HW 3

Mathematics 127
Mathematical and Computational Methods in Molecular Biology

Fall 2002
UC Berkeley, CA

Nasser M. Abbasi

Fall 2002 Compiled on August 2, 2022 at 8:09am [public]

Contents

1 Problems

Problem Set 3 (due Tuesday October 15)
MATH 127: Mathematical and Computational Methods in Molecular Biology

Please work on the starred problem alone.

Problem 1

Prove that a non-trivial connected graph is Eulerian if, and only if, every edge belongs to an odd number of cycles.

Problem 2^*

The goal of this problem is to obtain some familiarity with running RepeatMasker:

You'll need to use the web server at
`http://ftp.genome.washington.edu/cgi-bin/RepeatMasker`

a) Get the sequence with accession AF548051 from Genbank. How long is the particular A tail of the repeat? Find the article in the literature (on which this Genbank entry is based), read it, and report on the maximum length of observed Alu A tails in the human genome. What is the apparent function of the A tail?

b) RepeatMask the sequence at the RepeatMasker web server. How does the answer depend on the type of organism selected? What SW score do you get? What is its meaning?

c) How dependent is RepeatMasker on the A tail? Try removing the A tail. Does RepeatMasker still find the repeat? Try extending it considerably (beyond the observed length of A tail in the genome)? Does RepeatMasker still find the repeat?

Problem 3

Consider the overlap detection method described in class for DNA: Compute the convolution vector 4 times, each time setting one of the bases to 1 and the rest to 0. Suppose you try to find the overlap between two random DNA sequences of length 500, each of which contains equal amounts of the four bases. What do you expect the maximum value of an element in the convolution vector to be? What kind of bound does this place on the minimum size overlap you can detect reliably with the method?

Problem 4

The paper by Pevzner, Tang and Waterman states that "the spectral alignment problem can be efficiently solved by dynamic programming in the

case where the number of mutations is small". Make this statement precise and describe the algorithm for solving it.

Optional Problem

Implement the overlap detection method discussed in class.

Note: this is easy using the MATLAB **FFT** and **IFFT** commands (type *help FFT* or *help IFFT* to learn more about them). To view the convolution vector you can use the **PLOT** command.

a) Write a function that takes as input two sequences f, g of the same length and finds the convolution $f * g$.

b) Test your program on the sequences [00001010110000010] and [01010110000010000]. What is the best shift? (Remember, you will have to reverse one of the sequences).

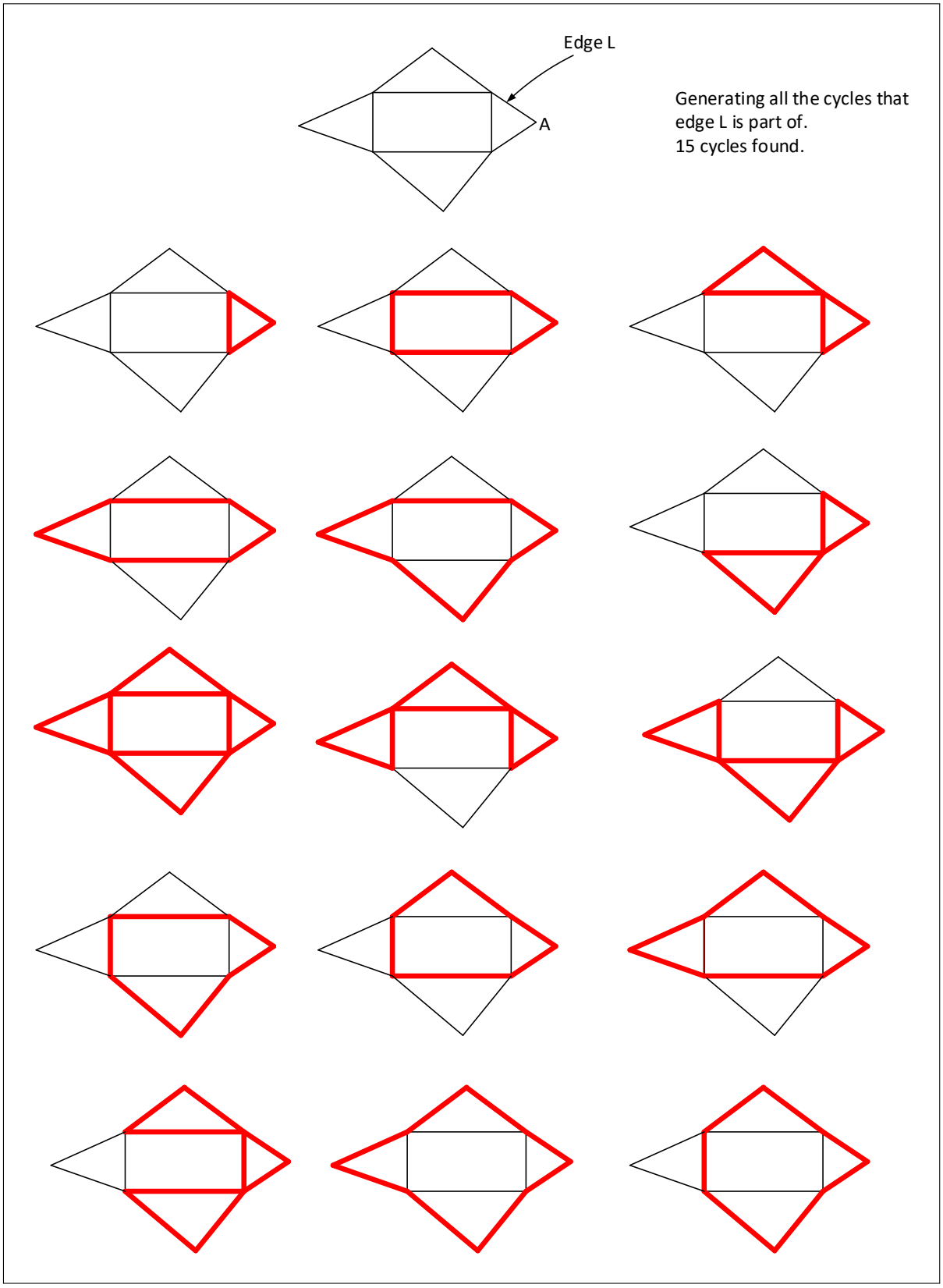


Figure 1: Example generating all cycles an edge is part of

2 Problem 1

Math 127, UC Berkeley.
HW 3
Problem 1
By Nasser Abbasi

Question
Prove that a non-trivial connected graphs is Eulerian IFF every edge belongs to an odd number of cycles.

