TROTT'S CORNER

Constructing Crossword Arrays Faster

Michael Trott

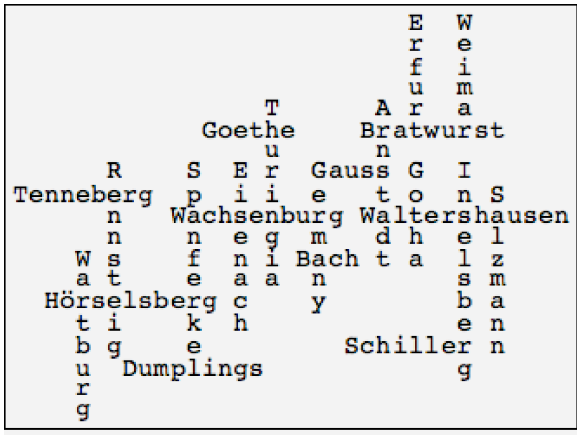
We implement an $O(n^2)$ algorithm to build a crossword array for a set of given words based on hashing techniques.

In Chapter 6 of my *Mathematica GuideBook for Programming*, as an example of list manipulations, I discuss how to build a crossword array. While this serves as a good example for more advanced list manipulations, using lists is not the optimal way to build crossword arrays. The complexity of adding a new word to $n-1$ words already placed using the list-based method is $O(n^2)$, so that the total complexity of constructing a crossword array with n words becomes $O(n^3)$.

Over the last few years, some *Mathematica* users have asked me if it would be possible to build crossword arrays faster. The answer is yes. If we use hashing techniques, we can quickly and in constant time check if a given lattice element is available and can therefore reduce the overall complexity from $O(n^3)$ to $O(n^2)$, which for thousands of words makes a large difference in computation time. Such a method is implemented in this column. When placing the individual letters of a word on a square grid, various cases have to be considered. As a result, this column contains a larger-than-normal amount of procedural code. Some of the function definitions are a bit longer, but with the comments between the computation steps and decision paths, the code should be easily readable.

To be self-contained here, we do not require any material from the *GuideBook* implementation.

Here is the formulation of the problem:
Suppose we are given a list of words as strings. The aim is to insert them in a rectangular grid of squares in such a way that each word is either horizontal (reading from left to right) or vertical (reading from top to bottom), and such that words are connected at squares containing a common letter. Here is an example that uses words of high relevance in the German state of Thuringia, some cultural, and some of culinary value. (In the rest of the column, we use more *Mathematica*-related words as examples.)



For better readability, we require that any two horizontal words and any two vertical words be separated by at least one blank square. In addition, no horizontal word should begin or end in a square that is next to one occupied by a letter in a vertical word, and similarly no vertical word should begin or end in a square that is next to one occupied by a letter in a horizontal word. (Equivalently said, these squares must be blank: those immediately to the left and right of a horizontal word and those immediately above and below a vertical word.) However, we do allow a word to begin or end in a square occupied by another word.

We position all words on a square grid (not limited in size) built from individual squares and keep track of the content of each square.

The function `positionAWord[`*characterList*`, `*k*`, {`*i0*`, `*j0*`}, `*hv*`]` positions the list of characters *characterList* so that the k^{th} element of this list is positioned at the square $\{i0, j0\}$ and the word *characterList* is aligned *hv* (either horizontally or vertically, indicated by `"↔"` or `"↕"`, respectively. We use `C[`*data*`]` to indicate the current status of a square. The following are possible:

- `C[]`—This square must stay empty.
- `C["`*char*`"]`—This square contains character *char* and is part of a horizontal and a vertical word.
- `C["`*char*`", "↔"]`—This square contains character *char* in a vertical word and could still be used in a horizontal word.
- `C["`*char*`", "↕"]`—This square contains character *char* and could still be used in a vertical word.
- `C[_, "↔"]`—This square contains no character yet and could be used in a horizontal word.
- `C[_, "↕"]`—This square contains no character yet and could be used in a vertical word.

All such information is associated with the down values of the function `status`; `status[{`*i*`, `*j*`}]` gives the current state of the square centered at `{`*i*`, `*j*`}`. It either returns `C[...]` or stays unevaluated in case the square has not yet been used in any way and does not neighbor any already positioned word.

We associate a function value for the current content of a square using `status[square]=value`. This allows for a constant-time lookup of the status of each square needed using `status[square]`.

The function `positionAWord` sets the values of the squares where the characters of a word are placed and also adds information about the potential later use of the neighboring squares. For instance, the squares immediately above or below a horizontally placed word can only be used later for vertical words.

