BMS 321 Project, Fall 2019

Diabetes Mellitus associated SNP Detection Algorithm

Done by:

Mayar Mansour	201700072
Muhammad Salah ElSadany	201700857

Supervised by:
Dr Eman Mostafa

Diabetes Mellitus associated SNP Detection Algorithm

Hash Indexing Algorithm

Mayar Mansour
Computational Biology and Genomics, Biomedical Sciences, University of Science and Technology, Zewail City, Giza, Egypt, s-mayarmansour@zewailcity.edu.eg
Muhammad ElSadany
Computational Biology and Genomics, Biomedical Sciences, University of Science and Technology, Zewail City, Giza, Egypt, s-muhammadelsadany@zewailcity.edu.eg

Abstract

Single Nucleotide Polymorphisms (SNPs) are one of the common genetic variations in which there is a single nucleotide base different among individuals. They can be dealt with as biological markers that help scientists in locating genes associated with diseases. Diabetes Mellitus is a chronic disease in which the pancreas does not produce enough insulin or the body cannot effectively use the insulin produced by it. This program aims to identify SNPs found in an entered sequence by the user by comparing it with the original insulin gene. Also, it links these found SNPs to a built database to check if these found SNPs have known clinical pathogenicity or not. The coding part was mainly categorized into 3 parts: 1. building a database. 2. reading file and Hash Algorithm. 3. getting position of SNP and retrieving data from database. The experimenting code was done by using known sequences of specific SNPs that are already recorded in the database. The results of testing the code showed all desired outputs which signify the important role desired by building the program. Based on the results and their evaluation, the program can deal with any entered sequence with the same length of insulin gene only, otherwise, it cannot proceed as the program mainly depends on aligning the entered sequence with the original one.

KEYWORDS Hash Algorithm, SNP Detection, Insulin Gene, Diabetes Mellitus

1. Introduction

Detecting genetic variation in the human genome is very important for understanding the variation in phenotype, especially in coding regions or regulatory regions. These mutations can affect the susceptibility to disease or cancer. Single Nucleotide Polymorphisms (SNPs) are the most common genetic variation where a single nucleotide either adenine, thymine, cytosine or guanine changes from one individual to the other in the same species (figure 1). Scientists found that they occur almost once every 1000 Nucleotide [1]. Diabetes Mellitus commonly known as diabetes is a chronic disease that occurs either when the pancreas does not produce enough insulin or when the body cannot effectively use the insulin it produces [2].

In 2016, diabetes was the direct cause of 1.6 million deaths and in 2012 high blood glucose was the cause of another 2.2 million deaths [2]. Poorly controlled diabetes over time leads to serious damage to many of the body's systems, especially the nerves and blood vessels as scientists found that SNPs associated with diabetes can act as predispositions for heart diseases and neuropathy. That is why it is very crucial to identify the SNPs associated with diabetes to evaluate if they are associated with other diseases in the long run or not. There are several algorithms for detecting SNPs in a certain gene, but most of them either depend on BWA or bowtie and that leads them to give false results [3].

2. Related work and Survey

At present, SNP detection strategies for Next Generation Sequencing (NGS) output are MAQ [4], SOAPsnp [5], SNVmix2 [6] and Bcftools [7], GATK [8]. MAQ and SOAPsnp are quick collection alignment tools that make use of the mass fraction deduced collection and the alignment of alignments. MAQ uses the pairing statistics to estimate the possibility of each alignment read error, and the Bayesian statistical model to assess the very last Genotype error probability. The combined binomial model is used to find out the SNP for SNVmix2, giving the self-assurance score for every SNP. Bcftools and GATK use Samtools as the idea for estimating SNPs using Bayesian chance estimates. BWA or Bowtie contrast is presently an incredibly good comparison of the software, but there are numerous problems [3], such as false positive, false negative, the opposite sequence cannot shape the problem; 16G memory can run more problems. Moreover, the results of a systematic review article aiming to evaluate and compare the capability of multiple aligners to provide a guiding resource for choosing suitable aligners dependent on the user's aim supported that. As it affirmed the fact that the Hash indexing based algorithm provides higher sensitivity and higher precision when dealing with mismatch and indel errors existed in simulated datasets [9]. That is why in our proposal we decided to use a Hash table indexing algorithm in detecting the SNPs in the insulin gene.