Answer

An euler graphs has an even degree on all of its vertices.
A cycle in the above, is meant to be a path $v_1, v_2, \ldots, v_n$ *where* $v_1 = v_n$

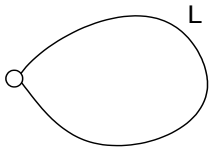
Since this is an IFF problem, There are two statements that needs to be proved.

Statement one: IF a graph is Euler, then each edge belongs to an odd number of edges.
Statement two: IF each edge belongs to odd number of cycles, then the graph is Euler.

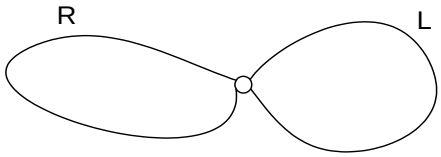
Proof for statement one

Proof by induction over the number of vertices v

For number of vertices = 1, the statement is true. Because given an Euler graph with one vertex, there are even number of edges, hence all edges must be of this form



Hence, each edge can only belong to 1 cycle. Since if we have more than one edge, as in



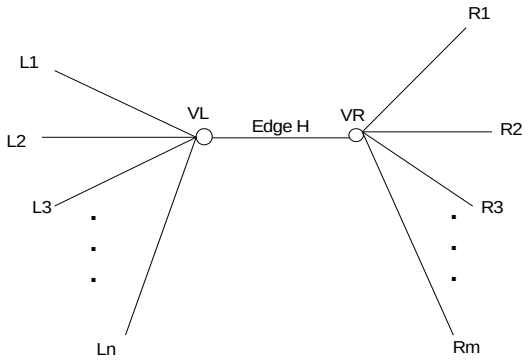
Then the cycle over L can not go over R, since this implies the vertex in the middle was visited twice. And the cycle over R can not go over L as well.

Hence the statement is true for all Euler graphs of one vertex.

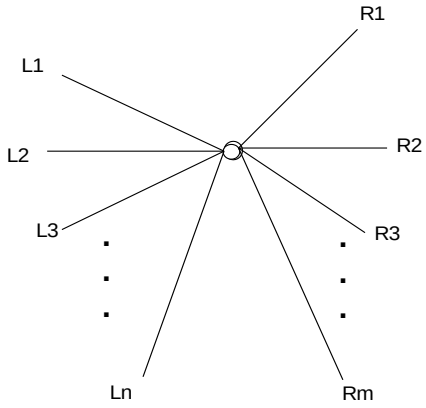
Now, Assume the statement is true for all connected Euler graphs with K vertices.

Now, need to show the statement is true for any Euler graph Z of K+1 vertices.

Looking at any one edge H in Z, such as (Figure 1 below)



Transform the graphs of K+1 vertices to K vertices, remove the edge H and collaps the two vertices to one vertex to get this (figure 2):



Since this is a K vertices Euler graph, then it will have all the edges in it part of odd number of cycles (by assumption). That is, number of cycles over each L edge is odd, and number of cycles of each R edge is odd.

Since the above also is a Euler graph, then the degree of the above one vertex is even. This means n+m is an even number. Hence n and m can be both even, or can be both odd numbers. (will show below that only case possible is for n,m to be both odd numbers).

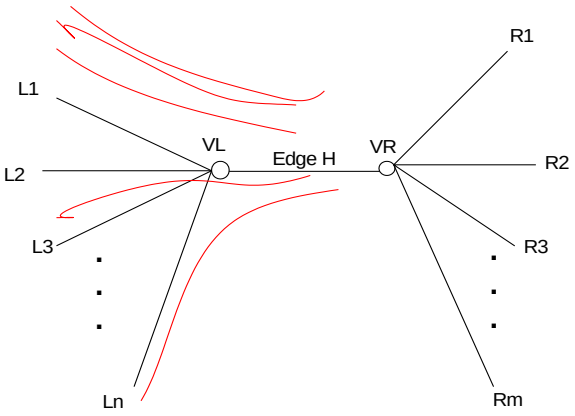
Now, looking back at figure 1 above, since this is a Euler graph, then the degree of VL is even, and the degree of VR is even.

But edge H adds one degree to each vertex, that is n+1 is even, and m+1 is even.

hence n and m must be both odd numbers.

Now, I need to show that an edge such as H in the Euler graph of K+1 vertices (see figure 1) can only be part of an odd cycles.

There are two cases here.
Case one, all the cycles passing over the edges L_i (or edges R_j) also pass over edge H.



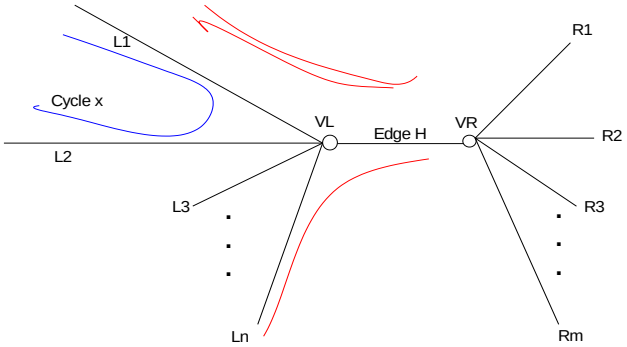
In this case, we have, since each edge in L has an odd number of cycles as shown above, the total number of cycles that edge H is part of is given by

$$q_1 + q_2 + \ldots + q_n = Z$$

where N is an odd number, and each q is an odd number.

When we add odd number of times odd numbers, we get an odd number. Hence Z is an odd number.
Hence number of cycles over H are odd.

Case two, in this case, not all cycles that pass over any or all edge L_i (or edges R_j) also pass over edge H as in the following figure example



Now, each time we get a cycle such as cycle x above, it will contribute 1 to the count of cycles to both edges it travels over (in this example edges L1 and L2).

This cycle also can not come back over another edge to cover H, since then it will visit vertex VL, and this is not possible by the definition of a cycle.

Hence for each such cycle as 'x' above, I have to subtract 2 from the total possible number of cycles passing over edge H.