The optional last argument of `positionAWord` allows for potential side effects, like monitoring the order of attaching the words, which we use below. The various cases to be considered are easily seen in the comments of the following code.

```
In[12]:= positionAWord[characterList_,
           {k_, {i0_, j0_}, hv : ("↔" | "↕")}, f_ : Null] :=
         Module[{λ = Length[characterList], i, j, cs},
           Which[(* The current word is to be placed horizontally. *)
             hv === "↔",
             (* Nothing can be placed
             immediately before or after the current word. *)
             status[{i0 - k, j0}] = C[];
            status[{i0 - k + λ + 1, j0}] = C[];
             (* Place the characters from the list
             characterList horizontally. *)
             Do[i = i0 - k + i; cs = status[{i, j0}];
               status[{i, j0}] =
               Which[(* Part of a vertical word is already in place. *)
                  MatchQ[cs, C[_String]], cs,
                  (* Both directions are now used. *)
                  MatchQ[cs, C[characterList[[i]], "↔"]],
               C[characterList[[i]]],
                  Head[cs] === C && cs[[2]] === "↔",

               (* Allow the first and last character to be reused. *)
                  If[i === 1 || i === λ, C[characterList[[i]], "↕"],
                C[characterList[[i]]]],
                  (* Nothing is already in place. *)

               Head[cs] === status, C[characterList[[i]], "↕"]];
                (* Here is a potential side effect. *)
                f[{i, j0}, characterList[[i]]],
                {i, λ}];
              (* Set the status of squares in
             the rows below and above the row just set. *)
              Do[i = i0 - k + i; cs = status[{i, j0 + δ}];
               status[{i, j0 + δ}] =
               Which[(* Part of a vertical word is already in place;
                     no further change is possible. *)
                  MatchQ[cs, C[]], C[],
                  MatchQ[cs, C[_String]], cs,
                  MatchQ[cs, C[_String, "↔"]], C[cs[[1]]],
                  MatchQ[cs, Verbatim[C[_, "↕"]]], cs,
                  MatchQ[cs, Verbatim[C[_, "↔"]]], C[],
                  (* Nothing is already in place;
                     only a vertical word could be placed. *)
                  Head[cs] === status, C[_, "↕"]],
               {δ, -1, 1, 2}, {i, λ}];
```

```
   (* Remove "↕" from up-down sandwiched characters. *)
   Do[i = i0 - k + j;

  If[status[{i, j0 - 1}] === status[{i, j0 + 1}] === C[],
      status[{i, j0}] = C[status[{i, j0}][[1]]]],
     {j, λ}],
   (* The current word is to be placed vertically. *)
   hv === "↕",
   (* Nothing can be placed
  immediately under or over the current word. *)
   status[{i0, j0 + k}] = C[];
 status[{i0, j0 + k - λ - 1}] = C[];
   (* Place the characters from the list
  characterList vertically. *)
   Do[j = j0 + k - j; cs = status[{i0, j}];
    status[{i0, j}] =

   Which[(* Part of a horizontal word is already in place. *)
       MatchQ[cs, C[_String]], cs,
       (* Both directions are now used. *)
       MatchQ[cs, C[characterList[[j]], "↕"]],
    C[characterList[[j]]],
       Head[cs] === C && cs[[2]] === "↕",

    (* Allow the first and last characters to be reused. *)
       If[j === 1 || j === λ, C[characterList[[j]], "↔"],
     C[characterList[[j]]]],
       (* Nothing is already in place. *)

    Head[cs] === status, C[characterList[[j]], "↔"]];
    (* This is a potential side effect. *)
    f[{i0, j}, characterList[[j]]],
    {j, λ}];
   (* Set the status of squares in the
  columns left and right of the column just set. *)
   Do[j = j0 + k - j; cs = status[{i0 + δ, j}];
    status[{i0 + δ, j}] =
    Which[(* Part of a horizontal word is already in place;
          no further change is possible. *)
       MatchQ[cs, C[]], C[],
       MatchQ[cs, C[_String]], cs,
       MatchQ[cs, C[_String, "↕"]], C[cs[[1]]],
       MatchQ[cs, Verbatim[C[_, "↔"]]], cs,
       MatchQ[cs, Verbatim[C[_, "↕"]]], C[],
       (* Nothing is already in place;
          only a vertical word could be placed. *)
       Head[cs] === status, C[_, "↔"]],
    {δ, -1, 1, 2}, {j, λ}];

(* Remove "↔" from left-right sandwiched characters.nb. *)
   Do[j = j0 + k - j;

  If[status[{i0 - 1, j}] === status[{i0 + 1, j}] === C[],
      status[{i0, j}] = C[status[{i0, j}][[1]]]],
     {j, λ}]
   ]]
```

As an example, we position the word *Mathematica* at the square {1, 1} horizontally.

```
In[13]:= positionAWord[Characters["Mathematica"], {1, {1, 1}, "↔"}]
```

And these are the current squares that are marked.

```
In[14]:= DownValues[status] // Column
```

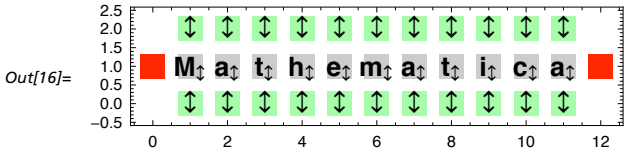
```
Out[14]= HoldPattern[status[{0, 1}]] :> C[]
         HoldPattern[status[{1, 0}]] :> C[_, ↕]
         HoldPattern[status[{1, 1}]] :> C[M, ↕]
         HoldPattern[status[{1, 2}]] :> C[_, ↕]
         HoldPattern[status[{2, 0}]] :> C[_, ↕]
         HoldPattern[status[{2, 1}]] :> C[a, ↕]
         HoldPattern[status[{2, 2}]] :> C[_, ↕]
         HoldPattern[status[{3, 0}]] :> C[_, ↕]
         HoldPattern[status[{3, 1}]] :> C[t, ↕]
         HoldPattern[status[{3, 2}]] :> C[_, ↕]
         HoldPattern[status[{4, 0}]] :> C[_, ↕]
         HoldPattern[status[{4, 1}]] :> C[h, ↕]
         HoldPattern[status[{4, 2}]] :> C[_, ↕]
         HoldPattern[status[{5, 0}]] :> C[_, ↕]
         HoldPattern[status[{5, 1}]] :> C[e, ↕]
         HoldPattern[status[{5, 2}]] :> C[_, ↕]
         HoldPattern[status[{6, 0}]] :> C[_, ↕]
         HoldPattern[status[{6, 1}]] :> C[m, ↕]
         HoldPattern[status[{6, 2}]] :> C[_, ↕]
         HoldPattern[status[{7, 0}]] :> C[_, ↕]
         HoldPattern[status[{7, 1}]] :> C[a, ↕]
         HoldPattern[status[{7, 2}]] :> C[_, ↕]
         HoldPattern[status[{8, 0}]] :> C[_, ↕]
         HoldPattern[status[{8, 1}]] :> C[t, ↕]
         HoldPattern[status[{8, 2}]] :> C[_, ↕]
         HoldPattern[status[{9, 0}]] :> C[_, ↕]
         HoldPattern[status[{9, 1}]] :> C[i, ↕]
         HoldPattern[status[{9, 2}]] :> C[_, ↕]
         HoldPattern[status[{10, 0}]] :> C[_, ↕]
         HoldPattern[status[{10, 1}]] :> C[c, ↕]
         HoldPattern[status[{10, 2}]] :> C[_, ↕]
         HoldPattern[status[{11, 0}]] :> C[_, ↕]
         HoldPattern[status[{11, 1}]] :> C[a, ↕]
         HoldPattern[status[{11, 2}]] :> C[_, ↕]
         HoldPattern[status[{12, 1}]] :> C[]
```

To see the progress of the construction visually, we define a function `makeCWPGridGraphics` that colors the squares according to their current status.

```
In[15]:= makeCWPGridGraphics[status_, lastAdd_: {}, fs_: 14, opts___] :=
         Module[{dv = DownValues[status, Sort → False],
             currentlyMarkedSquares, coloredSquares},
             (* These are the positions of the characters. *)
             currentlyMarkedSquares = {#[[1, 1]], #[[2]]} & /@ dv;
             (* Color squares according to status. *)
             coloredSquares =
              Switch[#2, C[], {Red, Rectangle[#1 - 1/3, #1 + 1/3]},
                  C[_String],
                {GrayLevel[0.9], Rectangle[#1 - 1/3, #1 + 1/3],
                        {Black, Text[Style[#2[[1]],
                   Bold, FontSize → fs], #1]}},
                C[_String, "↔" | "↕"],

                {GrayLevel[0.8], Rectangle[#1 - 1/3, #1 + 1/3],
                        {Black,

                  Text[Style[Subscript[Style[#2[[1]], Bold], #2[[2]]],
                             FontSize → fs], #1]}},
                Verbatim[C[_, "↔"]] | Verbatim[C[_, "↕"]],

                {RGBColor[0.6, 1, 0.6], Rectangle[#1 - 1/3, #1 + 1/3],
                        {Black, Text[Style[#2[[2]],
                    FontSize → fs], #1]}}] & @@@
                        currentlyMarkedSquares;
             Graphics[{coloredSquares,
                  If[lastAdd =!= {}, {Blue, Thickness[0.006],
                      Line[(lastAdd[[2]] + 5/12 #) & /@
                         {{-1, -1}, {1, -1}, {1, 1},
                          {-1, 1}, {-1, -1}}]}, {}]},
                   opts, Frame → True,
              PlotRange → All, PlotRangeClipping -> True]]
```