3. Problem Definition

As mentioned before, SNPs detection is very important as it can be a marker for diseases and their pathogenicity. The same is for Diabetes Mellitus detection and its linkage to SNPs related to the insulin gene. The proposed program mainly deals with SNPs related to Diabetes (pathogenic or likely pathogenic SNPs). The program makes it easier for checking the SNPs position for any entered sequence and rechecking that with the built database of already known SNPs with their clinical pathogenicity. It takes a lot of time when it comes to comparing sequences manually; that’s why we thought about this coding program that aims to give the needed exact output in less time with high efficiency. So, it can be said that the program saves all known SNPs with their positions for insulin gene, as in the database building part, and compares any entered sequence with the original insulin and gives all SNPs found if any. Therefore, it solves two linked problems together: 1. Having all pathogenic and likely pathogenic insulin SNPs in a saved database for further needs that can be easily navigated and retrieved. 2. Comparing any entered sequence from a file with the original insulin gene and giving output if all found SNPs to be checked with the database.

4. Proposed Method

Coding Pipeline:

First step for the coding building was the database. The built database is specific to all the SNPs found in the insulin gene that are correlated clinically to being pathogenic or likely pathogenic. These SNPs were extracted from the dbsnps in the NCBI [10].

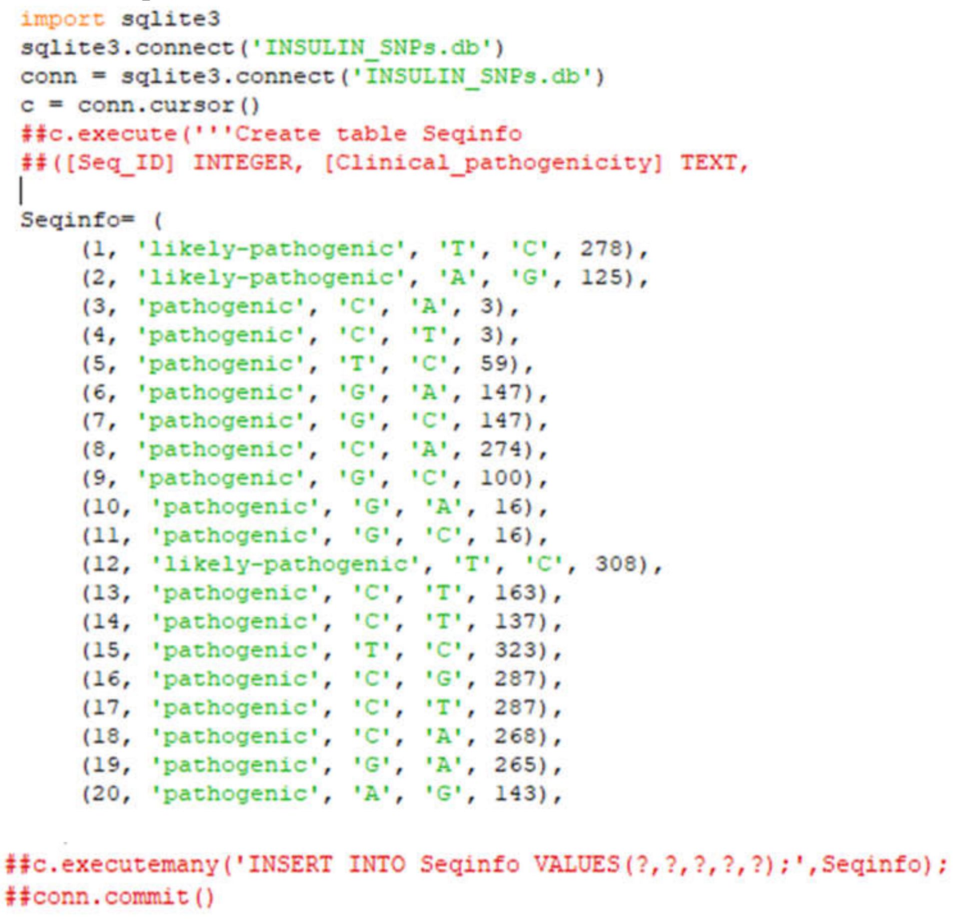
The second step was working on a sequence matching algorithm based on hash position index. This step is to compare the entered sequence from the user with the original gene sequence.

The third step was checking the differences between both sequences if present and get back the nucleotides' positions.

The fourth and last step was fetching the database searching for the position and its clinical pathogenicity and if not found it gives a "not defined pathogenicity" statement.

Algorithm:

Firstly, The building the database of known pathogenic and likely pathogenic SNPs of the insulin gene. The coding part of the database was as follows in figure (1). The database has only 1 table, named Seqinfo, that contains a sequence ID, clinical pathogenicity, original base, new base, and position.