So in this case, total number of cycles passing over edge H is

$$(q_1 + q_2 + \ldots + q_n) - (2^* \ \textit{number of cycles not going over edge } H)$$

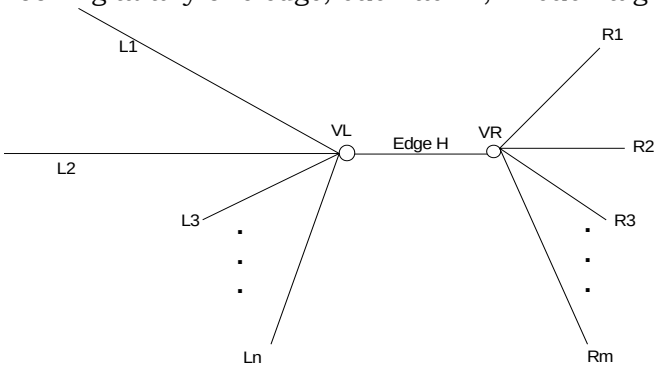
but since n is odd, (it is 1 subtracted from the degree of the vertex vL), and each q is odd, then the above is an odd number – even number. Which is an odd number.

Hence number of cycles over H are odd.

This completes the proof for statement one.

To prove statement two: *IF each edge belongs to odd number of cycles, then the graph is Euler.*

Looking at any one edge, such as H, in such a graph



I need to show, given that number of cycles over H are odd, then the degree of each vertex must be even (.i.e. the graph is Euler).

I use proof by contradiction.

In a proof by contradiction, when we need to proof the following statement is true

IF A => B

We assume that given the following is true:

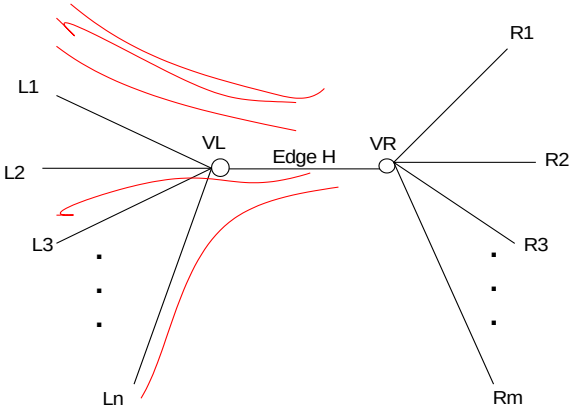
A and NOT B

and we try to show this is not possible. Hence A=>B.

So, in this example, A is 'each edge has odd number of cycles', B is 'Graph is Euler'.

So, Assume we have a graph with each edge has odd number of cycles, and it is NOT Euler.

Looking at the above figure, we have two cases.
Case one, all the cycles passing over edges L_i (or edges R_j) also pass over edge H.



Summing all cycles going over H, assume number of cycles over L_i *is* q_i

$q_1 + q_2 + \ldots + q_n = Z$

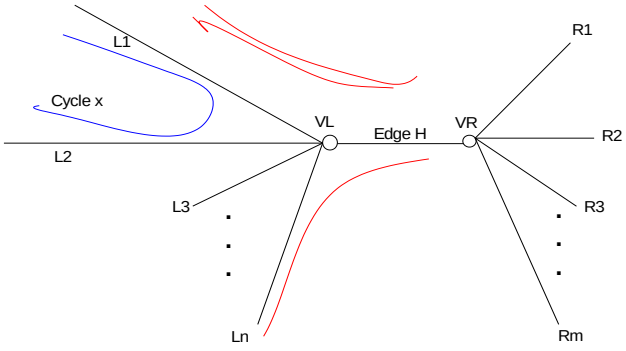
where Z is the cycles over H. But Z must be an odd number (given), and each q in the above sum is odd also (given).

The only way to add odd numbers N times and still get an odd number, is when N itself is odd.

Hence the number of L edges coming into VL in the above diagram is odd. Add the edge H itself, then the degree of vertex VL can only be even. Hence for this case, it is not possible to have odd number of cycles and have the vertex not have an even degree. Hence the graph must be Euler, which contradict the assumption.

Now to prove it for case two:

Case two: not all cycles that pass over the edges L_i (or edges R_j) also pass over edge H as in the example



Again, as argued earlier:

Now, each time we get a cycle such as cycle x above, it will contribute 1 to the count of cycles to both edges it travels over (in this example edges L1 and L2).

This cycle also can not come back over another edge (say L3) to cover H, since then it will visit vertex VL, and this is not possible by the definition of a cycle.

Hence for each such cycle as 'x' above, we subtract 2 from the total number of cycles passing over edge H. So total number of cycles Z going over H is

$$(q_1 + q_2 + \ldots + q_n) - (2 * \textit{cycles not going over } H) = Z$$

or

$$(q_1 + q_2 + \ldots + q_n) - \textit{even number} = Z$$

Since Z is odd, then $(q_1 + q_2 + \ldots + q_n)$ must be odd.
Hence n must be odd.

Hence the number of edges coming into VL in the above diagram is odd. Add the edge H itself, then the number degree of vertex VL can only be even. Hence for this case, it is not possible to have odd number of cycles and have the vertex not have an even degree. Hence the graph must be Euler, which contradict the assumption.

This completes the proof of the second statement.

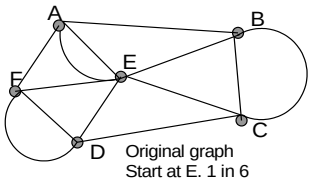
QED.

Problem 1 extra

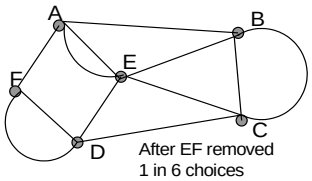
Extra on problem 1

Dr, as I was studying for this problem, I read about the Fleury algorithm to find all the cycles in an Euler graph. To learn how this algorithm works, I found 3 cycles in some graph, showing step by step how the algorithm works. I am including this below, (even thought it is not required to solve this problem, but it helped me understand the subject a little more).

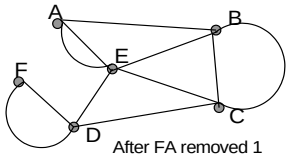
Only connected graphs with has no vertices of odd degree can have an Euler circuit.
So the above graphs must have at least one Euler circuit. To find, use Fleury algorithm:
1. Start at any vertex.
2. Pick an edge to travel. If one of the edges is a bridge (Going along it, there is no way to come back to the vertex other than on it, then do not select it, unless it is the one edge left from that vertex).
2. Remove the edge traveled. If a vertex left with no edges leaving it, remove the vertex.
3. repeat step 2 until no more edges left.
4. The order of the edges traveled is the Euler circuit.



