Úloha 1 - určovanie príbuznosti pomocou kompresie

In [1]:
```
import gzip

def loadfasta(filename,verbose=0):
    """ Parses a classically formatted and possibly
        compressed FASTA file into a dictionary where the key
        for a sequence is the first part of its header without
        any white space; if verbose is nonzero then the identifiers
        together with lengths of the read sequences are printed"""
    if (filename.endswith(".gz")):
        fp = gzip.open(filename, 'rt')
    else:
        fp = open(filename, 'r')
    # split at headers
    # data = fp.read().split('>')
    data = fp.read()
    data = data.split('>')
    fp.close()
    # ignore whatever appears before the 1st header
    data.pop(0)
    # prepare the dictionary
    D = {}
    for sequence in data:
        lines = sequence.split('\n')
        header = lines.pop(0).split()
        key = header[0]
        D[key] = ''.join(lines)
        if verbose:
            print("Sequence %s of length %d read" % (key,len(D[key])))
    return D

seq = loadfasta('Seq.fasta')
```

First we read `Seq.fasta` file. We do that using the `loadfasta` method from seminar.

In [2]:
```
from gzip import compress as gzip_compress
from os.path import getsize, splitext
from subprocess import run

def gencompress_size(genom, reference=None):
    gename = "gene"
    genomfile = gename+'.tmp'

    with open(genomfile, "w+") as file:
        file.write(genom)

    if not reference == None:
        refname = "ref"
        refile = refname+'.tmp'

        with open(refile, "w+") as file:
            file.write(reference)

        run(["GenCompress.exe", genomfile, "-c", refile])
    else:
        run(["GenCompress.exe", genomfile])

    return getsize(gename+".GEN")

def gzipcompress_size(genom, reference=None):
    size = len(gzip_compress(bytes(genom, encoding='utf8')))

    if not reference == None:
        size = len(gzip_compress(bytes(reference + genom, encoding='utf8'))) - size
    return size

compressize = {
    "gen": gencompress_size,
    "gzip": gzipcompress_size
}
```

Then we define methods for computing size of compressed sequences with or without reference sequences using different algorithms (**GenCompress** and **gzip**). Beside the methods we define compressize switch variable for more transparent use.

In [3]:
```
compressize['gen'](seq['A'], seq['B'])
```

Out[3]: 1377

In [4]:
```
compressize['gzip'](seq['A'], seq['B'])
```

Out[4]: 1354

Now we need to define distances. The distance will be computed using the formula from the assignment and expressed in percentage rounded to 0 digits.

In [5]:
```
def distance(genA, genB, compress = compressize['gen']):
    return round((1 - (compress(genA) - compress(genA, genB)) / compress(genA + gen
B)) * 100)
```

In [6]:
```
distance(seq['A'], seq['B'], compressize['gen'])
```

Out[6]: 95

Distance table computation:

In [7]:
```
from math import inf

def distance_table(sequencedict, compress = compressize['gen']):
    table = [[inf for x in range(len(sequencedict))] for y in range(len(sequencedic
t))]
    names = [key for key in sequencedict.keys()]
    for A in range(len(names)):
        for B in range(len(names)):
            if A < B:
                table[A][B] = distance(sequencedict[names[A]], sequencedict[names[B
]], compress)
            elif A == B:
                table[A][B] = 0
            else:
                table[A][B] = table[B][A]

    return names, table
```

In [8]:
```
distance_table(seq, compressize['gzip'])
```

Out[8]:
```
(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'],
 [[0, 83, 96, 81, 74, 81, 68, 62],
  [83, 0, 90, 50, 62, 46, 72, 66],
  [96, 90, 0, 60, 50, 58, 62, 57],
  [81, 50, 60, 0, 67, 45, 75, 68],
  [74, 62, 50, 67, 0, 80, 80, 74],
  [81, 46, 58, 45, 80, 0, 75, 68],
  [68, 72, 62, 75, 80, 75, 0, 79],
  [62, 66, 57, 68, 74, 68, 79, 0]])
```

Compute both tables:

In [9]:
```
tables = {}
for compression in compressize:
    print('Computing distance table using compression:', compression)
    tables[compression] = distance_table(seq, compressize[compression])
```

```
Computing distance table using compression: gen
Computing distance table using compression: gzip
```

In [10]:
```
from pprint import pprint
pprint(tables)
```

```
{'gen': (['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'],
         [[0, 95, 96, 95, 95, 95, 96, 97],
          [95, 0, 73, 50, 69, 49, 91, 93],
          [96, 73, 0, 76, 72, 76, 93, 94],
          [95, 50, 76, 0, 71, 44, 91, 92],
          [95, 69, 72, 71, 0, 71, 90, 92],
          [95, 49, 76, 44, 71, 0, 91, 93],
          [96, 91, 93, 91, 90, 91, 0, 90],
          [97, 93, 94, 92, 92, 93, 90, 0]]),
 'gzip': (['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'],
          [[0, 83, 96, 81, 74, 81, 68, 62],
           [83, 0, 90, 50, 62, 46, 72, 66],
           [96, 90, 0, 60, 50, 58, 62, 57],
           [81, 50, 60, 0, 67, 45, 75, 68],
           [74, 62, 50, 67, 0, 80, 80, 74],
           [81, 46, 58, 45, 80, 0, 75, 68],
           [68, 72, 62, 75, 80, 75, 0, 79],
           [62, 66, 57, 68, 74, 68, 79, 0]])}
```

From the tables we just generated we can see the probable affinity of species.

