```
words = open("./names.txt" , "r") |>
    io -> read(io, String) |>
    s -> split(s, "\n");
```

```
["emma", "olivia", "ava", "isabella", "sophia", "charlotte", "mia", "amelia", "harper", "evelyn"]
```

```
words[1:10]
```

```
# all words are lowercase
```

```
32033
```

```
words |> length
```

```
(2, 15)
```

```
extrema(s -> length(s), words)  # min-len word, max-len word
```

```
[('.', 'o'), ('o', 'l'), ('l', 'i'), ('i', 'v'), ('v', 'i'), ('i', 'a'), ('a', '.')]
```

```
begin
    # computes all the bigrams of a given words
    const BOS = '.'
    const EOS = '.'
    const W  = string(BOS, words[2], EOS)

    zip(W[1:end], W[2:end]) |> collect
end
```

```
Dict(('x', 'z') ⇒ 19, ('a', 'f') ⇒ 134, ('k', 'a') ⇒ 1731, ('p', 'c') ⇒ 1, ('s', 'k') ⇒ 82, ('s', 'u') ⇒ 185, ('a', 'm') ⇒
```

```
begin
    # build all the bigrams
    bigrams = Dict{Tuple, Integer}()

    for w ∈ words
        w = string(BOS, w, EOS)
        for bigram ∈ zip(w[1:end], w[2:end]) |> collect
            bigrams[bigram] = get(bigrams, bigram, 0) + 1
        end
    end

    bigrams
end
```

```
top10 =
 [('n', '.') ⇒ 6763, ('a', '.') ⇒ 6640, ('a', 'n') ⇒ 5438, ('.', 'a') ⇒ 4410, ('e', '.') ⇒ 3983, ('a', 'r') ⇒ 3264, ('e', 'l'
```

```
# top 10 most frequent for example
top10 = sort(bigrams |> collect; by = t -> t[2], rev = true)[1:10]
```

Instead of a dictionary we want a 2D-array, where the rows will be the first character and the columns will be the second character and the cell will store how often the second character follows the first one.

We have 26 characters (a..z) and two special characters '<', '>', thus we need a (28, 28) 2D-array

```
all_chars =
 ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

```
# let join all the words together to get all the characters from 'a'..'z'
all_chars = words |> join |> s -> split(s, "") |> s -> first.(s) |> Set |> collect |> sort
```

```
27
```

```
begin
    # char to int - generator/ no array
    ctoi = Dict(
        s => ix for (ix, s) ∈ enumerate(all_chars)
    )
    ctoi[BOS] = 27 # = EOS
end
```

```
Dict{Char, Int64}(
    'n' ⇒ 14
    'f' ⇒ 6
    'w' ⇒ 23
    'd' ⇒ 4
    'e' ⇒ 5
    'o' ⇒ 15
    'h' ⇒ 8
    'j' ⇒ 10
    'i' ⇒ 9
    'k' ⇒ 11
    'r' ⇒ 18
    's' ⇒ 19
    't' ⇒ 20
    'q' ⇒ 17
    'y' ⇒ 25
    'a' ⇒ 1
    'c' ⇒ 3
    'p' ⇒ 16
    'm' ⇒ 13
    'z' ⇒ 26
    '.' ⇒ 27
    'g' ⇒ 7
    'v' ⇒ 22
    'l' ⇒ 12
    'u' ⇒ 21
    'x' ⇒ 24
    'b' ⇒ 2
)
```

```
ctoi
```

```
27×27 Matrix{Int32}:
  556   541   470  1042   692  134  168  2332  …  381  834  161  182  2050  435  6640
  321    38     1    65   655    0    0    41      45    0    0    0    83    0   114
  815     0    42     1   551    0    2   664      35    0    0    3   104    4    97
 1303     1     3   149  1283    5   25   118      92   17   23    0   317    1   516
  679   121   153   384  1271   82  125   152      69  463   50  132  1070  181  3983
  242     0     0     0   123   44    1     1  …   10    0    4    0    14    2    80
  330     3     0    19   334    1   25   360      85    1   26    0    31    1   108
    ⋮                         ⋮                ⋱    ⋮                         ⋮
  642     1     0     1   568    0    0     1       7    7    0    0   121    0    88
  280     1     0     8   149    2    1    23      25    0    2    0    73    1    51
  103     1     4     5    36    3    0     1       5    0    3   38    30   19   164
 2143    27   115   272   301   12   30    22     141  106    4   28    23   78  2007
  860     4     2     2   373    0    1    43  …   73    2    3    1   147   45   160
 4410  1306  1542  1690  1531  417  669   874      78  376  307  134   535  929     0
```

```
begin
    # let's build our 2D-array
    const N = 27

    # NOTE: alternatively do not use the dictionary at all...
    vbigrams = fill(Int32(0), (N, N))
    for bigram ∈ keys(bigrams)
        ix, jx = ctoi[bigram[1]], ctoi[bigram[2]]
        vbigrams[ix, jx] = bigrams[bigram]
    end
    vbigrams
end
```