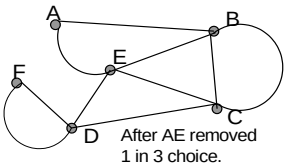
Original graph
Start at E. 1 in 6 choice.



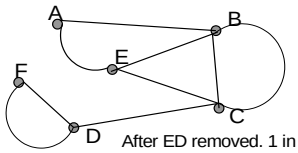
After EF removed
1 in 6 choices



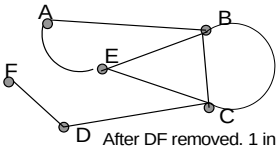
After FA removed 1
in 4 choices.



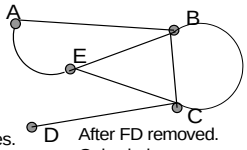
After AE removed
1 in 3 choice.



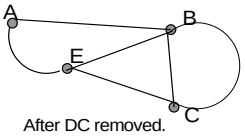
After ED removed. 1 in 3 choice. Notice now have option of going DC or DF.



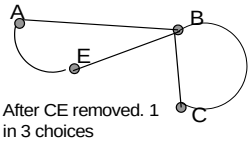
After DF removed. 1 in 2 choices. Notice, can not travel DC, since DC is a bridge.



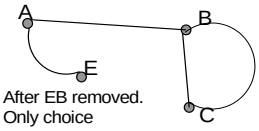
After FD removed.
Only choice.



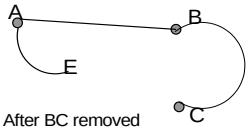
After DC removed.
Only choice



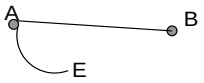
After CE removed. 1
in 3 choices



After EB removed.
Only choice



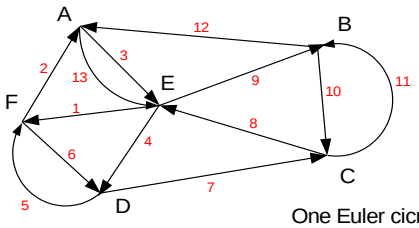
After BC removed
1 in 2 choices



After CB removed

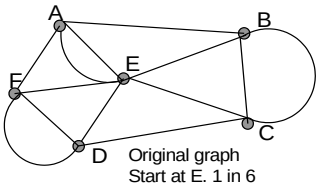
After BA removed

After AE removed. Back to starting vertex, and no edges left. Euler circuit.

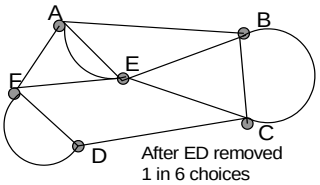


One Euler cicruit.

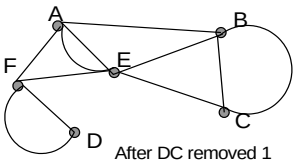
Showing Euler circuit. Visit each edge once, and visit all edges. Starting from E and back to E.



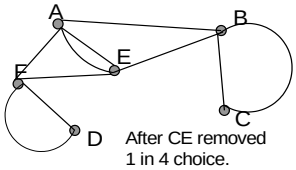
Original graph
Start at E. 1 in 6 choice.



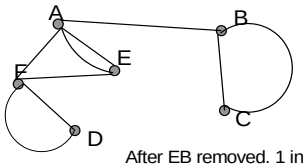
After ED removed
1 in 6 choices



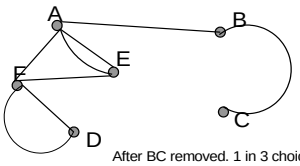
After DC removed 1
in 3 choices.



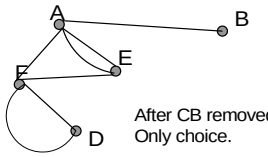
After CE removed
1 in 4 choice.



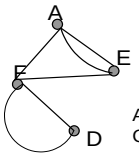
After EB removed. 1 in 4 choice.



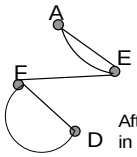
After BC removed. 1 in 3 choices.
Notice, can not remove BA, since BA is a bridge.



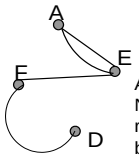
After CB removed.
Only choice.



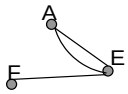
After BA removed.
Only choice.



After AF removed. 1
in 3 choices



After FD removed.
Notice, can not
remove FE as it is a
bridge

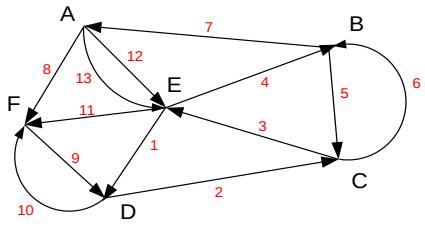


After DF removed
only choice.

After FE removed.
Only choice

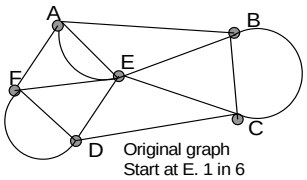
After EA removed

After AE removed. Back to starting vertex, and no edges left. Euler circuit.

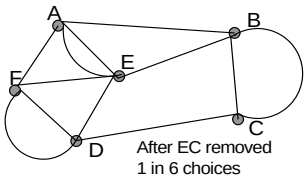


Showing Euler circuit. Visit each edge once, and visit all edges. Starting from E and back to E.

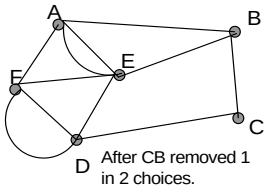
second Euler cicruit starting from E



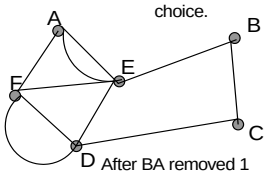
Original graph
Start at E. 1 in 6 choice.



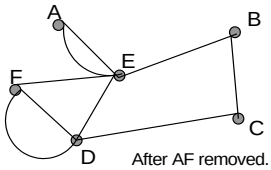
After EC removed
1 in 6 choices



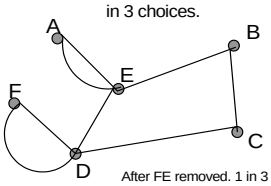
After CB removed 1
in 2 choices.



