Greedy-Superstring

The Algorithm

The naive implementation of the *Greedy-Superstring* implementation first calculates the overlap of every pair of disjoint strings in the inputs, this procedure runs in $\mathcal{O}(n^3)$ where n is the length of the input. Afterwards the two strings with the largest overlap are merged into one. The algorithm then calculates the new overlaps between all pairs leftover. This has to be repeated n times, yielding a runtime of $\mathcal{O}(n^4)$.

The idea behind the $\mathcal{O}(n^3)$ implementation of the algorithm is that in every merge step two strings with a maximal overlap are substituted by their merger. Afterward we only have to calculate the overlaps of the new merged string with all the other leftover strings. This can be done in $\mathcal{O}(n^2)$ and since this procedure has to be repeated at maximum n times we arrive at a $\mathcal{O}(n^3)$ runtime.

In [2]:

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 13 11:20:08 2021

@author: Florian Gottscheber, Niclas Krembsler, Phillip Kojo Ampadu, Christian Singer
"""
# Numpy is needed because of the array data structure it provides.
import numpy as np
```

In [3]:

```
#Algorithm 4: Set of disjoint strings
def disjoint_string(F: list) -> list:
    dis_str = []
    for s in F:
        s_super = []
        for e in F:
            if s in e:
                s_super.append(e)
        if len(s_super) == 1:
            dis_str.append(s_super[0])

    return dis_str

# Helper for Algorithm 5
def find_overlap(frag1, frag2):
        # Compare possible overlaps from biggest to smallest.
        for i in range(len(frag1)-1,0,-1):
            # Beginning frag1 matches end frag2
            if frag1[:i] == frag2[-i:]:
                return i
            # End frag1 matches beginning frag2.
            elif frag1[-i:] == frag2[:i]:
                return -i

        return 0
```

In [4]:

```
def GreedySuperstring(F: list) -> str:
    # Remove all sequences that are subsequences.
    dis_str = disjoint_string(F)
    n_frags = len(dis_str)
    # Initialize memory table for overlap between the i-th and j-th fragment in each iteration
    Overlaps = np.zeros((n_frags, n_frags))
    for i in range(n_frags):
        frag1 = dis_str[i]
        for j in range(i+1,n_frags):
            frag2 = dis_str[j]
            Overlaps[i,j] = find_overlap(frag1,frag2)

# Every Iteration two fragments are merged, reducing n_frags by 1.
for i in range(n_frags):

    # End merging process if no strings overlap anymore.
    if np.max(np.abs(Overlaps)) == 0:
        return "".join(dis_str)

    # Determine the pair of fragments for which the overlap is maximal.
    max_overlap_pos = np.argmax(np.abs(Overlaps))
    # Since type(max_overlap_pos) is float the actual indices have to be interfered via this formula.
    merge_idx1, merge_idx2 = max_overlap_pos // Overlaps.shape[0], max_overlap_pos % Overlaps.shape[1]

    # Actual fragments to be merged
    m1 = dis_str[merge_idx1]
    m2 = dis_str[merge_idx2]

    # Substitute the string m2 with empty strings, m1  will be substituted by the merged string.
    # len(dis_str) remains constant throughout the for-loop.
    dis_str[merge_idx2] = ""

    # Numerical value of the maximum overlap decided on whether to merge frag1 onto frag2 or the reverse.
    max_overlap = int(Overlaps[merge_idx1, merge_idx2])

    # Merge non overlapping beginning of m1 with m2.
    if max_overlap < 0:
        merged_string = m1[:max_overlap] + m2

    # Merge m2 with non overlapping end of m1.
```

```
        else:
            merged_string =  m2 + m1[max_overlap:]

        # Substitute m1 with the merged string
        dis_str[merge_idx1] = merged_string

        # Calculate new overlaps of the merged string with all the other strings left.
        # Since m2 was substituted by "" there won't be any overlaps possible anymore.