The following graphic visualizes the current status of the squares near the placed characters.

```
In[16]:= makeCWPGridGraphics[status]
```



The function `positioningAWordIsPossibleQ[`*characterList*`, `*k*`, {`*i0*`, `*j0*`}, `*hv*`]` checks if it is possible to position the list of characters *characterList* so that the k^{th} element of this list is positioned at $\{i0, j0\}$ and the word is aligned *hv* either horizontally or vertically, indicated by `"↔"` or `"↕"`, respectively. To do this we must check that each of the needed squares is either available or already has the needed character in place. And the neighboring squares must be empty or have letters from an already orthogonally positioned word. The auxiliary function `whileTrueLoops` is an iterating function that stops iterating when the first

incompatible square is found by using the `Throw-Catch` mechanism. In the following, we always assume that a new word is only placed using the function `positionAWord` in case `positioningAWordIsPossibleQ` gives True.

```
In[17]:= SetAttributes[whileTrueLoops, HoldAll]

         whileTrueLoops[body_, iters__] :=
           Catch[Table[If[Not[body], Throw[False]], iters]] =!= False;

In[19]:= positioningAWordIsPossibleQ[characterList_,
           {k_, {i0_, j0_}, hv : ("↔" | "↕")}] :=
         Module[{λ = Length[characterList], i, j, cs},
           Which[(* The current word is to be placed horizontally. *)
             hv === "↔",
             (* A blank square is
             immediately before and after the current word;
               it is either unused or a reserved space. *)
             MatchQ[status[{i0 - k, j0}],
              _status | C[] | Verbatim[C[_, "↔"]]] &&
             MatchQ[status[{i0 - k + λ + 1, j0}],
              _status | C[] | Verbatim[C[_, "↔"]]] &&
             (* Any character must find an empty space
              or must match an existing one. *)
                   (whileTrueLoops[i = i0 - k + i; cs = status[{i, j0}];
                cs =!= C[] && Not[MatchQ[cs, C[_, "↕"]]] &&
                ((Head[cs] === status) ||
                 (MatchQ[cs, C[_String]] &&
                    cs[[1]] === characterList[[i]]) ||
                 (Head[cs] === C && Length[cs] === 2 &&
                    cs[[2]] === "↔" &&
                  (cs[[1]] === characterList[[i]] || cs[[1]] === _))),
                {i, λ}]) &&
             (* Spaces above and below must
             be free or must be in use for vertical words. *)
             (whileTrueLoops[i = i0 - k + i; cs = status[{i, j0 + δ}];
                (* Part of a vertical
                word is already in place or still usable *)
                cs === C[] || MatchQ[cs, C[_String]] ||
                (Head[cs] === status) ||
                 (Head[cs] === C && cs[[1]] === _) ||
                 MatchQ[cs, C[_String, "↔"]] ,
               {δ, -1, 1, 2}, {i, λ}]),
             (* The current word is to be placed vertically. *)
             hv === "↕",
             (* A space is needed
             immediately before and after the current word;
               it is either unused or a reserved space. *)
             MatchQ[status[{i0, j0 + k}],
              _status | C[] | Verbatim[C[_, "↕"]]] &&
             MatchQ[status[{i0, j0 + k - λ - 1}],
              _status | C[] | Verbatim[C[_, "↕"]]] &&
             (* Any character must find an empty space
              or must match an existing one. *)
                   (whileTrueLoops[j = j0 + k - j; cs = status[{i0, j}];
                cs =!= C[] && Not[MatchQ[cs, C[_, "↔"]]] &&
                ((Head[cs] === status) ||
                 (MatchQ[cs, C[_String]] &&
```

```
        cs[[1]] === characterList[[j]]) ||
     (Head[cs] === C && Length[cs] === 2 &&
        cs[[2]] === "↕" &&
      (cs[[1]] === characterList[[j]] || cs[[1]] === _))),
    {j, λ}]) &&
(* Spaces left and right must
be free or must be in use for horizontal words. *)
(whileTrueLoops[j = j0 + k - j; cs = status[{i0 + δ, j}];
    (* Part of a horizontal word is already in place;
       no further change is possible. *)
    cs === C[] ||
   MatchQ[cs, C[_String]] || (Head[cs] === status) ||
    (Head[cs] === C && cs[[1]] === _) ||
    MatchQ[cs, C[_String, "↕"]],
  {δ, -1, 1, 2}, {j, λ}])
 ]]
```

With the already placed horizontal word *Mathematica*, we can position another copy of *Mathematica* vertically.

```
In[20]:= positioningAWordIsPossibleQ[
          Characters["Mathematica"], {1, {1, 1}, "↕"}]
Out[20]= True
```

But, we cannot place another horizontal *Mathematica* at the same position.

```
In[21]:= positioningAWordIsPossibleQ[
          Characters["Mathematica"], {1, {1, 1}, "↔"}]
Out[21]= False
```

Nor can we place another copy along the already placed word starting in the middle.

```
In[22]:= positioningAWordIsPossibleQ[
          Characters["Mathematica"], {3, {3, 3}, "↔"}]
Out[22]= True
```

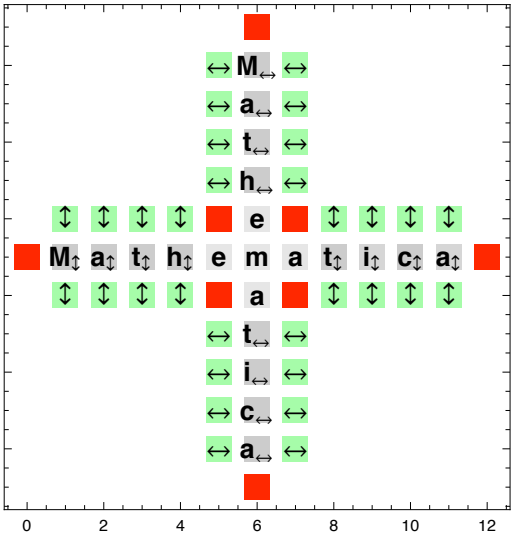
Of course, we could also position another copy of *Mathematica* in the middle, say at the first "m" of the horizontal *Mathematica*.

```
In[23]:= positioningAWordIsPossibleQ[
          Characters["Mathematica"], {1, {6, 6}, "↕"}]
Out[23]= True

In[24]:= positionAWord[Characters["Mathematica"], {1, {6, 6}, "↕"}]
```

This gives the following state of our grid.

```
In[25]:= makeCWPGridGraphics[status]
```

Out[25]=



The function `findAPossiblePositionAndPositionAWord[`*characterList*`]` finds a possible placement for the characters of the word *characterList* and places those characters. The function either returns the coordinates of the placement of the attachment character or returns `$Failed` in case no possible attachment is found. We also give this function a few self-explanatory options that, among others, allow the placement of the new words to be as central as possible or as far apart as possible. The option setting `Inner` tries to attach the next word inside the given words, the setting `Outer` tries to attach the next word at the outside of the given words, and the option setting `RandomSample` chooses a random position for the next attachment.