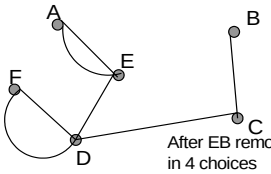
After BA removed 1
in 3 choices.



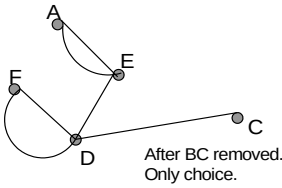
After AF removed. 1 in 3 choice.



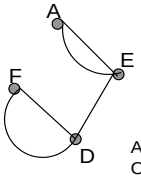
After FE removed. 1 in 3 choices.



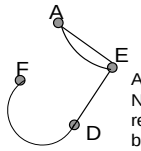
After EB removed. 1
in 4 choices



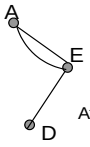
After BC removed.
Only choice.



After CD removed.
Only choice



After DF removed.
Notice, can not
remove DE as it is a
bridge

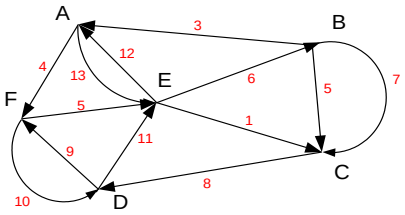


After FD removed.

After DE removed.
Only choice

After EA removed

After AE removed. Back to
starting vertex, and no
edges left. Euler circuit.



Showing Euler circuit. Visit each edge
once, and visit all edges. Starting
from E and back to E.

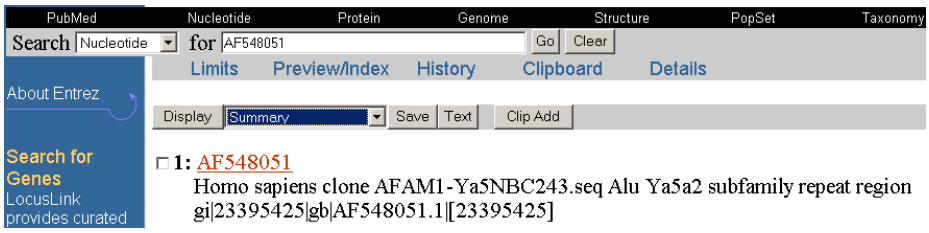
Third Euler cicruit starting from E

3 Problem 2

MATH 127, UC Berkeley
HW 3
Problem 2
Nasser Abbasi

Part A

From NCBI web site http://www.ncbi.nlm.nih.gov , typed AF548051 in the 'for' window, with 'search' window set to Nucleotide, and hit'GO'. The result is:



Then select the 'FASTA' option in the display menu, and click on the link, I get

```
>gi|23395425|gb|AF548051.1| Homo sapiens clone AFAM1-Ya5NBC243.seq Alu Ya5a2 subfamily repeat region
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATCACGAGGTC
AAGAGATCGAGACCATCCCGGCTAAAACGGTGAAACCCCGTCTCTACTAAAAATACAAAAAAATTAGCCG
GGCGTAGTGGCGGGCGCCTGTAGTCCCAGCTACTTGGGAGGCTGAGGCAGGAGAATGGCGTGAACCCGGG
AGGCGGAGCTTGCAGTGAGCCGAGATCCCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTC
TCAAAAAAAAAAAAAAAAAAAAAAAAAAGGAAA
```

The A tail of the above repeat is 'AAAAAAAAAAAAAAAAAAAAAAAGCAAA' which is **24 long**. (I stopped counting at just before 'GGAAA'. Since from the paper, it said to stop counting the A-tail when at least 2 consecutive non-adenosine bases show up.

From the article,

```
AUTHORS   Roy-Engel,A.M., Salem,A.H., Oyeniran,O.O., Deininger,L.,
          Hedges,D.J., Kilroy,G.E., Batzer,M.A. and Deininger,P.L.
  TITLE     Active alu element 'A-Tails': size does matter
  JOURNAL   Genome Res. 12 (9), 1333-1344 (2002)
```

From the article, it said the maxmimum A-length for Alu was 97 bases:

> Some of the A-lengths in this group reach as high as 97 bases for Alu and almost 180 bases for L1.

The apparent function of the A-Tail, is that the length of the Alu A-Tail correlate to which Alu is active, that is, which Alu elements are able to retropose. So the A-Tail is a factor in determining the retropositional capability of the Alu sequence as it said in the citation.

Part B

cut/paste the above, and went to http://ftp.genome.washington.edu/cgi-bin/RepeatMasker and submitted the above sequence using default setting. Tried for all organisms. The result is shown below

Organism	Number of matching repeats found in DB	SW score	Matching repeats	family	Matching position In query
Primates	1	2870	AluYa5	SINE	1..311
RODENTs	2	528 772	PB1 FAM	SINE SINE	2..132 137..296
Other mammals	2	625 805	FLAM_A FAM	SINE SINE	2..133 137..300
other vertebrates	0				
Arabidopsis	1	219	(A)n	Simple_repeat	283..311
Grasses	0				
Drosophila	1	219	(A)n	Simple_repeat	283..311

By doing the search assuming the query sequence belongs to different organism (that is, seach different repeats database), a different alignment score is obtained. The highest score was for Primates covering the whole query sequence. This means the primates have all of the query sequence (1..311) repeated in the genome in different locations. While for Rodents and other mammals, only parts of the sequence is repeated (2 parts, the first 'half' and the 'second' half). So, this repeat sequence is unique to Primates only.

In Arabidopsis and Drosophilla, only the tail-A 'AAAAAAAAAAAAAAAAAAAAAAAAGGAAA' was found to be a repeat.

Part C

Here, I removed the A-Tail. This is the subsequence AAAAAAAAAAAAAAAAAAAAAAAAGGAAA from the original sequence, and re-run all the tests as above. This is the new table.

Organism	Number of matching repeats found in DB	SW Score	Matching repeats	family	Matching position In query
Primates	1	2637	AluYa5	SINE	1..282
RODENTs	2	528 649	PB1 FAM	SINE SINE	2..132 137..282
Other mammals	2	625 660	FLAM_A FAM	SINE SINE	2..133 137..282
other vertebrates	0				
Arabidopsis	0				
Grasses	0				
Drosophila	0				

So this shows that repateMasker was still able to find the repeats in the data base without the A-Tail.