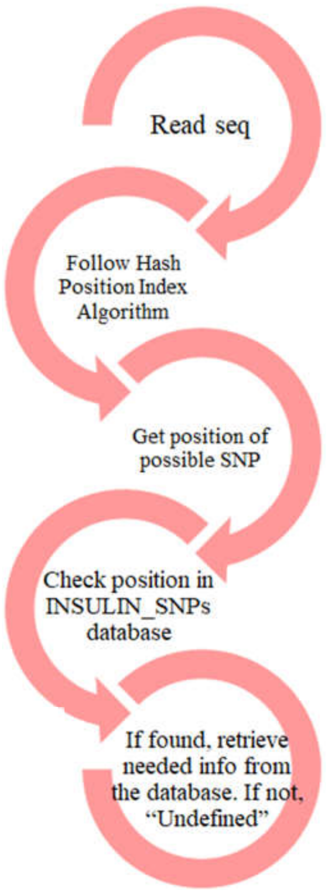


```
import sqlite3
sqlite3.connect('INSULIN_SNPs.db')
conn = sqlite3.connect('INSULIN_SNPs.db')
c = conn.cursor()
##c.execute('''Create table Seqinfo
##([Seq_ID] INTEGER, [Clinical_pathogenicity] TEXT,
|
Seqinfo= (
    (1, 'likely-pathogenic', 'T', 'C', 278),
    (2, 'likely-pathogenic', 'A', 'G', 125),
    (3, 'pathogenic', 'C', 'A', 3),
    (4, 'pathogenic', 'C', 'T', 3),
    (5, 'pathogenic', 'T', 'C', 59),
    (6, 'pathogenic', 'G', 'A', 147),
    (7, 'pathogenic', 'G', 'C', 147),
    (8, 'pathogenic', 'C', 'A', 274),
    (9, 'pathogenic', 'G', 'C', 100),
    (10, 'pathogenic', 'G', 'A', 16),
    (11, 'pathogenic', 'G', 'C', 16),
    (12, 'likely-pathogenic', 'T', 'C', 308),
    (13, 'pathogenic', 'C', 'T', 163),
    (14, 'pathogenic', 'C', 'T', 137),
    (15, 'pathogenic', 'T', 'C', 323),
    (16, 'pathogenic', 'C', 'G', 287),
    (17, 'pathogenic', 'C', 'T', 287),
    (18, 'pathogenic', 'C', 'A', 268),
    (19, 'pathogenic', 'G', 'A', 265),
    (20, 'pathogenic', 'A', 'G', 143),


##c.executemany('INSERT INTO Seqinfo VALUES(?,?,?,?,?);',Seqinfo);
##conn.commit()
```

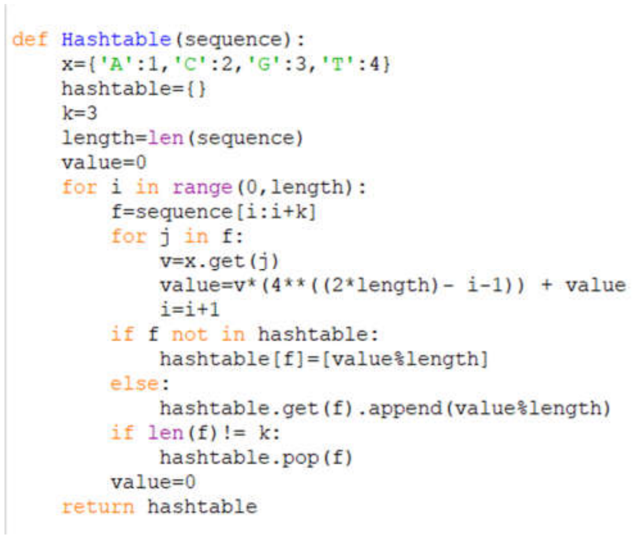
figure(1)

Figure(2) illustrates the algorithm flow in the coding part of the program running. After asking the user to enter the pass for the query file. The file is parsed to extract the sequence using a function called compile. The next step is to check the length of the sequence to make sure there are no insertions or deletions using a function called check and if the length did not match the insulin gene, the user will be asked to enter another file.