In[26]:=
```
(* This is an auxiliary function for
 sorting the possible attachment squares. *)
sortCData[l_, {io_, dr_}] :=
With[{f = If[io === Inner, Less, Greater],
   g = If[dr === Deterministic, Sort, RandomSample],
   h = If[io === Random, RandomSample, Identity]},
  h[Flatten[
   g /@ (Map[Last, Split[Sort[{Norm[#1[[2]]], #1} & /@ l,
                    #1[[1]] ~ f ~ #2[[1]] &],
                  #1[[1]] === #2[[1]] &], {2}]), 1]]]
```

In[27]:=
```
Options[findAPossiblePositionAndPositionAWord] =
   {AttachmentDirection → Automatic,
    AttachmentPosition → Inner,
    AttachmentReproducibility → Deterministic};
```

```
In[28]:= findAPossiblePositionAndPositionAWord[
           characterList_, f_: Null, opts : OptionsPattern[]] :=
         Module[{availableAttachmentPoints,
            availableAttachmentPointsGrouped,
             possibleCharacterMatches, rules, pts,
            data, aDir, dataD, aPos, aRep,
             dataP, possiblePositionFound, μ, counter},

           (* These are the squares available for attaching the new word. *)
            availableAttachmentPoints = Reverse /@
                Cases[
              {#[[1, 1]], #2} & @@@ DownValues[status, Sort → False],
                  {_, C[_String, "↔" | "↕"]}];
            (* Group by letter. *)

           availableAttachmentPointsGrouped = {#[[1, 1]], Last /@ #} & /@
                Split[{#[[1, 1]], #} & /@
               Sort[availableAttachmentPoints],
                   #1[[1]] === #2[[1]] &];
            (* Find matching squares. *)
            possibleCharacterMatches =
            Cases[availableAttachmentPointsGrouped,
                          {Alternatives @@ characterList, _}];
            rules = (#[[1, 1]] → Last /@ #) & /@

             Split[Sort[MapIndexed[(#1 → #2[[1]]) &, characterList]],
                   #1[[1]] === #2[[1]] &];
            pts = {#1 /. rules, #2} & @@@ possibleCharacterMatches;
            data = {#1, #2[[2]], #2[[1, 2]]} & @@@
                      Flatten[Outer[List, #1, #2, 1] & @@@ pts, 2];
            (* Find current option settings. *)
            aDir = OptionValue[AttachmentDirection];
            aPos = OptionValue[AttachmentPosition];
            aRep = OptionValue[AttachmentReproducibility];
            dataD = Which[aDir === "↔", Cases[data, {_, _, "↔"}],
                   aDir === "↕", Cases[data, {_, _, "↕"}],
                   aDir === Automatic, data];
            dataP = sortCData[dataD, {aPos, aRep}];

           (* Try to fit the word in the order of attachment letters found .*)
            possiblePositionFound = False;
            (* Loop over all possible attachment squares. *)
            μ = Length[dataP]; counter = 1;
            While[counter <= μ &&
               Not[(possiblePositionFound =
                  positioningAWordIsPossibleQ[
                  characterList, dataP[[counter]]])],
                counter++];
            (* Position at the square found. *)
            If[possiblePositionFound,
             positionAWord[characterList, dataP[[counter]], f];
             dataP[[counter]], $Failed]]
```

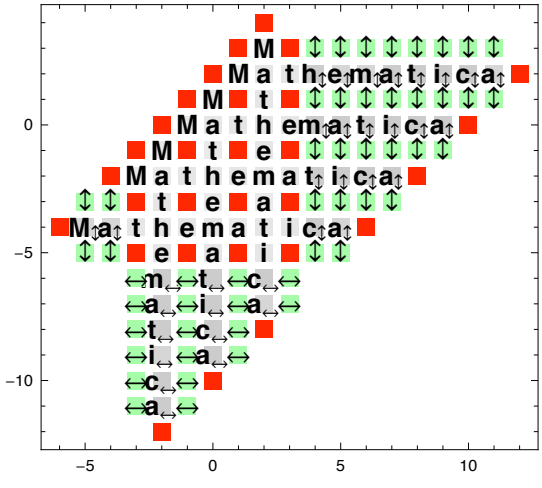
Finally, we define the function `makeCWPGrid[`*status*`]` which constructs the actual grid of letters from the down values of `status`.

```
In[29]:= makeCWPGrid[status_] :=
         Module[
           {dv = DownValues[status, Sort → False], placedCharacters,
            xMin, xMax, yMin, yMax, placedCharactersShifted, T},
            (* The characters are positioned. *)
            placedCharacters = {#2[[1]], #1} & @@@
               Cases[{#[[1, 1, 1]], #[[2]]} & /@ dv,
                  {_, C[_String] | C[_String, _]}];
            (* Here are the shifts needed. *)
            {xMin, yMin} = Min /@ Transpose[Last /@ placedCharacters];
            (* Here are the shifted characters. *)
            placedCharactersShifted = {#1, #2 - {xMin, yMin} + 1} & @@@
                                      placedCharacters;
            (* Here are the extensions. *)
            {{xMin, xMax}, {yMin, yMax}} =
           {Min[#], Max[#]} & /@ Transpose[Last /@
                                    placedCharactersShifted];
            (* Fill out the array. *)
            T = Table["", {y, yMin, yMax}, {x, xMin, xMax}];
            (T[[#2[[2]], #2[[1]]]] = #1) & @@@ placedCharactersShifted;
            (* Return the array as a grid. *)
            Grid[Reverse @ T, Spacings → (* tight *) {0, -0.1}]]
```

The following graphic shows the state of all already touched squares after attaching the word *Mathematica* six times to the initial word *Mathematica*. The gray squares contain already placed words, the red squares must stay empty, and the green squares allow further words to be attached. The arrows indicate in which direction potential words can be attached.

```
In[30]:= Clear[status];
         positionAWord[Characters["Mathematica"], {2, {0, 0}, "↔"}]
         Do[findAPossiblePositionAndPositionAWord[
            Characters["Mathematica"]], {6}];
         makeCWPGridGraphics[status]
```