        # Since Overlaps is symmetric both the merge_idx'th column and row have to be calculated again.
        for j in range(merge_idx1+1, len(Overlaps)):
                Overlaps[merge_idx1,j] = find_overlap(merged_string, dis_str[j])

        for i in range(merge_idx1):
                Overlaps[i,merge_idx1] = find_overlap(dis_str[i], merged_string)

        # All overlaps of strings with the merge_idx2'th element of dis_str are zero.
        for j in range(merge_idx2, len(Overlaps)):
                Overlaps[merge_idx2,j] = 0

        for i in range(merge_idx2):
                Overlaps[i,merge_idx2] = 0

        Overlaps[merge_idx1,merge_idx1] = 0

    return "".join(dis_str)
```

Unknown Text

In [8]:

```
with open('Textfragmente.txt', "r") as f:
    lines = f.read().splitlines()

text = GreedySuperstring(lines)
print(text)
```

```
Das Wohltemperierte Klavier (BWV 846â€“893) ist eine Sammlung von PrÃ¤ludien und Fugen fÃ¼r ein Tasteninstrument von Johann Sebast
ian Bach in zwei Teilen. Teil I stellte Bach 1722, Teil II 1740/42 fertig. Jeder Teil enthÃ¤lt 24 Satzpaare aus je einem PrÃ¤ludium
und einer Fuge in allen Dur- und Molltonarten, chromatisch aufsteigend angeordnet von C-Dur bis h-Moll. Mit dem Begriff Clavier, de
r alle damaligen Tasteninstrumente umfasste, lieÃŸ Bach die Wahl des Instruments fÃ¼r die AusfÃ¼hrung bewusst offen. Die Orgel sch
eidet in den meisten FÃ¤llen aus, da Bach keine separate Pedalstimme notierte oder als solche bezeichnete und die Orgeln seiner Ze
it mitteltÃ¶nig gestimmt waren. Der grÃ¶ÃŸte Teil des Werks ist offenbar fÃ¼r Clavichord oder Cembalo konzipiert. Nach einer Ã„uÃŸ
erung Johann Nikolaus Forkels hatte Bach eine Vorliebe fÃ¼r das Clavichord. Im Nekrolog von 1754 steht dagegen Ã¼ber Bach: Die Cla
vicymbale wuÃŸte er, in der Stimmung, so rein und richtig zu temperiren, daÃŸ alle Tonarten schÃ¶n und gefÃ¤llig klangen. Das Werk
wird heute sowohl auf dem Cembalo als auch auf dem modernen Klavier bzw. FlÃ¼gel gespielt.
```

Unknown DNA-Sequence, Part 1

In [11]:

```
with open('DNA-Fragmente 1.txt', "r") as f:
    lines = f.read().splitlines()