Now, I extend the A-Tail, by adding ‘A’s to the end of the tail. I added 120 ‘A’s to the A-tail, now the A-tail length is 149 (since the A-Tail original length was 29). I choose 120, since from the paper it said that the longest A-Tail was 97 in recent Alu insertions. So, this means an extension of 50 beyond the longest A-Tail

So, the new query sequence I used is this:

```
>gi|23395425|gb|AF548051.1| Homo sapiens clone AFAM1-Ya5NBC243.seq Alu Ya5a2 subfamily repeat region
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATCACGAGGTC
AAGAGATCGAGACCATCCCGGCTAAAACGGTGAAACCCCGTCTCTACTAAAAATACAAAAAAATTAGCCG
GGCGTAGTGGCGGGCGCCTGTAGTCCCAGCTACTTGGGAGGCTGAGGCAGGAGAATGGCGTGAACCCGGG
AGGCGGAGCTTGCAGTGAGCCGAGATCCCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTC
AAAAAAAAAAAAAAAAAAAAAAAAAGGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A
```

So, the new query sequence is now of length 311+120=431

I run the same tests as above, and the result is shown in this table

Organism	Number of matching repeats found in DB	SW Score	Matching repeats	family	Matching position In query
Primates	1	2880	AluYa5	SINE	1..431
RODENTs	2	528 773	PB1 FAM	SINE SINE	2..132 137..431
Other mammals	2	625 806	FLAM_A FAM	SINE SINE	2..133 137..431

other vertebrates	0				
Arabidopsis	1	1299	(A)n	Simple_repeat	283..431
Grasses	0				
Drosophila	1	1299	(A)n	Simple_repeat	283..431

The above shows extending the A-tail considerably did not affect repeateMasker ability to find the repeats.
Notice that in Arabidopsis and Drosophilla, only the tail-A is the repeat. The more 'A's I added, the higher the score became for those organisms. This shows there are long simple_repeats of 'A's in those organisms.

4 Problem 3

MATH 127, UC Berkeley.
HW3
Problem 3
Nasser Abbasi

Answer
To help me understand this problem, I worked a simple example of how to use convolution to find similarity between 2 DNA sequences.

Original DNA sequences we want to find the where max alignment between them occur

ACGTT
TCGAA

Replace one letter at by 1, and all the others by zero. Do this for A,C,G,T at a time. So we get 4 sets of sequences. Do the convolution between each 2 sequences at a time. So, we get 4 convolution vectors a the end. Next, sum the 4 convolution vectors.

	A=1	C=1	G=1	T=1
	10000	01000	00100	00011
	00011	01000	00100	10000
convolution	01100	10000	00100	01100

Now, sum the 4 convolution vectors

01100
10000
00100
01100
12300

So, the max value is at position 3 (it has the value of 3). So, looking back the two DNA sequences, this tells us where the max similarity is.

ACGTT
TCGAA

Nasser Abbasi
overlap_cobvolution.vsd
oct 12, 2002.

The convolution is a circular convolution. Done using these steps.

1. Align the sequences on top of each others, multiply corresponding elements (i.e. element at position j from top row with element at position j from second row, and add the final result.

multiply ↓

10000
00011
00000 = 0

→ *sum*

The above gives us the first element of the convolution vector, which is zero.

2. Now do a one position right shift of the lower sequence, and wrap around. So, a sequence 'abcde' when shifted one position to the right, will become 'eabcd'.

10000
10001
10000 = 1

The above gives us the first element of the convolution vector, which is 1.
again, right shift the lower sequence again from above, and multiply and sum, we get

10000
11000
10000 = 1

again, right shift the lower sequence again from above, and multiply and sum, we get

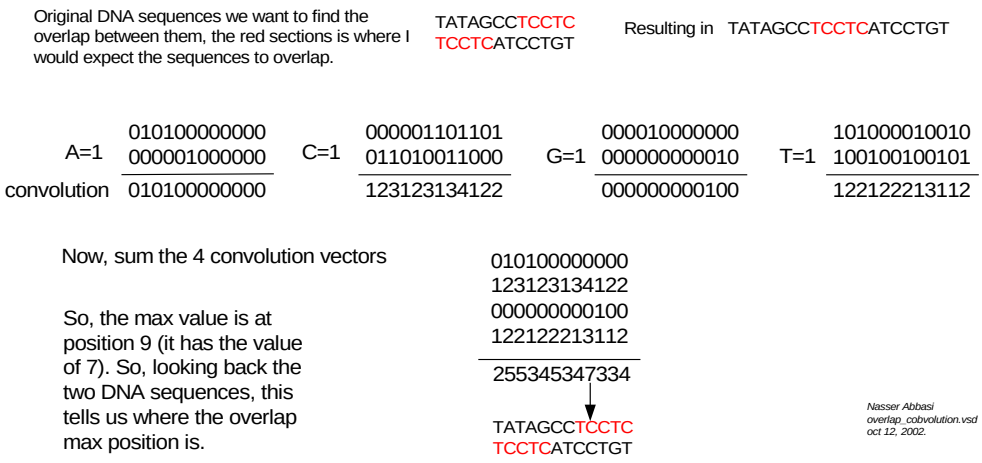
10000
01100
00000 = 0

again, right shift the lower sequence again from above, and multiply and sum, we get

10000
00110
00000 = 0

Since now the lower sequence has been shifted 4 times, and the size of the sequence is 5, we stop. The final convolution vector is then [01100].

To make sure the above makes sense and I did not make any mistakes, I worked a bigger example below to verify the method, to find the overlap between two larger DNA sequences.



The above shows the method does find the overlap region. (but it is not clear to me how does one go about determining the extent of the overlap if needed. May be the extent of the overlap is not needed, we just need a point to know where to fold the two sequences next to each other from) and this method gives us this.

Now, to answer the question. For the maximum value, it will occur when we have maximum overlap.

The max is when the two sequences are identical ofcourse. However, since these are random sequences, one would assume this is not likely to occur.

Since there are equal number of each base, there will be 125 of each type of base. These will be randomly distributed. So there is a chance of 1 out of 4 that an A from one sequence will be at the same position as an A in the other sequence, and the same for T,G and C.