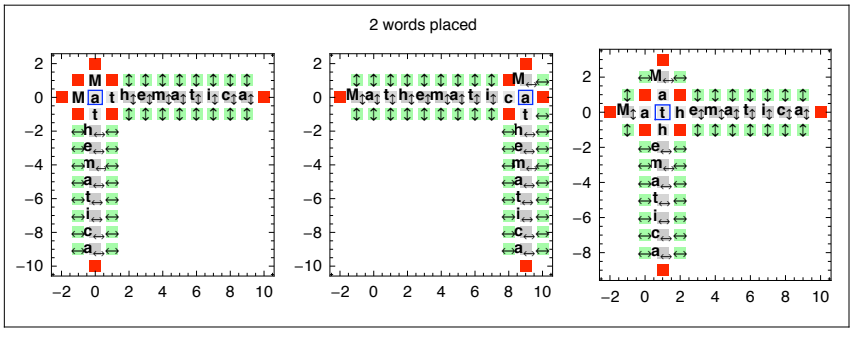
Out[33]=

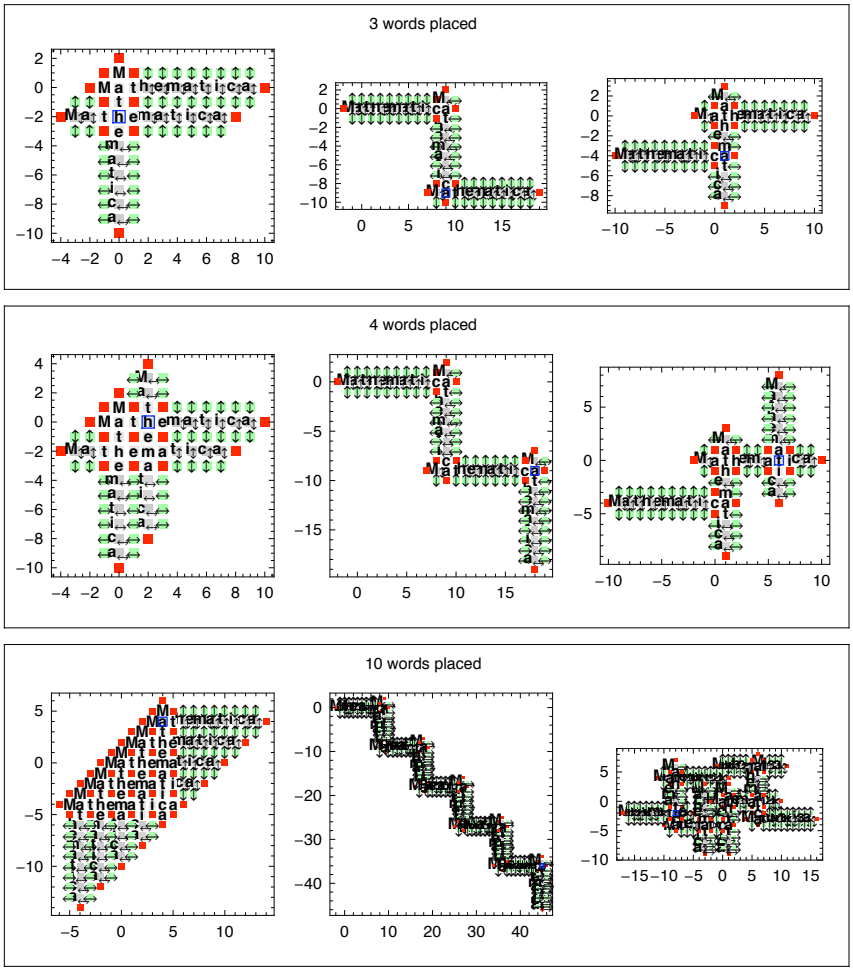


```
In[34]:= SeedRandom[123];
(progressGraphics[#] =
Block[{status, pr, lA, stateGraphicsList},
stateGraphicsList = Flatten[
     {(* Place the first word. *)
      positionAWord[
           Characters["Mathematica"], {2, {0, 0}, "↔"}];
      (* Make the graphics. *)
      makeCWPGridGraphics[status],
      (* Place the words and make the graphics. *)
      Table[lA = findAPossiblePositionAndPositionAWord[
             Characters["Mathematica"],
             Null, AttachmentPosition → #];
         makeCWPGridGraphics[status, lA], {12}]}];
(* Extend the last graphic—make it all fit. *)
pr = FullOptions[Last[stateGraphicsList], PlotRange] +
       {{-1, 1}, {-1, 1}};
(* Use the size of the last graphic in all graphics. *)
Show[#, PlotRange → pr, ImageSize → 400] & /@
      stateGraphicsList]) & /@
                    {Inner, Outer, Random};
```

The following graphic shows some steps in the process of building the crossword arrays. Looking in detail at the values of the squares makes it easy to understand the details of the above-implemented algorithm.

```
In[36]:= Do[Print @
     GraphicsRow[Show[progressGraphics[#][[k]] /.
        (FontSize → _) :> (FontSize → Inherited),
       PlotRange → All] & /@ {Inner, Outer, Random},
                    PlotLabel → Row[{k, " words placed"}],
   ImageSize → {600, Automatic}] ,
  {k, {2, 3, 4, 10}}]
```

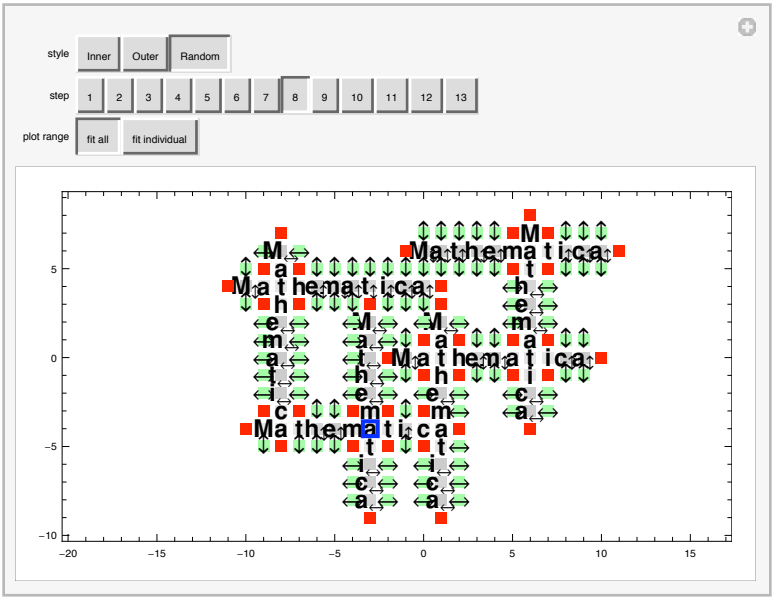




Using the function `Manipulate`, we can easily construct an interactive version that lets us see all the steps of the growth process. The following interactive version lets us monitor the growth for the first twelve steps.