Read seq
Follow Hash Position Index Algorithm
Get position of possible SNP
Check position in INSULIN_SNPs database
If found, retrieve needed info from the database. If not, "Undefined"

figure(2)

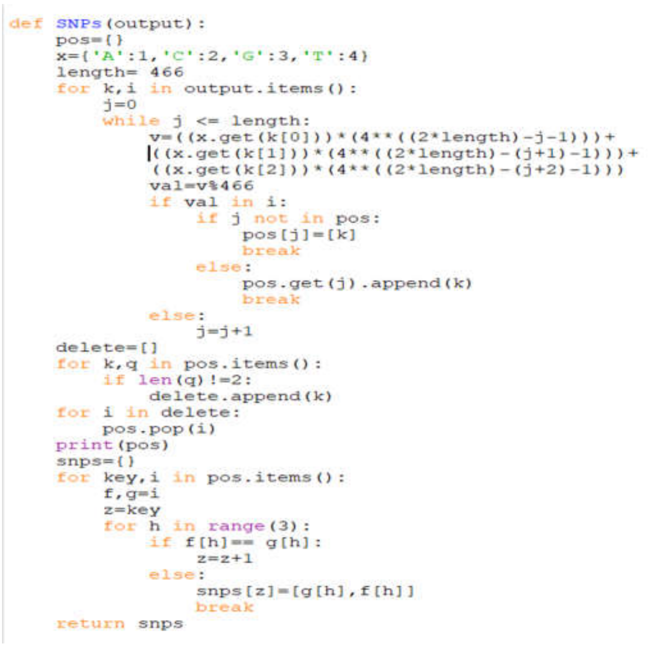
Then, it reaches the Hash Position Index Algorithm where the sequences both the original and the query will be divided into overlapping kmers of length =3 (k=3) then the four bases are converted into four binary digits, where A=1, C=2, G=3, T=4. The Hash value algorithm formula used for each kmer is the sum for the hash value of each character in the kmer following this function Hash(value)=value*[4^(2*k-m-1)], where value represents the corresponding value of the character, K represents the length of M, and k-mer represents the position range of the character in the string [0- (m-1)]. This is written as a function named Hashtable shown below in figure (3)



```
def Hashtable(sequence):
    x={'A':1,'C':2,'G':3,'T':4}
    hashtable={}
    k=3
    length=len(sequence)
    value=0
    for i in range(0,length):
        f=sequence[i:i+k]
        for j in f:
            v=x.get(j)
            value=v*(4**((2*length)- i-1)) + value
            i=i+1
        if f not in hashtable:
            hashtable[f]=[value%length]
        else:
            hashtable.get(f).append(value%length)
        if len(f)!= k:
            hashtable.pop(f)
        value=0
    return hashtable
```

figure(3)

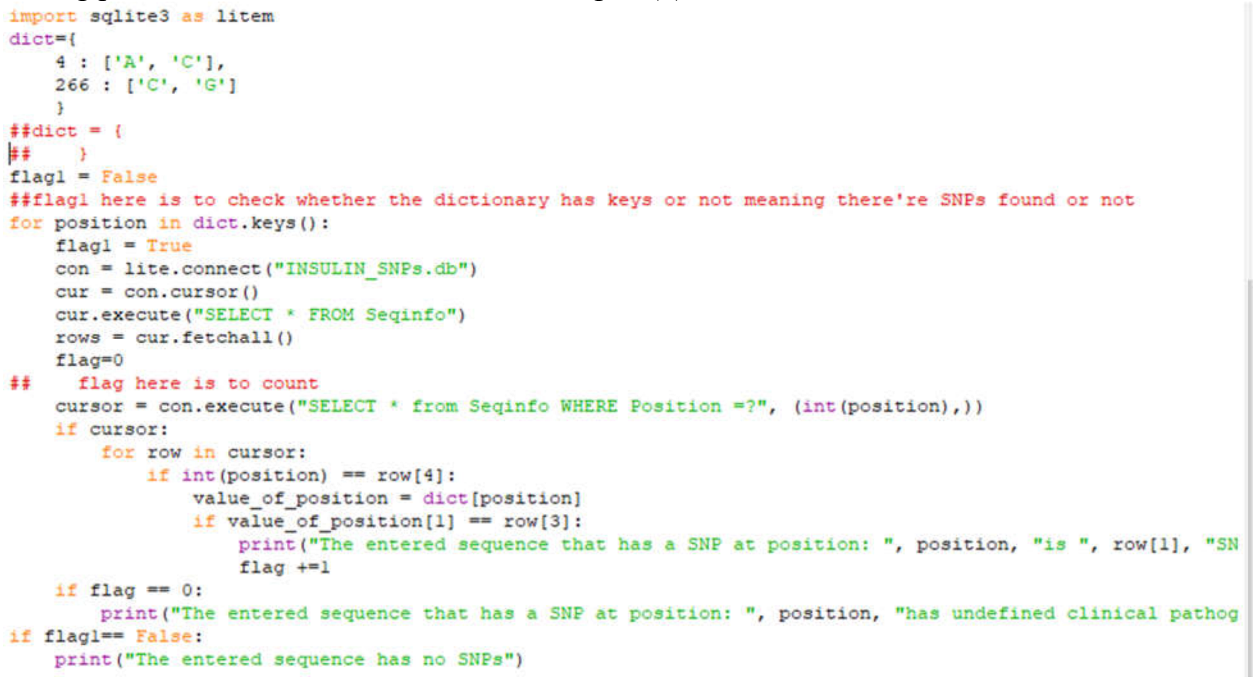
The 2 hashtables produced from Hashtable function will have the kmer as their key and a list for the hash values as their value. They will be compared using a function name diff to get the different values between them that indicate the position of SNPs. The output of diff will enter another function named SNPs to interpret the hash values and get the position for each SNP by reversing the hash function. The output of SNPs will be a dictionary with the key is the position of the SNP and the value is a list with the nucleotide in the original sequence and in the query. By following the code, the program will give a dictionary of all different SNPs found and the new base by comparing both entered and original sequences.



```
def SNPs(output):
    pos={}
    x={'A':1,'C':2,'G':3,'T':4}
    length= 466
    for k,i in output.items():
        j=0
        while j <= length:
            v=((x.get(k[0]))*(4**((2*length)-j-1)))+
            ((x.get(k[1]))*(4**((2*length)-(j+1)-1)))+
            ((x.get(k[2]))*(4**((2*length)-(j+2)-1)))
            val=v%466
            if val in i:
                if j not in pos:
                    pos[j]=[k]
                    break
                else:
                    pos.get(j).append(k)
                    break
            else:
                j=j+1
    delete=[]
    for k,q in pos.items():
        if len(q)!=2:
            delete.append(k)
    for i in delete:
        pos.pop(i)
    print(pos)
    snps={}
    for key,i in pos.items():
        f,g=i
        z=key
        for h in range(3):
            if f[h]== g[h]:
                z=z+1
            else:
                snps[z]=[g[h],f[h]]
                break
    return snps
```