For **gen**:

1. F and D (44)
2. B and F (49)
3. B and D (50)

Thus, we can conclude that *F*, *D* and *B* are very close to each other.

For **gzip**:

1. F and D (45)
2. F and B (46)
3. B and D (50)

Thus, we can see that *F*, *D* and *B* are very close in this compression as well.

From this we can conclude that in both polygenetic trees nodes representing these species will look similar to:

```
    /--B
--|     /--F
    \--|
        \--D
```

The result is not same in all cases though. For example the last row denoting the affinity of *H* to other species differs a lot. For **gen** the compression denotes almost none affinity to any of the other species. For **gzip**, however, there is very high affinity to *C* - 57 (for **gen** the affinity to *C* was 94).

From this we can conclude that the tree will differ for *H* - for **gzip** tree there should be visible affinity to *C* and for **gen** the *H* node should be somewhere at the top layers of the tree, with high distances to other nodes (closest should be *G* with distance 90).

As of which algorithm is more suitable for polygenetic tree creation, I would say that **GenCompress** suits better the need. Our aim is not to compress genomes as much as possible but to find similarities in genoms. With **GenCompress** we actually try to compress one genom using the other. With **gzip** we just concatenate both sequences and hope that it will use the first for referencing the other. But that does not have to be the case and the compression algorithm may have used some other method to obtain the best result and therefore we cannot *trust* the **gzip** result as much as the **GenCompress** result.

Now we need to create the phylogenetic tree. We will use `ete3` library designed for phylogenetic tree representation. For more information about the library see ete3 (http://etetoolkit.org/ipython_notebook/).

Before we can create the tree, we need to cluster the data using the distance matrix. For this purpose I use dedupe-hcluster (https://pypi.org/project/dedupe-hcluster/) module.

In [22]:
```
import sys
!{sys.executable} -m pip install dedupe-hcluster
!{sys.executable} -m pip install ete3
```

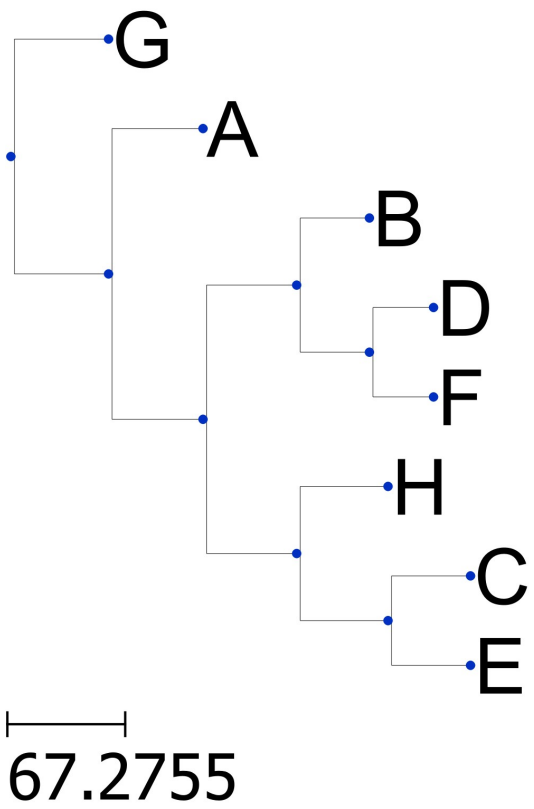
```
Requirement already satisfied: dedupe-hcluster in c:\users\jakub\anaconda3\lib\s
ite-packages (0.3.6)
Requirement already satisfied: numpy>=1.12.1; python_version == "3.6" in c:\user
s\jakub\anaconda3\lib\site-packages (from dedupe-hcluster) (1.14.3)
Requirement already satisfied: future in c:\users\jakub\anaconda3\lib\site-packa
ges (from dedupe-hcluster) (0.17.1)

You are using pip version 18.0, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.

Requirement already satisfied: ete3 in c:\users\jakub\anaconda3\lib\site-package
s (3.1.1)

You are using pip version 18.0, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.
```

In [35]:
```
from hcluster import linkage, to_tree
from ete3 import Tree

def build_tree(table):
    #hcluster part
    l = linkage(table, "single") # link the data
    T = to_tree(l) # create a tree

    #ete2 section
    root = Tree()
    root.dist = 0
    root.name = "root"
    item2node = {T: root}
    names = "ABCDEFGH"

    to_visit = [T]
    while to_visit: # While there are items to visit
        node = to_visit.pop() # take the first
        cl_dist = node.dist / 2.0 # take the cluster distance
        for ch_node in [node.left, node.right]: # fill in the binary sons
            if ch_node:
                ch = Tree() # create a tree
                ch.dist = cl_dist # set its distance
                ch.name = names[ch_node.id] if ch_node.id < len(names) else str(ch_
node.id) # and name
                item2node[node].add_child(ch) # Add it as a child of parent node
                item2node[ch_node] = ch # Set it as a node
                to_visit.append(ch_node) # Visit it

    # This is your ETE tree structure
    return root
```

Finally we can build the tree and render it using render function of ete3 tree node.

In [58]:
```
tree = build_tree(tables['gzip'][1])
_ = tree.render('tree1.png', w=1000, units='px')
```

Phylogenetic tree #1 - gzip



In [59]:
```
tree = build_tree(tables['gen'][1])
_ = tree.render('tree2.png', w=1000, units='px')
```

Phylogenetic tree #2 - GenCompress