In[37]:=
```
Manipulate[Show[progressGraphics[ior][[k]],
           If[pr === "fit", PlotRange → All, Sequence @@ {}]],
        {{ior, Inner, "style"}, {Inner, Outer, Random}},
        {{k, 8, "step"},
  Range[1, Length[progressGraphics[ior]], 1], SetterBar},
        {{pr, Automatic, "plot range"},
         {Automatic → "fit all", "fit" → "fit individual"}},
  SaveDefinitions → True]
```

Out[37]=



Now let us analyze the complexity of the approach implemented. Because the lookup of the current state of a square is approximately constant (independent of the number of already set squares), we expect a linear complexity $O(n)$ in the number of words. As a test, we place the word *Mathematica* 500 times. To visually follow the progress, we use `Monitor`.

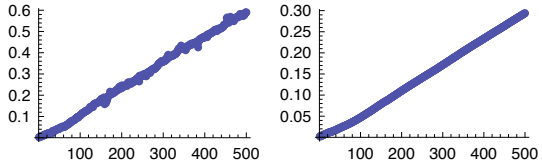
In[38]:=

```
Clear[status];
SeedRandom[1];
With[{μL = 500, wordCharacters = Characters["Mathematica"]},
positionAWord[Characters["Mathematica"], {2, {0, 0}, "↔"}];
Monitor[
timingDataListCache =
     Table[Δt = Timing[findAPossiblePositionAndPositionAWord[
                    wordCharacters, Null,
                    AttachmentPosition → Inner,
                    AttachmentReproducibility → Random]][[1]],
               {z, μL}];,
    Row[{Text["◇ Attaching word number "],
     Text[ToString[z]], Text[" out of "],
       Text[ToString[μL]], ": ", NumberForm[Δt, 3], " seconds"}]]]
```

Next, we display the measured timings. The right graphic shows the cumulative average time needed to attach a word. It increases linearly with the number of the word to be attached.

```
In[41]:= GraphicsRow[{ListPlot[timingDataListCache],
           ListPlot[averagedTimingDataCache =
             MapIndexed[{#2[[1]], #1/#2[[1]]}&,
               Accumulate[timingDataListCache]]]}]
```

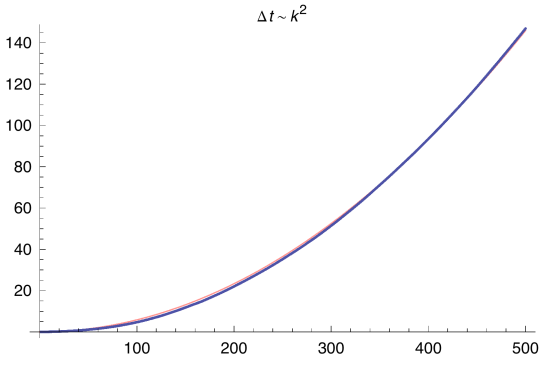
Out[41]=



And the cumulative time to form a crossword array of n words has complexity $O(n^2)$. The following graphic together with a fit shows this clearly.

```
In[42]:= (* Here is the timing data for a crossword array. *)
         cumulativeTimingDataListCache = Accumulate[timingDataListCache];

         (* Show the timing data and fitted
          curve for building a crossword array. *)
         Show[{Plot[Evaluate[Fit[Drop[MapIndexed[{#2[[1]], #1}&,
                 cumulativeTimingDataListCache], 200],
                   {k^2}, k]], {k, 0, Length[
               cumulativeTimingDataListCache]},
                PlotStyle → Directive[Opacity[0.5], Red]],
                ListPlot[cumulativeTimingDataListCache,
            PlotStyle → PointSize[0.004]]},
              PlotLabel → HoldForm[Δt ~ k^2]]
```

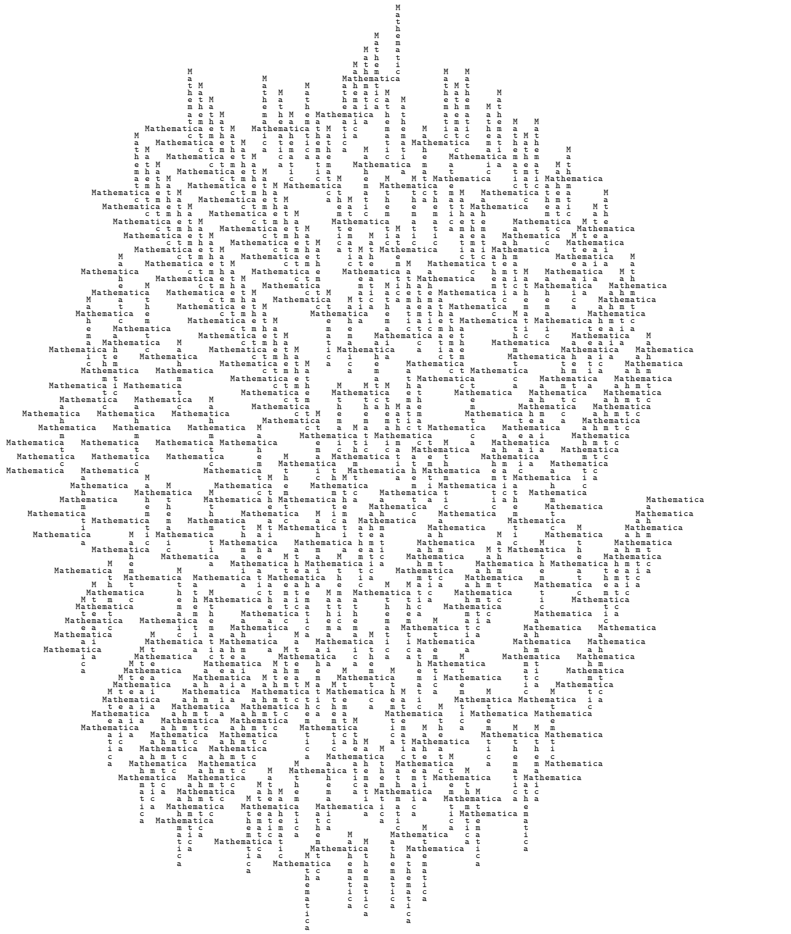
Out[43]=



This is the resulting crossword array. We use a small font to display it in its entirety.

```
In[44]:= Style[makeCWPGrid[status], 5]
```

Out[44]=



Because of the much better runtime of this caching-based approach for a larger numbers of words (say thousands), we can now quickly position the first 250 words by using a built-in function in a crossword array.