```
itoc =
  Dict(5 ⇒ 'e', 16 ⇒ 'p', 20 ⇒ 't', 12 ⇒ 'l', 24 ⇒ 'x', 8 ⇒ 'h', 17 ⇒ 'q', 23 ⇒ 'w', 19 ⇒ 's', 1 ⇒ 'a', 22 ⇒ 'v', 6 ⇒ 'f
```

```
# reverse int to char dict
itoc = Dict(
    i => ch for (ch, i) ∈ ctoi
)
```

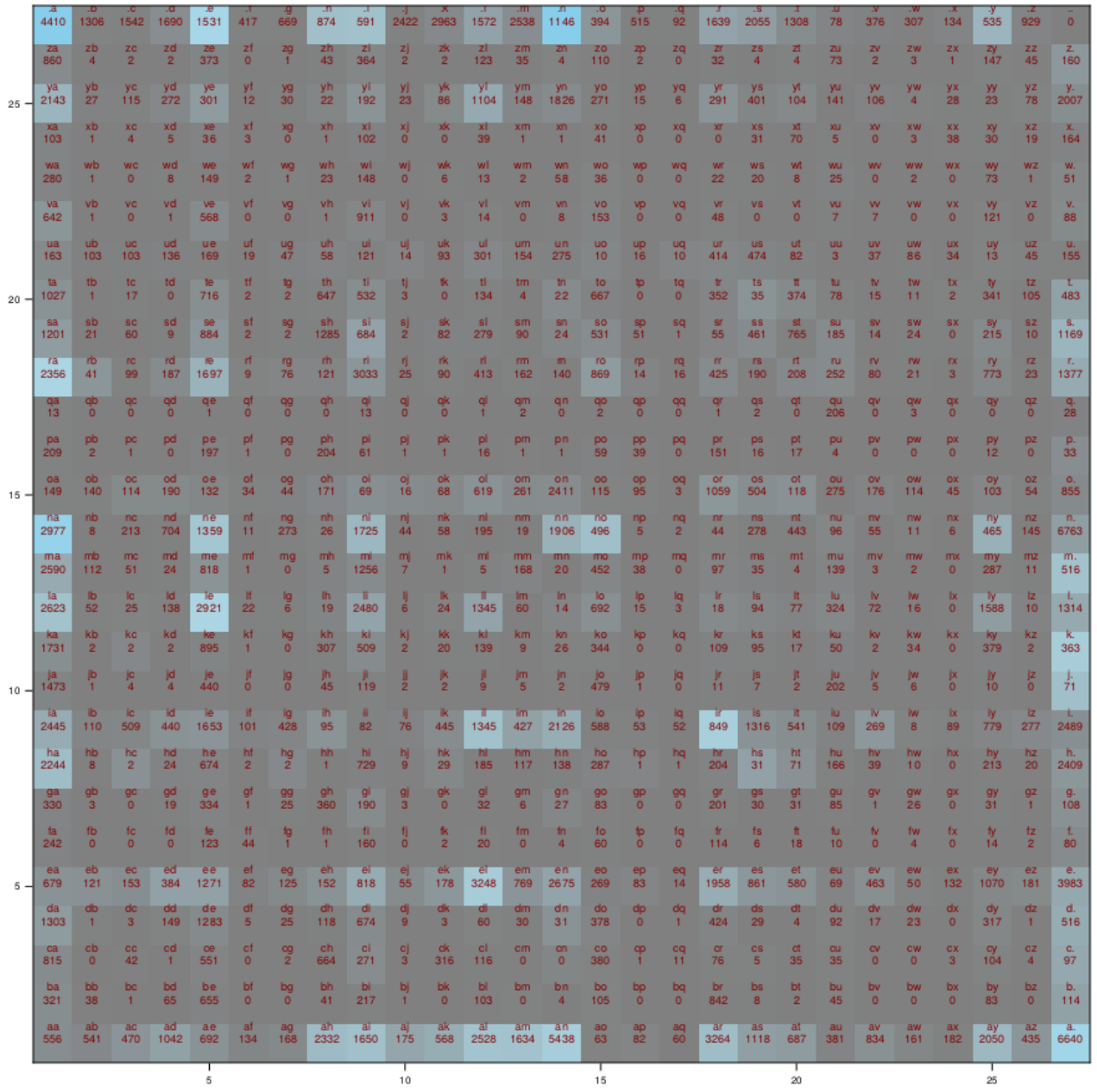
```
using CairoMakie
```

```
# heatmap(vbigrams, colorrange = (0, top10[1][2]),  colormap=Reverse(:viridis))
```

```
using Colors
```

```
λ (generic function with 1 method)
```

```
λ(x) = (x / maximum(vbigrams)) * 255 |> x -> floor(Int, x)
```



```
begin
    f = Figure(resolution = (900, 900), fontsize=8)

