b, {b}}, {{a}, a, {a, a}}, {{a}, b, {a, b}}, {{b}, a, {a}},
   {{b}, b, {b, b}}, {{a, a}, a, {a, a, a}}, {{a, a}, b, {a, a, b}}, {{a, b}, a, {a}},
   {{a, b}, b, {a, b, b}}, {{b, b}, a, {a}}, {{b, b}, b, {b, b}}, {{a, a, a}, a, {a, a, a}},
   {{a, a, a}, b, {a, a, a, b}}, {{a, a, b}, a, {a, a, b, a}}, {{a, a, b}, b, {a, a, b, b}},
   {{a, b, b}, a, {a}}, {{a, b, b}, b, {b, b}}, {{a, a, a, b}, a, {a, a, b, a}},
   {{a, a, a, b}, b, {a, a, b, b}}, {{a, a, b, a}, a, {a, a}}, {{a, a, b, a}, b, {a, a, b, a, b}},
   {{a, a, b, b}, a, {a}}, {{a, a, b, b}, b, {a, a, b, b, b}}, {{a, a, b, a, b}, a, {a}},
   {{a, a, b, a, b}, b, {a, b, b}}, {{a, a, b, b, b}, a, {a}}, {{a, a, b, b, b}, b, {b, b}}},
  {}, {{a}, {a, a}, {b, b}, {a, a, a}, {a, a, b}, {a, b, b}, {a, a, a, b},
   {a, a, b, a}, {a, a, b, b}, {a, a, b, a, b}, {a, a, b, b, b}}}
```

Exercise 3 - Analyze a word using a dictionary automaton

```
ScanWordDictionary[A_, x_] := Module[{state, i, curr, pos, s},
                                     └módulo
    state = A[[4]];
    pos = {};
    For[i = 1, i ≤ Length[x], i++,
    └para cada   └longitud
      state = Cases[A[[3]], {state, x[[i]], _}][[1, 3]];
              └casos
      If[MemberQ[A[[5]], state],
      └si └¿contenido en?
        (*AppendTo[pos,i-Length[state]+1];*)
          └añade al final      └longitud
        For[s = state,
        └para cada
          Length[s] ≥ Length[A[[5, 1]]], s = LongestSuffix[Rest[s], A[[5]]],
          └longitud   └longitud                               └todos excepto el primero
          If[s, (*TRUE*), Break;, AppendTo[pos, i - Length[s] + 1];];
          └si              └finaliza i·· └añade al final    └longitud
        ];
      ];
    ];
    Return[Sort[pos]];
    └retorna └ordena
   ];
```

```
ScanWordDictionary[DictionaryAutomaton[M], {a, a, b, a, a, a}]
```

```
{{1, {a}}, {1, {a, a}}, {1, {a, a, b}}, {1, {a, a, b, a}}, {2, {a}},
 {4, {a}}, {4, {a, a}}, {4, {a, a, a}}, {5, {a}}, {5, {a, a}}, {6, {a}}}
```

```
ScanWordDictionaryPro[M_, x_] := Module[{A, state, i, curr, pos, s},
                                 módulo
  A = DictionaryAutomaton[M];
  state = A[[4]];
  pos = {};
  For[i = 1, i ≤ Length[x], i++,
  para cada       longitud
   state = Cases[A[[3]], {state, x[[i]], _}][[1, 3]];
           casos
   If[MemberQ[A[[5]], state],
   si  ¿contenido en?
    For[s = state, Length[s] ≥ Length[A[[5, 1]]], s = LongestSuffix[Rest[s], M],
    para cada      longitud     longitud                            todos excepto el p
      If[MemberQ[M, s],
      si  ¿contenido en?
        AppendTo[pos, {i - Length[s] + 1, s, i}];,
        añade al final     longitud
        AppendTo[err, {s, False}]; Break;,
        añade al final    falso     finaliza iteración
        AppendTo[err, {s, "Other"}]; Break;
        añade al final               finaliza iteración
       ];
     ];
   ];
  ];
  Return[pos];
  retorna
 ];
```

```
ScanWordDictionaryPro[M, {a, a, b, a, a, a}]
```

```
{{1, {a}, 1}, {2, {a}, 2}, {1, {a, a, b}, 3}, {1, {a, a, b, a}, 4},
 {4, {a}, 4}, {5, {a}, 5}, {4, {a, a, a}, 6}, {6, {a}, 6}}
```

Testing

```
err = {}
```

```
{}
```

```
err
```

```
{{{a, a}, False}, {{a, a}, False}}
```

```
state = {a, a}; A = DictionaryAutomaton[M]; pos = {};
```

```
s = state
```

```
{a, a}
```

```
Length[s] ≥ Length[A[[5, 1]]]
longitud    longitud
```

```
True
```

```
s

{a, a}

If[s, , False, AppendTo[pos, i - Length[s] + 1]]
si      falso   añade al final    longitud

{0}

s = LongestSuffix[s, A[[5]]]

{a, a}

M

{{a}, {b, b}, {a, a, a}, {a, a, b}, {a, b, b},
 {a, a, a, b}, {a, a, b, a}, {a, a, b, a, b}, {a, a, b, b, b}}

A = DictionaryAutomaton[M]; x = {a, a}; state = A[[4]]

{}

i = 1;

i ≤ Length[x]
    longitud

True

state = Cases[A[[3]], {{a, a, b, a}, a, _}][[1, 3]]
        casos

{a, a}

MemberQ[A[[5]], state]
¿contenido en?

False

state

{{{}, a, {a}}}

state[[1]]

{{}, a, {a}}

! {a, a} ≠ False
          falso

For[i = 1, ! {a} ≠ False && i < 10, i++, Plot[i]]
para cada       falso                    representaci
```