In[45]:=

```
Clear[status];
SeedRandom[123];

allBuiltInNames = Select[ Names["System`*"],
    Not[StringMatchQ[ToString[FullForm[#]], "*Formal*"]] &];

positionAWord[
  Characters[allBuiltInNames[[1]]], {1, {0, 0}, "↔"}];

(Monitor[timeSpaceData =
 Table[currentWord = allBuiltInNames[[z]];
    Timing[lA = findAPossiblePositionAndPositionAWord[
          Characters[currentWord]];
    If[lA === $Failed, Abort[], lA]], {z, 2, 250}],
   Row[{Text["◇ Positioning word number " <>
       ToString[z] <> ": "], ToString[currentWord]}]]); //
 Timing
```

Out[49]= {17.8735, Null}

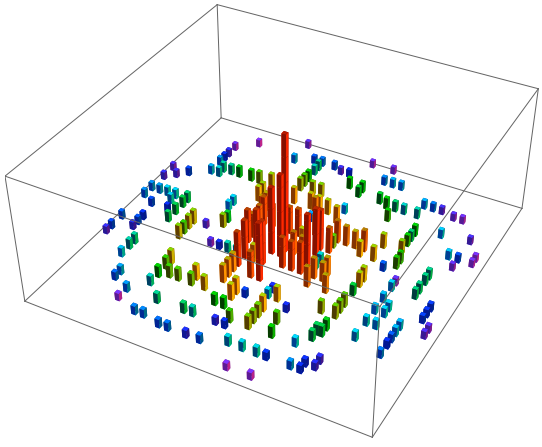
In[50]:= **Style[makeCWPGrid[status], 6]**

Out[50]=

The following graphic shows the order in which the words were placed in the plane. Red, tall towers indicate words that were positioned early, and blue, low towers indicate later ones.

```
In[51]:= Graphics3D[{EdgeForm[],
           N[MapIndexed[{Hue[0.78 (#2[[1]]/Length[timeSpaceData])],
                       Cuboid[Append[#1[[2, 2]] - 1/2, 0],
                          Append[#1[[2, 2]] + 1/2, #2[[1]]^(-1/2)]]} &,
                       timeSpaceData]]},
             PlotRange → All, BoxRatios → {1, 1, 0.4}]
```

Out[51]=



And here is a crossword array of all the visualization functions and their options.

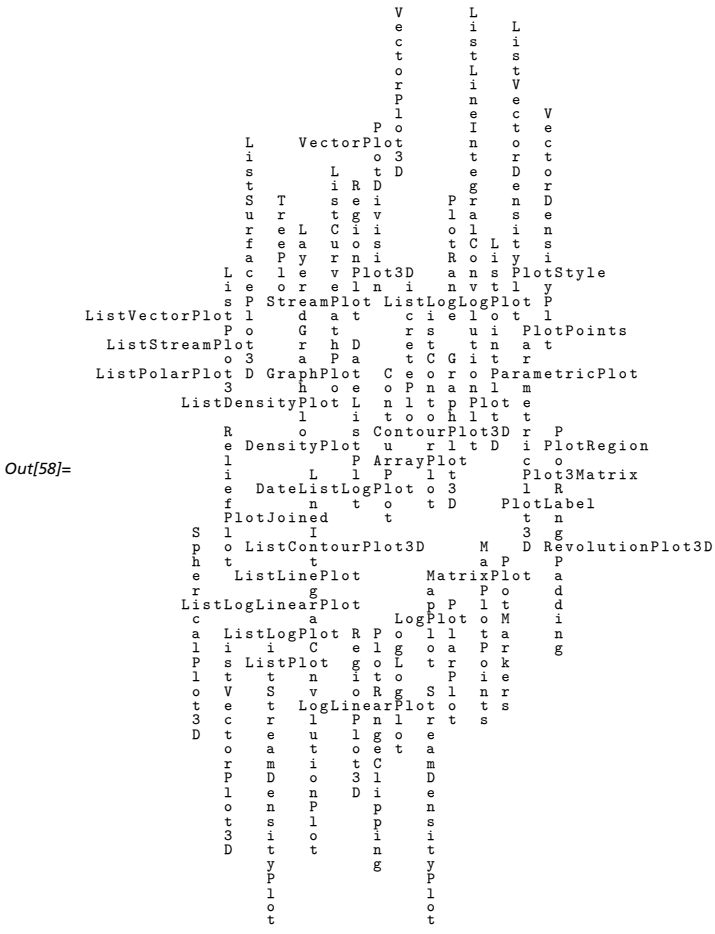
```
In[52]:= visualizationRelatedMathematicaSymbols =
           Union[Flatten[{#, First /@ Options[ToExpression[#]]} & /@
              Names["*Plot*"]]];
         Length[visualizationRelatedMathematicaSymbols]
```

Out[53]= 186

```
In[54]:= Clear[status];
         SeedRandom[222];

         positionAWord[
           Characters[visualizationRelatedMathematicaSymbols[[1]]],
           {1, {0, 0}, "↔"}];

         Do[currentWord = visualizationRelatedMathematicaSymbols[[z]];

          findAPossiblePositionAndPositionAWord[Characters[currentWord]],
             {z, 2, Length[visualizationRelatedMathematicaSymbols]}]
```

In[58]:=
```
makeCWPGrid[status]
```

Out[58]=



We continue by using the potential function `f` in `positionAWord`. We use the function to record to which positioned word a character belongs.

In[59]:=
```
TMJPhraseWords =
  ToUpperCase /@ {"The", "Mathematica", "Journal",
                  "Programming", "Graphics",
    "Numerics", "Symbolics", "Interfaces"};

Clear[status, f];

positionAWord[Characters[TMJPhraseWords[[1]]],
  {1, {0, 0}, "↔"}, (f[1][#1] = #2) &];

(Monitor[Table[lA = findAPossiblePositionAndPositionAWord[
                Characters[RandomChoice[TMJPhraseWords]],
                (f[z][#1] = #2) &];
        If[lA === $Failed, Abort[], lA], {z, 2, 50}],
   "◇ " <> ToString[z]]);
```