fragments1 = GreedySuperstring(lines)
print(fragments1)
```

```
TGTATACATGGAATATGTAAAGCTTTTATATGTCAGTCACACCTCAGTAAAGTGGTTTACCTATCTATCTATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTTCCTAAAAAAAGGAAGGGAGAAGA
GAGGAAAAGATGTTCAGGGAGCTACCATTTTGTTTCTAGCTGTGATTTTATAAAATGATAGACACTTTTATCTTTGTGTTACGTTCCTACCCCCAGTCCTCCAAATTATGGATCTGTGCCATTTGTACCG
TGGACTTTTCTGTTTTCTGGGATCTGGAGAGGAAGACTCAGTCCAGAATCCTCCCAGGGCCTTGAAAGTCCATCTCTGACCCAAAACAATCCAAGTAAGTACCTAATTCCTTTGGGAGTGGGTTGTGTAT
CTCACAGCAACAGAGAAAAAATAGTCACTTAAAAGTTTCTCTTTGACATCTGTAATGTATGTCAATAAATGAATTCTAAGTTAGTAGAGTTTGATGATTGACTTCAGTTGTAAACTCTTCTAGCCAGGAG
TTTTTCTTATACTCATTTTAAAAAAGAGAGAAACTAAAAAACAAAAAGAAGCAGAAGCAAAAGTTAATGAGTCTTAACAGTTGCTTACCTATTGAAAACTTATTTAGAAATACTCTTTTAACATTGTGGT
CACCTGAGTAAATCACTGGAGATAGTGCATTTCAGAAATGTCTCCGTTCTGATTCCATAAACAATTTGACTTGTATAGTGTGCTATATTTTGGTGATTTATCAAATCTTGATGTGAGTTTGGGAGTATTG
CTAATGTCAGATGACTTGGGAACTAAGAATAAGACATTTAACCTATGCTTAATTGAAATGAAATTTTTCCCTGAGGATGTTGCAACAAATACTGATGCAACTCCTGGTTAACTGATAAAGTACTGGCCAG
GGACAAAGCTCTCTTGCAGCAATTTCCCACCACGTACCTCTGCCCTCTCCTCACAGCTGGAGAGGGAAAGTCATGGAATCCTTGTCCTTCCTCTTGTTTCCACCTCTTCAAGATTGGGCCAATTGCAATG
GAATATCCATTGGTTGTGAGGCCTTTGTACTCTGCAAGGAAAAGAAAAGAAATGTGTGTATGTATGAGTGTGTGATGGAGCTAACTTTTCTACAATGTCTACTAACATGTCCTAGCCTTTACTTCATTCG
CCTGTTTCCTTCTCACAAAAACCCTGTATGGGAGTTTTTCTTTACTTTTTATTATTATTTTTTTGAGACAAAGTCTCGCTCTGTCTCCCAGGCTGGAGTGCAGTGGCGCTATATCGGCTCACTGCAGCCT
CCACCTCCCGGGTTCAAGCGATTCTCCTGCCTCAGCCTCCTGAGTAGCTGGTACTACAGGCGTGCACCACCATGCCACTATTTTTTGTATTTTTAGTAGCGGGACCTGAACTTGAGGGCGGGTCTTTCTG
ACTCCAAAGCCTCTTCCTGGCTACTCTGATATTGGCTATTGGCGGAGGCTGGGAAAACTTGAAATGGGGAATGCTTTCCATTTTGAATATTAATATGACAGGAAATATCAGATGGAAATATTTTTAAAAG
ATAGAGACGGGGTTTCACTATGTTGGCCAGACTGGTCTCGAACTCTTGACCTCAGGTGATCCGCCCGCCTCGGCTTCCCAGAGTGCTAGGATTACAGGCGTGTATGCCTATCCCCAGACTCTCTCCTCCT
CCTCACCTCATTGTCTCCCCGACTTATCCTAATGCGAATTGGGTTTTTATTCAGAAGGGAGGGGCAGGAGGGAATGACAAGTGACTCACCTTGAATTCTTCCTCTAAGAAACTCACACCTGAGCTTTGAG
CTATAAAGAAATCTGATGCTGTTTCTGGTGCTGTCTTAGAATCACTTCAGGAGTATTGACAAGAGGGGTAGGAACCCTTAGCCGTTTCCTGAAACCTCCTGCATAGGGCATTTTCGAGAGATTGCACCAT
CAGATGAGAAAACTGAGACTCAAAAAATACAAGTGACCCGTCCACAGGCAGATAGTTAGGAAATAATATTAGTGATAAATAAGAAGGCAGGAAGAAACTTTTGGAGGTGATGGATAGGTTTATGGTATAG
ATTGTGGTGGAGCCTGACTTACTTTAGTAATAAAATTGTCCAAGGACTAAATTTATAGATAAGATACCTCTTTGTCTCCTTATTGACAGAGTGAATGGGGCAACTGTGGCATTCCAGCCTGACAGGGGTG
ATTTGTAGCAAAATCGTCCTGAGACCCTTCCTC
```

Observation: The length of the sequence isn't divisible by three, hence there is no reading frame that can translate all triplets into amino acids.

In [26]:

```
len(fragments1) % 3
```

Out[26]:

```
2
```

Unknown DNA-Sequence, Part 2

In [12]:

```
with open('DNA-Fragmente 2.txt', "r") as f:
    lines = f.read().splitlines()

fragments2 = GreedySuperstring(lines)
print(fragments2)
```

```
GTGTATGCCTATCCCCAGACTCATCAAAGTGTATACATGGAATATGTAAAGCTTTTATATGTCAGTCACACCTCAGTAAAGTGGTTTACCTATCTATCTATCTATCTATCTATCTATCTATCTAAATTTT
```

TTTTTCTGTTCCTAAAAAAAGGAAGGGAGAAGAGAGGAAAAGATGTTCAGGGAGCTACCATTTTGTTTCTAGCTGTGATTTTATAAAATGATAGACACTTTTATCTTTGTGTTACGTTCCTACCCCCAGT
CCTCCAAATTATGGATCTGTGCCATTTGTACCGTGGACTTTTCTGTTTTCTGAGGATGTTGCAACAAATACTGATGCAACTCCTGGTTAACTGATAAAGTACTGGCCAGGGACAAAGCTCTCTTGTCCTG
AGACCCTTCCTCAAGATTTGCAGCAATTTCCCACCACGTACCTCTGCCCTCTCCTCACAGCTGGAGAGGGAAAGTCATGGAATCCTTGTCCTTCCTCTTGTTTCCACCTCTTCAAGATTGGGCCAATTGC
AATGGAATATCCATTGGTTGTGAGGCCTTTGTACTCTGCAAGGAAAAGAAAAGAAATGTGTGTATGTATGAGTGTGTGATGGAGCTAACTTTTCTACAATGTCTACTAACATGTCCTAGCCTTTACTTCA
TTCGCCTGTTTCCTTCTCACAAAAACCCTGTATGGGAGTTTTTCTTTACTTTTTATTATTATTTTTTTGAGACAAAGTCTCGCTCTGTCTCCCAGGCTGGAGTGCAGTGGCGCTATATCGGCTCACTGCA
GCCTCCACCTCCCGGGTTCAAGCGATTCTCCTGCCTCAGCCTCCTGAGTAGCTGGTACTACAGGCGTGCACCACCATGCCACTATTTTTTGTATTTTTAGTAGAGACGGGGTTTCACTATGTTGGCCAGA
CTGGTCTCGAACTCTTGACCTCAGGTGATCCGCCCGCCTCGGCTTCCCAGAGTGCTAGGATTACAGGCGTGAGCCACTGCGCCCAGCCAGGAGTTTTTCTTATACTCATTTTACAGATGAGAAAACTGAG
ACTCAAAAAATACAAGTGACCCGTCCACAGGCAGATAGTTAGGAAGTAGCGGGACCTGAACTTGAGGGCGGGTCTTTCTGACTCCAAAGCCTCTTCCTGGCTACTCTGATATTGGCTATTGGCGGAGGCT
GGGAAAACTTGAAATGGGGAATGATCGGGGAGCGGCGAGGGGGGACCAGCCGTTAAGCATTCCAGCCTGACAGGGGTGATTTGTTAAACCCAGGAACTAGTTAGACGTTTCCTGAAACCTCCTGCATAGG
GCATTTTCGAGAGATTGCACCATCACTCTCTCCTCCTCCTCACCTCATTGTCTCCCCGACTTATCCTAATGCGAAATTGGATTTGTAGCAAAATCGCTGGGATCTGGAGAGGAAGACTCAGTCCAGAATC
CTCCCAGGGCCTTGAAAGTCCATCTCTGACCCAAAACAATCCAAGTAAGTACCTAATTCCTTTGGGAGTGGGTTGTGTATCTCACAGCAACAGAGAAAAAATAGTCACTTAAAAGTTTCTCTTTGACATC
TGTAATGTATGTCAATAAATGAATTCTAAGTTAGTAGAGTTTGATGTAAAGTCCTGAAAATTAAAAAAGAGAGAAACTAAAAAACAAAAAGAAGCAGAAGCAAAAGTTAATGAGTCTTAACAGTTGCTTA
CCTATTGAAAACTTATTTAGAAATACTCTTTTAACATTGTGGTCACCTGAGTAAATCACTGGAGATAGTGCATTTCAGAAATGTCTCCGTTCTGATTCCATAAACAATTTGACTTGTATAGTGTGCTATA
TTTTGGTGATTTATCAAATCTTGATGTGAGTTTGGGAGTATTGCTAATGTCAGATGACTTGGGAACTAAGAATAAGACATTTAACCTATGCTTAATTGAAATGAAATTTTTCCCTAGAAGAAGAGTAGGT
GGAAAAAGTCTTCTTTCTTGACTTCAGTTGTAAACTCTTCTATTGCTTTCCATTTTGAATATTAATATGACAGGAAATATCAGATGGAAATATTTTTAAAAGATAGAAATGTGAGTATGACGAAGAACTT
TAGTAATAAAATTGTCCAAGGACTAAATTTATAGATAAGATACCTCTTTGTCTCCTTATTGACAGAGTGAATGGGGCAACTGTGGAGCCTGACTTACTTCTTTTAATTGGGTTTTTATTCAGAAGGGAGG
GGCAGGAGGGAATGACAAGTGACTCACCTTGAATTCTTCCTCTAAGAAACTCACACCTGAGCTTTGAGCTATAAAGAAATCTGATGCTGTTTCTGGTGCTGTCTTAGAATCACTTCAGGAGTATTGACAA
GAGGGGTAGGAACCCTTAGAAATAATATTAGTGATAAATAAGAAGGCAGGAAGAAACTTTTGGAGGTGATGGATAGGTTTATGGTATAGATTGTGGTGATGATTTAATGA

Unknown DNA-Sequence, Part 3

In [13]:

```
with open('DNA-Fragmente 3.txt', "r") as f:
    lines = f.read().splitlines()

fragments3 = GreedySuperstring(lines)
print(fragments3)
```

ACTCTCTCCTCCTCCTCACCTCATTGTCTCCCCGACTTATCCTAATGCGAAATTGGATTCTGAGCATTTGTAGCAAAATCGCTGGGATCTGGAGAGGAAGACTCAGTCCAGAATCCTCCCAGGGCCTTGA
AAGTCCATCTCTGACCCAAAACAATCCAAGTAAGTACCTAATTCCTTTGGGAGTGGGTTGTGTATCTCACAGCAACAGAGAAAAAATAGTCACTTAAAAGTTTCTCTTTGACATCTGTAATGTATGTCAA
TAAATGAATTCTAAGTTAGTAGAGTTTGATGTAAAGTCCTGAAAATTAAAAAAGAGAGAAACTAAAAAACAAAAAGAAGCAGAAGCAAAAGTTAATGAGTCTTAACAGTTGCTTACCTATTGAAAACTTA
TTTAGAAATACTCTTTTAACATTGTGGTCACCTGAGTAAATCACTGGAGATAGTGCATTTCAGAAATGTCTCCGTTCTGATTCCATAAACAATTTGACTTGTATAGTGTGCTATATTTTGGTGATTTATC
AAATCTTGATGTGAGTTTGGGAGTATTGCTAATGTCAGATGACTTGGGAACTAAGAATAAGACATTTAACCTATGCTTAATTGAAATGAAATTTTTCCCTAGAAGAAGAGTAGGTGGAAAAAGTCTTCTT
TCTTGACTTCAGTTGTAAACTCTTCTATTGCTTTCCATTTTGAATATTAATATGACAGGAAATATCAGATGGAAATATTTTTAAAAGATAGAAATGTGAGTATGACGAAGAACTTTAGTAATAAAATTGT
CCAAGGACTAAATTTATAGATAAGATACCTCTTTGTCTCCTTATTGACAGAGTGAATGGGGCAACTGTGGAGCCTGACTTACTTCTTTTAATTGGGTTTTTATTCAGAAGGGAGGGGCAGGAGGGAATGA
CAAGTGACTCACCTTGAATTCTTCCTCTAAGAAACTCACACCTGAGCTTTGAGCTATAAAGAAATCTGATGCTGTTTCTGGTGCTGTCTTAGAATCACTTCAGGAGTATTGACAAGAGGGGTAGGAACCC
TTAGAAATAATATTAGTGATAAATAAGAAGGCAGGAAGAAACTTTTGGAGGTGATGGATAGGTTTATGGTATAGATTGTGGTGATGATTTAATGAGTGTATGCCTATCCCCAGACTCATCAAAGTGTATA
CATGGAATATGTAAAGCTTTTATATGTCAGTCACACCTCAGTAAAGTGGTTTACCTATCTATCTATCTATCTATCTATCTATCTATCTAAATTTTTTTTTCTGTTCCTAAAAAAAGGAAGGGAGAAGAGA