figure(4)

After getting the dictionary, the next step was to fetch the database and check these SNPs positions and the new nucleotide base for each SNP if found or not. If both the SNP position and the new nucleotide were found in the database, the program will give an output that this position has a clinical pathogenicity X, for example, which is also retrieved from the database. The coding part for this function is as follows in figure(5).



```
import sqlite3 as litem
dict={
    4 : ['A', 'C'],
    266 : ['C', 'G']
    }
##dict = {
##    }
flag1 = False
##flag1 here is to check whether the dictionary has keys or not meaning there're SNPs found or not
for position in dict.keys():
    flag1 = True
    con = lite.connect("INSULIN_SNPs.db")
    cur = con.cursor()
    cur.execute("SELECT * FROM Seqinfo")
    rows = cur.fetchall()
    flag=0
##    flag here is to count
    cursor = con.execute("SELECT * from Seqinfo WHERE Position =?", (int(position),))
    if cursor:
        for row in cursor:
            if int(position) == row[4]:
                value_of_position = dict[position]
                if value_of_position[1] == row[3]:
                    print("The entered sequence that has a SNP at position: ", position, "is ", row[1], "SN
                    flag +=1
    if flag == 0:
        print("The entered sequence that has a SNP at position: ", position, "has undefined clinical pathog
if flag1== False:
    print("The entered sequence has no SNPs")
```

figure(5)

5. Evaluation\ Experiment\Results

Each part of the previous coding parts was checked several times with already known outputs to check the code validity and accuracy.

1. **Database creation evaluation:**

   Evaluating this part was by checking the created database and its contents by DB Browser for SQLite. This part was checked as Done efficiently as shown in figure(6) below. The evaluation was by checking table creation and writing all inputs correctly inside it and checking the data type of each column.