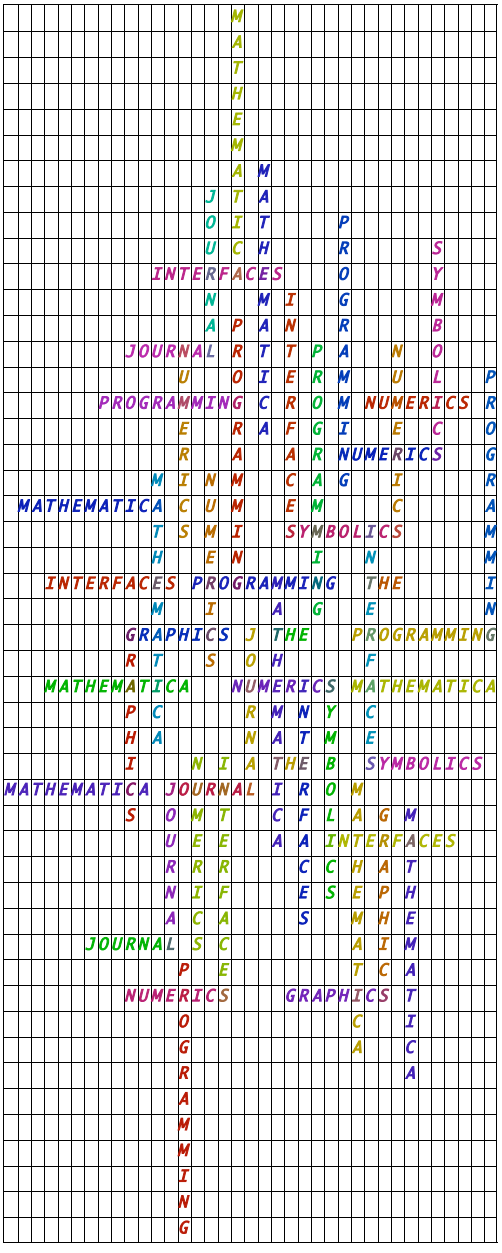
By implementing a small variation of the function `makeCWPGrid`, we can now use the information associated with `f` to color the positioned words.

```
In[63]:= makeColoredCWPGrid[f_] :=
         Module[
           {placedCharacters, xMin, xMax, yMin, yMax, T, λ, color,
            placedCharactersGrouped, addColor, placedCharactersColored},
           placedCharacters =
           {#[[1, 1, 1]], #[[1, 1, 0, 1]], #[[2]]} & /@ SubValues[f];
           {xMin, yMin} = Min /@ Transpose[
             First /@ placedCharacters];
           placedCharactersShifted =
           {#1 - {xMin, yMin} + 1, #2, #3} & @@@
                                  placedCharacters;
           {{xMin, xMax}, {yMin, yMax}} =
                  {Min[#], Max[#]} & /@
            Transpose[First /@ placedCharactersShifted];
           T = Table["", {y, yMin, yMax}, {x, xMin, xMax}];
           (* Color each word randomly;
         blend color at intersecting words. *)
           λ = Max[#[[2]] & /@ placedCharactersShifted];
           Do[color[k] = Darker[Hue[RandomReal[]]], {k, λ}];

           placedCharactersGrouped = Split[Sort[placedCharactersShifted],
                            #1[[1]] === #2[[1]] &];
           addColor[l_] := {l[[1, 1]], Style[l[[1, 3]],
                    If[Length[l] > 1,
              Blend[color[#[[2]]] & /@ l], color[l[[1, 2]]]]]};
           placedCharactersColored = addColor /@
            placedCharactersGrouped;
           (T[[#1[[2]], #1[[1]]]] = #2) & @@@ placedCharactersColored;
           Grid[Reverse @ T, Spacings → 0]]
```

This results in the following colored grid elements. At the crossings, we blend the colors.

```
In[64]:= Append[makeColoredCWPGrid[f] /.

          s_String :> Style[s, Bold, Italic, FontFamily → "Monaco"],
         Dividers → All]
```

Out[64]=



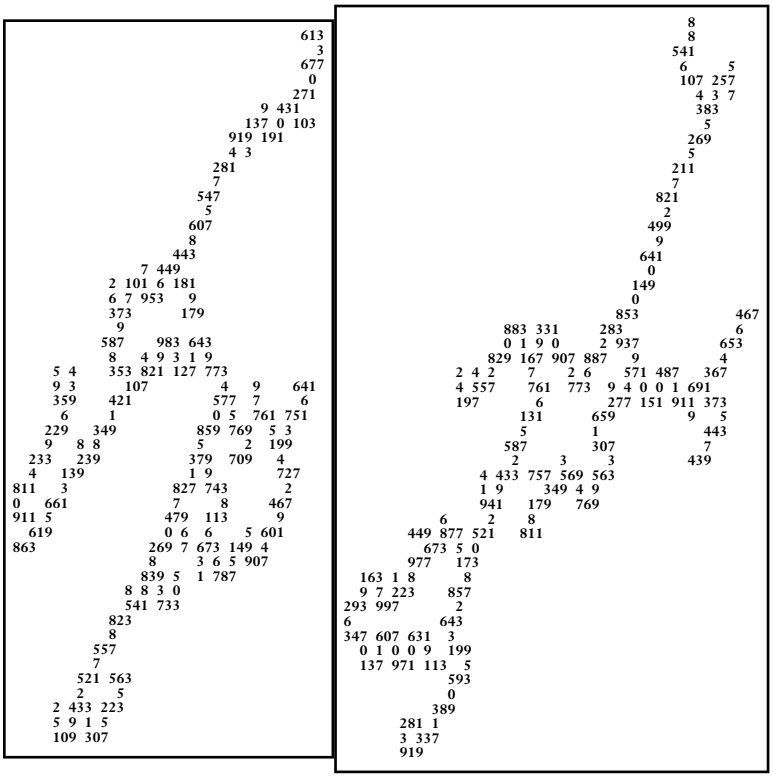
We end with a function `CrossWordConstructionCached` that tries to position a list of words into one crossword array.

```
In[65]:= CrossWordConstructionCached[{startString_String, "↔" | "↕"},
                  workStrings : {_String ..}, opts___] :=
         Module[{words, lA, rotateCounter = 0},
          Clear[status];
          (* Prepare workstring as single characters. *)
          words = Characters /@ workStrings;
          (* Position first word. *)
          positionAWord[Characters[startString], {1, {0, 0}, "↔"}];
          While[words =!= {} && rotateCounter < Length[words],
             lA = findAPossiblePositionAndPositionAWord[
               First[words], Null, opts];
             If[lA === $Failed, rotateCounter++;
              words = RotateRight[words],
               rotateCounter = 0; words = Rest[words]]];
          {makeCWPGrid[status], words}]
```

Here are two ways to position all the three-digit primes in such an array.

```
In[66]:= Row[Framed[Style[#, 8]] & /@ ((SeedRandom[#];
          threeDigitPrimes = RandomSample[ToString /@
                Prime[Range[PrimePi[99] + 1, PrimePi[999]]]];
          CrossWordConstructionCached[{First[threeDigitPrimes], "↔"},
                Rest[threeDigitPrimes]][[1]]) & /@ {7, 37}]
```

Out[66]=



References

M. Trott, "Constructing Crossword Arrays Faster," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.11.2–1

About the Author

Michael Trott
Senior Member, Technical Staff
Wolfram Research, Inc.
mtrott@wolfram.com