Since the sequence length is 500, the chance of both sequences coming out the same is then (1/4)^500, which is almost impossible, but when this is the case, each convolution vector will a max value of 125, and the final convolution vector (the sum of the 4 convolution vectors will have a max value of **500.**

But for normal random distribution, the bases are uniformly distributed, and there is 25% chance that a base at one position in one sequence will be the same as the base in the same position in the second sequence. So, the convolution vector for say 'A' will have **a max value of 125/4 or about 32.25**. But we add 4 convolution vectors to get the final

convolution vector, and the chance that the vector for 'G' or 'C or 'T' will have its maximum value at the same column as 'A' is 1/500, so I can not just add 32.5 4 times to get 125. (I am assuming a fair random sequence generator, else I would have picked 125 as the maximum).

Instead, I think I should pick one maximum from one vector (say 'A') which is 32.5, and then assume the value at this column in the second, third and fourth row vectors from each of other 3 convolution vectors is the average value, which is 16 so, this gives a maximum of 32.5+16+16+16=**80.**

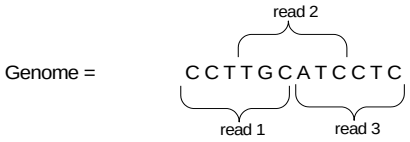
This 80 value gives a minimum size of 80 overlap that can be reliably detected (or about 20% of the size of the sequence).

5 Problem 4

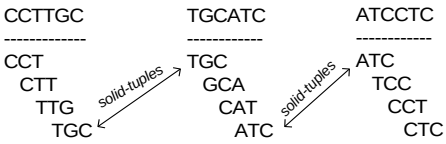
MATH 127
HW 3
Problem 4
Nasser Abbasi

The Euler assembly algorithm is a determination of superpath in de-Bruijn graph. A read is an edge in such a graph. However, these reads should have little errors in them. To remove these errors early on, the reads are broken into l-tuples and multiple sequence alignment is performed on these short strings and read errors detected and the original reads are then modified to remove these errors, and the new modified error-corrected reads (reduced errors) are then used in the de-Bruijn graph.

The spectral alignment problem is one solution to finding these read errors early on. To help me understand the concepts in the paper more, I made the following very simple example. Imagine a Genome G which is 'CCTTGCATCCTC', with 3 reads, each of length 6. let l, the length of the tuple be 3. This is what I get:



Assume we have 3 reads, each of length n=6. Assume L=3 (tuple size). Each read is broken into n-L+1, or 4 tuples as shown.



In the above, using M=1, there are those shown solid-tuples. Solid tuple are ones that belong to more than M reads. In the above simple example, there are 3 such solid tuples. The rest are called weak L-tuples. The approximated Genome (used by the error-correction method in the paper) will then be the set of all **solid**-tuples.

Now, a collection of L-tuples short strings is called a spectrum. A string 's' belongs this spectrum if each one of its L-tuples can be found in this spectrum. The spectral alignment problem is to find the minimum number of letter changes we have to make in the string 's' to cause all its tuples to be found in the given spectrum.

As an example, let string s='CCTG', with L=3, it then has 'CCT' and 'CTG' as its two tuples. (4-3+1=2).

Given a spectrum T={ 'ACT','TCT','CCA','CTG'}, we see that 's' as it stands does not belong to T, since one of 's' tuples (the first one) is missing from T.

To make 's' a T-string, we have to change one or more letters in 's', but when doing this change, we have to make sure we do not cause a removal of a tuple that was already found in T, else we just added a tuple in and removed one. In the above then, if we change the first letter in 's' from C to A, this will cause 's' now to be a T-string, since now the first tuple becomes 'ACT' instead of 'CCT' and this already in T. Hence this is the minimum change needed to make 's' be a T-string. A one letter change. **So, the reason why we need the number of mutations made to be small, so that we do not end up changing the original reads too much, causing the final approximated Genome to be much different from the actual one.** The approximated Genome (with error-correction) is the one used to construct the de-Bruijn graph.

The algorithm for spectral alignment works as follows.

Step 1. First construct the set of all solid L-tuples from all the initial reads from the sequencing project. (this is the set of all L-tuples that are found in more than M reads, where M is some threshold). This will be our initial spectrum, called T.

Step 2. Now, for each read which does not have all its L-tuples in T (i.e. it contains a number of weak L-tuples with respect to T) make the smallest letter changes (mutations) to cause one weak L-tuple to become a solid L-tuple (i.e. to show up in T). For example, in the above, I made one letter change in the weak L-tuple 'CCT' to cause to become a solid L-tuple.

Step 3. Now, generate the T spectrum again.

Repeat steps 2,3. Each time using the newly modified reads until no more reads with weak L-tuples is found (or stop at some threshold on number of mutations to make?).

This completes the algorithm, and generates new modified reads and an approximate Genome based on those error-corrected reads.

6 Problem 5

Math 127
HW3
Problem 5 (Optional)
Nasser Abbasi

Part (A)

I wrote a function that takes two DNA sequences, and returns the convolution vector.
The position of the maximum element in the convolution vector is where the overlap occurs.

This below are few examples showing how to use this function. (I highlight by red where the overlap actually is).

Example 1

```
>> clear all
>>%  1 2 3 4 5 67 8 9 0 1 23 4 5 6 7 8
>> x='ACGTATTACCCCGGGCCC';
>> y='CCCCGGGCCCTAGTATTT';
>> nma_hw3_problem5(x,y)
ans =

   4   5   4   2   3   2   6  10  15   8   6   6   2   3   6   4   4   3

>> [v,I]=max(ans);
>> I

I =

   9
```

So, this says the overlap occurred at position 9, which is correct by looking at the above sequences.

Example 2

```
>>%    1 2 3 4 5 6 78 9 0 1 2 3 4 5 6 7 8 9 0
>> x='ACTGTTGATATATATATATATA';
>> y='ATATATCGTCGTCGTCGTCG';
>> nma_hw3_problem5(x,y)

   4    4    3    5    4    6    2   10    3    7    4    9    1   10    3    7    5    7  2    6

>> [v,I]=max(ans);
>> I

I =

    8
```

Notice that there are another maximum location (of value 10) in the convolution vector, which is at position 14. This is the ‘end’ location. The max() function in matlab finds the first max in a vector. For overlap, I need the last one, the one near the edge.

Part B

To test the program against the given sequences, I use directly the circular convolution function I wrote (which is called by the function used in part A).