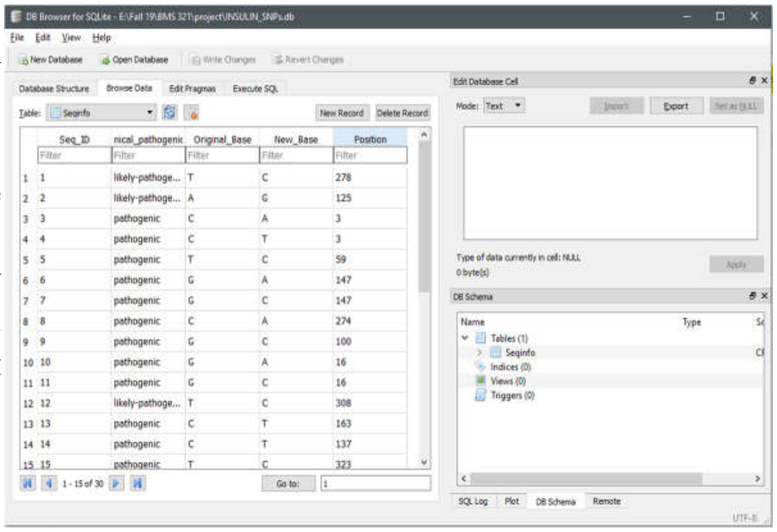


Table: Seqinfo

	Seq_ID	nical_pathogenic	Original_Base	New_Base	Position
1	1	likely-pathoge...	T	C	278
2	2	likely-pathoge...	A	G	125
3	3	pathogenic	C	A	3
4	4	pathogenic	C	T	3
5	5	pathogenic	T	C	59
6	6	pathogenic	G	A	147
7	7	pathogenic	G	C	147
8	8	pathogenic	C	A	274
9	9	pathogenic	G	C	100
10	10	pathogenic	G	A	16
11	11	pathogenic	G	C	16
12	12	likely-pathoge...	T	C	308
13	13	pathogenic	C	T	163
14	14	pathogenic	C	T	137
15	15	pathogenic	T	C	323

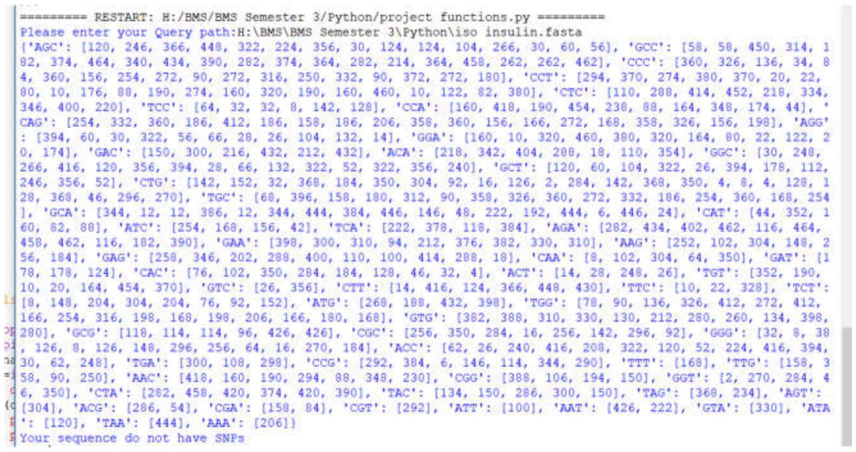
1 - 15 of 30

Type of data currently in cell: NULL
0 byte(s)

figure(6)

2. **Hash position index evaluation:**

   Evaluating this part was by entering the original sequence as the query and comparing the hash tables to ensure that there will be no SNPs detected and the hash function is excuted properly. The output is shown below in figure (7)