GGAAAAGATGTTCAGGGAGCTACCATTTTGTTTCTAGCTGTGATTTTATAAAATGATAGACACTTTTATCTTTGTGTTACGTTCCTACCCCCAGTCCTCCAAATTATGGATCTGTGCCATTTGTACCGTG
GACTTTTCTGTTTTCTGAGGATGTTGCAACAAATACTGATGCAACTCCTGGTTAACTGATAAAGTACTGGCCAGGGACAAAGCTCTCTTGTCCTGAGACCCTTCCTCAAGATTTGCAGCAATTTCCCACC
ACGTACCTCTGCCCTCTCCTCACAGCTGGAGAGGGAAAGTCATGGAATCCTTGTCCTTCCTCTTGTTTCCACCTCTTCAAGATTGGGCCAATTGCAATGGAATATCCATTGGTTGTGAGGCCTTTGTACT
CTGCAAGGAAAAGAAAAGAAATGTGTGTATGTATGAGTGTGTGATGGAGCTAACTTTTCTACAATGTCTACTAACATGTCCTAGCCTTTACTTCATTCGCCTGTTTCCTTCTCACAAAAACCCTGTATGG
GAGTTTTTCTTTACTTTTTATTATTATTTTTTTGAGACAAAGTCTCGCTCTGTCTCCCAGGCTGGAGTGCAGTGGCGCTATATCGGCTCACTGCAGCCTCCACCTCCCGGGTTCAAGCGATTCTCCTGCC
TCAGCCTCCTGAGTAGCTGGTACTACAGGCGTGCACCACCATGCCACTATTTTTTGTATTTTTAGTAGAGACGGGGTTTCACTATGTTGGCCAGACTGGTCTCGAACTCTTGACCTCAGGTGATCCGCCC
GCCTCGGCTTCCCAGAGTGCTAGGATTACAGGCGTGAGCCACTGCGCCCAGCCAGGAGTTTTTCTTATACTCATTTTACAGATGAGAAAACTGAGACTCAAAAAATACAAGTGACCCGTCCACAGGCAGA
TAGTTAGGAAGTAGCGGGACCTGAACTTGAGGGCGGGTCTTTCTGACTCCAAAGCCTCTTCCTGGCTACTCTGATATTGGCTATTGGCGGAGGCTGGGAAAACTTGAAATGGGGAATGATCGGGGAGCGG
CGAGGGGGGACCAGCCGTTAAGCATTCCAGCCTGACAGGGGTGATTTGTTAAACCCAGGAACTAGTTAGACGTTTCCTGAAACCTCCTGCATAGGGCATTTTCGAGAGATTGCACCATCA

Observation: The length of the sequence is divisible by three, hence there is now a reading frame that can translate all triplets into amino acids.

In [23]:

```
len(fragments3) % 3
```

Out[23]:

0