    Axis(f[1, 1], aspect = DataAspect(), backgroundcolor = :gray90)

    heatmap!(vbigrams, colorrange = (0, top10[1][2]), colormap=["grey", "lightblue", "skyblue"],
        # colormap=Colors.colormap("Blues", N*N),
        overdraw=true,
        transparency=true)

    for i ∈ 1:N, j ∈ 1:N
        text!(
            (j, i),
            text = string("$(itoc[i])$(itoc[j])", "\n$(vbigrams[i, j])"),
            align = (:center, :baseline),
            textsize=9,
            color="maroon"
        )
    end
    f
end
```

```
[556, 541, 470, 1042, 692, 134, 168, 2332, 1650, 175, 568, 2528, 1634, 5438, 63, 82, 60, 3264, 1118, 687, 381, 834, 161, 182, 2050,
```

```
vbigrams[1, :]
```

```
begin
    vprobs = Float64.(vbigrams[1, :]) / sum(vbigrams[1, :])
    @assert sum(vprobs) ≈ 1.0
end
```

```
27
```

```
length(vprobs)
```

```
[0.40418, 0.289218, 0.306602]
```

```
begin
    # Example
    using Random
    Random.seed!(42);
    p = rand(3)
    p /= sum(p)
end
```

```
using StatsBase
```

```
P = 27×27 Matrix{Float64}:
 0.0164249  0.0159825    0.0138889    …  0.0604801   0.0128568    0.19583
 0.120509   0.0145958    0.000748503     0.0314371   0.000374251  0.0430389
 0.229278   0.000280978  0.012082        0.0295027   0.00140489   0.0275358
 0.236104   0.000362122  0.000724244     0.0575774   0.000362122  0.0936085
 0.0332518  0.00596577   0.00753056      0.0523716   0.00889976   0.194817
 0.26073    0.00107296   0.00107296   …  0.0160944   0.00321888   0.0869099
 0.169396   0.00204708   0.000511771     0.0163767   0.00102354   0.055783
 ⋮                                   ⋱               ⋮
 0.247308   0.000769231  0.000384615     0.0469231   0.000384615  0.0342308
 0.293933   0.00209205   0.00104603      0.0774059   0.00209205   0.0543933
 0.143646   0.00276243   0.00690608      0.0428177   0.0276243    0.227901
 0.218709   0.00285627   0.0118331       0.00244823  0.00805876   0.204835
 0.355052   0.00206186   0.00123711   …  0.0610309   0.0189691    0.0663918
 0.137586   0.0407673    0.0481285       0.0167187   0.0290081    3.11915e-5
```

```
# and now use the trick of adding 1 everywhere, to avoid the 0 prob - which may result in NaN values

P = Float64.(vbigrams[1:N, 1:N] .+ 1.) ./ sum(vbigrams .+ 1, dims=2) # sum over rows
```

```
for ix ∈ 1:N
    @assert sum(P[ix, :]) ≈ 1  # all rows must sum to 1
end
```

```
rng = MersenneTwister(42)
```

```
rng = MersenneTwister(42)
```

```
for jx ∈ 1:32
    ix = 1
    chars_out = Char[]
    while true
        v_probs = P[ix, :] # v_probs = fill(1.0/N, N) # uniform
        ix = StatsBase.sample(rng, 1:N, ProbabilityWeights(vprobs), 1; replace=true)[1]
        ix == N && break
        push!(chars_out, itoc[ix])
    end
    println("$(jx) ", join(chars_out))
end
```

```
1 vbcbc
2 nrnhnd
3 m
4
5 sm
6 kehnhtuimnhh
7 ntdnvyilr
8 znl
9 tny
10
11 mat
12
13 mnr
14
15 hm
16 rdsr
17 hjrnsvtkrvnn
18
19
20 ynn
21 ur
22 jr
23 rrlrvm
24 eu
25 ay
26 rbirr
27 yhvx
28 nh
29
30 r
31 ln
32 drvinvlnnnhhin
```

```julia
begin
    using Printf

    log_likelihood₁ = 0.0
    kx = 5
    for bigram ∈ keys(bigrams)
        ix, jx = ctoi[bigram[1]], ctoi[bigram[2]]

        p₁ = P[ix, jx]
        log_p₁ = log(p₁)
        global log_likelihood₁ += log_p₁
        println("""$(bigram[1])$(bigram[2]): $(@sprintf "%.4f" p₁) $(@sprintf "%.4f" log_p₁)""")
        global kx -= 1
        kx == 0 && break
    end

    nll = -log_likelihood₁  # negative log-likelihood
    println("log_likelihood: $(@sprintf "%2.4f" nll) | $(@sprintf "%2.4f" nll / N)")
end
```

```
xz: 0.0276 -3.5891
af: 0.0040 -5.5262
ka: 0.3418 -1.0735
pc: 0.0019 -6.2663
sk: 0.0102 -4.5848
log_likelihood: 21.0399 | 0.7793
```

NOTE: the aim of our work is to minimize the negative log-likelihood (nll), which is also about minimizing the average of nll (which summarize the quality of the model).)

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```