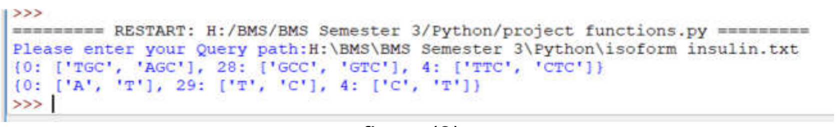


```
========= RESTART: H:/BMS/BMS Semester 3/Python/project functions.py =========
Please enter your Query path:H:\BMS\BMS Semester 3\Python\iso insulin.fasta
{'AGC': [120, 246, 366, 448, 322, 224, 356, 30, 124, 124, 104, 266, 30, 60, 56], 'GCC': [58, 58, 450, 314, 1
82, 374, 464, 340, 434, 390, 282, 374, 364, 282, 214, 364, 458, 262, 262, 462], 'CCC': [360, 326, 136, 34, 8
4, 360, 156, 254, 272, 90, 272, 316, 250, 332, 90, 372, 272, 180], 'CCT': [294, 370, 274, 380, 370, 20, 22,
80, 10, 176, 88, 190, 274, 160, 320, 190, 160, 460, 10, 122, 82, 380], 'CTC': [110, 288, 414, 452, 218, 334,
346, 400, 220], 'TCC': [64, 32, 32, 8, 142, 128], 'CCA': [160, 418, 190, 454, 238, 88, 164, 348, 174, 44], '
CAG': [254, 332, 360, 186, 412, 186, 158, 186, 206, 358, 360, 156, 166, 272, 168, 358, 326, 156, 198], 'AGG'
: [394, 60, 30, 322, 56, 66, 28, 26, 104, 132, 14], 'GGA': [160, 10, 320, 460, 380, 320, 164, 80, 22, 122, 2
0, 174], 'GAC': [150, 300, 216, 432, 212, 432], 'ACA': [218, 342, 404, 288, 18, 110, 354], 'GGC': [30, 248,
266, 416, 120, 356, 394, 28, 66, 132, 322, 52, 322, 356, 240], 'GCT': [120, 60, 104, 322, 26, 394, 178, 112,
246, 356, 52], 'CTG': [142, 152, 32, 368, 184, 350, 304, 92, 16, 126, 2, 284, 142, 368, 350, 4, 8, 4, 128, 1
28, 368, 46, 296, 270], 'TGC': [68, 396, 158, 180, 312, 90, 358, 326, 360, 272, 332, 186, 254, 360, 168, 254
], 'GCA': [344, 12, 12, 386, 12, 344, 444, 384, 446, 146, 48, 222, 192, 444, 6, 446, 24], 'CAT': [44, 352, 1
60, 82, 88], 'ATC': [254, 168, 156, 42], 'TCA': [222, 378, 118, 384], 'AGA': [282, 434, 402, 462, 116, 464,
458, 462, 116, 182, 390], 'GAA': [398, 300, 310, 94, 212, 376, 382, 330, 310], 'AAG': [252, 102, 304, 148, 2
56, 184], 'GAG': [258, 346, 202, 288, 400, 110, 100, 414, 288, 18], 'CAA': [8, 102, 304, 64, 350], 'GAT': [1
78, 178, 124], 'CAC': [76, 102, 350, 284, 184, 128, 46, 32, 4], 'ACT': [14, 28, 248, 26], 'TGT': [352, 190,
10, 20, 164, 454, 370], 'GTC': [26, 356], 'CTT': [14, 416, 124, 366, 448, 430], 'TTC': [10, 22, 328], 'TCT':
[8, 148, 204, 304, 204, 76, 92, 152], 'ATG': [268, 188, 432, 398], 'TGG': [78, 90, 136, 326, 412, 272, 412,
166, 254, 316, 198, 168, 198, 206, 166, 180, 168], 'GTG': [382, 388, 310, 330, 130, 212, 280, 260, 134, 398,
280], 'GCG': [118, 114, 114, 96, 426, 426], 'CGC': [256, 350, 284, 16, 256, 142, 296, 92], 'GGG': [32, 8, 38
, 126, 8, 126, 148, 296, 256, 64, 16, 270, 184], 'ACC': [62, 26, 240, 416, 208, 322, 120, 52, 224, 416, 394,
30, 62, 248], 'TGA': [300, 108, 298], 'CCG': [292, 384, 6, 146, 114, 344, 290], 'TTT': [168], 'TTG': [158, 3
58, 90, 250], 'AAC': [418, 160, 190, 294, 88, 348, 230], 'CGG': [388, 106, 194, 150], 'GGT': [2, 270, 284, 4
6, 350], 'CTA': [282, 458, 420, 374, 420, 390], 'TAC': [134, 150, 286, 300, 150], 'TAG': [368, 234], 'AGT':
[304], 'ACG': [286, 54], 'CGA': [158, 84], 'CGT': [292], 'ATT': [100], 'AAT': [426, 222], 'GTA': [330], 'ATA
': [120], 'TAA': [444], 'AAA': [206]}
Your sequence do not have SNPs
```

figure(7)

3. **Giving a dictionary as an output of SNPs and their positions:**

   Evaluating this part was by entering the query as a file with known SNPs and comparing the output of the SNPs and their position with the known positions of SNPs (0,4,29). The output is shown below in figure(8)

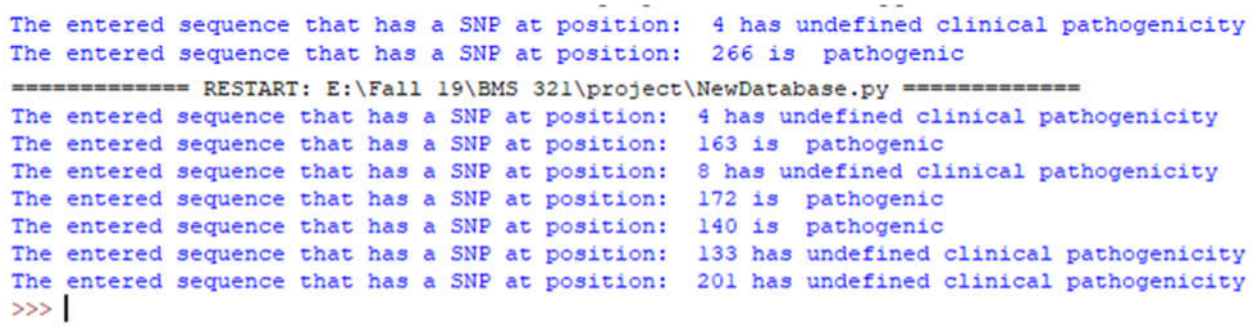


```
>>>
========= RESTART: H:/BMS/BMS Semester 3/Python/project functions.py =========
Please enter your Query path:H:\BMS\BMS Semester 3\Python\isoform insulin.txt
{0: ['TGC', 'AGC'], 28: ['GCC', 'GTC'], 4: ['TTC', 'CTC']}
{0: ['A', 'T'], 29: ['T', 'C'], 4: ['C', 'T']}
>>> |
```

figure(8)

4. **Checking dictionary output and giving the desired output:**

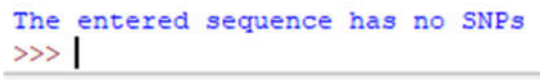
    That part was checked several times, 8, by having different values of already known positions in the database and positions that are not in the database. In addition to that, an empty dictionary case was checked and its output was designed to give a printing sentence stating that there are no SNPs found. Illustrated in figure(9), the different trials of the first case for having a position in the database and not having. The output of this case was correctly as needed.



```
The entered sequence that has a SNP at position:  4 has undefined clinical pathogenicity
The entered sequence that has a SNP at position:  266 is  pathogenic
============= RESTART: E:\Fall 19\BMS 321\project\NewDatabase.py =============
The entered sequence that has a SNP at position:  4 has undefined clinical pathogenicity
The entered sequence that has a SNP at position:  163 is  pathogenic
The entered sequence that has a SNP at position:  8 has undefined clinical pathogenicity
The entered sequence that has a SNP at position:  172 is  pathogenic
The entered sequence that has a SNP at position:  140 is  pathogenic
The entered sequence that has a SNP at position:  133 has undefined clinical pathogenicity
The entered sequence that has a SNP at position:  201 has undefined clinical pathogenicity
>>> |
```

figure(9)

Illustrated in figure(10) the output of the second case for having an empty dictionary.



```
The entered sequence has no SNPs
>>> |
```

figure(10)

6. Conclusion

To conclude, we can confirm that the proposed algorithm is working efficiently and can be released for users to use it easily. Based on the results and the evaluation section, the program can deal with any entered sequence with the same length of the insulin gene only, otherwise, it cannot proceed as the program mainly depends on aligning the entered sequence with the original one. So if the entered sequence had insertions or deletions, it wouldn't be recommended to use our program. The database has a limitation that it does not contain benign SNPs and only contains known pathogenic and likely pathogenic SNPs for the insulin gene. One more limitation for the program is that if the sequence has multiple SNPs it wouldn't give the user the overall clinical pathogenicity as this is not provided in the database and it's recommended to follow an experimental procedure for that.

7. References

[1] G. Reference, "What are single nucleotide polymorphisms (SNPs)?", Genetics Home Reference, 2019. [Online]. Available: https://ghr.nlm.nih.gov/primer/genomicresearch/snp?fbclid=IwAR0tJhUIP2QNuNm9bjo5Yq4zbO2ltlAVPtuK6NBkU8xf9EAeQNJK2VU5uCY. [Accessed: 15- Dec- 2019].

[2] "Diabetes", Who.int, 2016. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/diabetes. [Accessed: 15- Dec- 2019].

[3] Chen, G. & Xie, X.. (2017). A light weight SNP detection algorithm for the breast cancer targeted sequencing data. Biomedical Research (India). 28. 3574-3579.

[4] Li H, Ruan J, Durbin R. Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res 2008; 18: 1851-1858.

[5] Li R, Li Y, Fang X, Yang H, Wang J. SNP detection for massively parallel whole-genome resequencing. Genome Res 2009; 19: 1124-1132.

[6] Li R, Li Y, Kristiansen K, Wang J. SOAP: short oligonucleotide alignment program. Bioinformatics 2008; 24: 713-714.

[7] Goya R, Sun MG, Morin RD, Leung G, Ha G. SNVMix: predicting single nucleotide variants from next-generation sequencing of tumors. Bioinformatics 2010; 26: 730-736.

[8] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J. The Sequence alignment/map format and SAMtools. Bioinformatics 2009; 25: 2078-2079.

[9] J. Shang, F. Zhu, W. Vongsangnak, Y. Tang, W. Zhang and B. Shen, "Evaluation and Comparison of Multiple Aligners for Next-Generation Sequencing Data Analysis", BioMed Research International, vol. 2014, pp. 1-16, 2014. Available: 10.1155/2014/309650 [Accessed 16 December 2019].

[10] "dbSNP Home Page", *Ncbi.nlm.nih.gov.* [Online]. Available: http://www.ncbi.nlm.nih.gov/SNP/. [Accessed: 16- Dec- 2019].