This is the output

```
>> help nma_cconv

 function nma_cconv(A,B) implement direct circular convolution for A,B
 vectors. Must be of equall length (for now).
 returns V, the convolution vector

>>%   1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
>> x=[0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0];
>> y=[0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0];

>> nma_cconv(x,y)

ans =

   1    2    1    5    1    2    1    1    1    2    1    1    1    1    2    1    1

>>
```

So, from the above, the maximum convolution is at position **4** and this gives the best shift.

```
function V=nma_hw3_problem5(x,y)
%function p=nma_hw3_problem5(x,y)
%
% This function takes 2 DNA sequences, and returns
% the convolution vector for the sequneces
%
% calls the function nma_cconv.m to do the convolution

p=0;

if(nargin ~=2)
    error('x,y expected');
end

N=length(x);
if(N ~= length(y))
    error('x,y must be equal length');
end

DNA='ATCG';
V=zeros(N,1);  % store final convolution vector here.

for(i=1:length(DNA))
    V=V+nma_cconv(normalize(x,(DNA(i))), ...
        normalize(y,(DNA(i))));
end

V=V';

%%%%%%%%%%%%%%%%%%%%%%%%%
% this function replaces each letter in S with 0
% except those that matches w, they are replaced with 1
%%%%%%%%%%%%%%%%%%%%%%%%%
function T=normalize(S,w)

S=upper(S);
T=zeros(length(S),1);

for(i=1:length(S))
    if(S(i)==w)
        T(i)=1;
    end
end
```

```
function v=nma_cconv(A,B)
%function nma_cconv(A,B) implement direct circular convolution for A,B
%vectors. Must be of equall length (for now).
%returns V, the convolution vector

%By Nasser Abbasi, Oct 12,2002.
%Written for HW 3, problem 5.
%Math 127, UC Berkeley.

N=length(A);

if(N ~= length(B))
    error('A,B must be of equal length');
end

Bi=1:N;
v=zeros(N,1);

for(i=0:N-1)
    v(i+1)=sum(A .* B(Bi));
    Bi=[N-i:N 1:N-(i+1) ];   %right sided circular shift
end

v=v';
```

6.1 Problem 5 source code

Matlab code

```
function V=nma_hw3_problem5(x,y)
%function p=nma_hw3_problem5(x,y)
%
% This function takes 2 DNA sequences, and returns
% the convolution vector for the sequneces
%
% calls the function nma_cconv.m to do the convolution

p=0;

if(nargin ~=2)
    error('x,y expected');
end

N=length(x);
if(N ~= length(y))
    error('x,y must be equal length');
end

DNA='ATCG';
V=zeros(N,1);  % store final convolution vector here.
V=V';

for(i=1:length(DNA))
    V=V+nma_cconv(normalize(x,(DNA(i))), ...
        normalize(y,(DNA(i))));
end

%%%%%%%%%%%%%%%%%%%%%%%%
% this function replaces each letter in S with 0
% except those that matches w, they are replaced with 1
%%%%%%%%%%%%%%%%%%%%%%%%
function T=normalize(S,w)

S=upper(S);
T=zeros(length(S),1);

for(i=1:length(S))
    if(S(i)==w)
        T(i)=1;
    end
end
```

```
function [y,zf]=nma_filter_v1(b,a,x,zi)
% function y=nma_filter_v1(b,a,x,zi)
%
% implement IIR filter to give the same output as matlab own
% filter.m function, and then convert the code to C++.
%
% The filter is a "Direct Form II Transposed"
% implementation of the standard difference equation:
%
%    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
%                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
%

% Nasser Abbasi, sept 27,2002.
% done for the 5Prime project work.

b=b(:);
a=a(:);

if(size(b,2) ~= 1)
   error('b must be a vector');
end

if(size(a,2) ~= 1)
   error('a must be a vector');
end

nb = length(b);
na = length(a);

if( nb ~= na)
   error('length(a) must equal length(b) in this implementation');
end

if(a(1) ~= 1.0)
   a=a/a(1);
   b=b/a(1);
end

nChannels=size(x,2);
nScan=size(x,1);
y=zeros(nScan,nChannels);
zf=zeros(nb-1,nChannels);

if(nargin==4)
   if(~isempty(zi))
      nCol=size(zi,2);
```

```
      if(nCol>nChannels)
         error('Zi number of columns can not be larger than number of X columns');
      end
      for(k=1:nCol)
         zf(:,k)=zi(:,k);
      end
   end
end

for(k=1:nChannels)
   for(n=1:nScan)
      nDelay=1;
      zf(:,k)=Z_(nDelay,x(:,k),y(:,k),zf(:,k),a,b,n-1);
      y(n,k)=b(1)*x(n,k)+zf(nDelay,k);
   end
   k
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% find the delay z at level at time t
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function zf=Z_(nDelay,x,y,zf,a,b,n)

nb=length(b);

if(n==0)
   return;
else
   if(nDelay==nb-1)
      zf(nDelay)=b(nDelay+1)*x(n)-a(nDelay+1)*y(n);
   else
      zf=Z_(nDelay+1,x,y,zf,a,b,n-1);
      zf(nDelay)=b(nDelay+1)*x(n)+zf(nDelay+1)-a(nDelay+1)*y(n);
   end
end
```

```
function c=nma_dconv(x,y)
%function c=nma_dconv(x,y)
% finds convolution using direct method.

if(length(x) ~= length(y))
   error('x and y must be same length');
end

n=length(x);
```

```
c=zeros(n,1);
yi=[1 n:-1:2];
yy=y(yi);

for(shift=0:n-1)

   t=0;
   for(i=1:n)
      t=t+x(i)*yy(i);
   end
   c(shift+1)=t;

   yi=[ mod((yi+1),n) ];
   I=find(yi==0);
   yi(I)=n;
   yy=y(yi);

end
```

```
function c = nma_cconv2(a,b,ctr)

if (exist('ctr') ~= 1)
  ctr = 0;
end

if (( size(a,1) >= size(b,1) ) & ( size(a,2) >= size(b,2) ))
    large = a; small = b;
elseif  (( size(a,1) <= size(b,1) ) & ( size(a,2) <= size(b,2) ))
    large = b; small = a;
else
  error('one arg must be larger than the other in both dimensions!');
end

ly = size(large,1);
lx = size(large,2);
sy = size(small,1);
sx = size(small,2);

%% These values are the index of the small mtx that falls on the
%% border pixel of the large matrix when computing the first
%% convolution response sample:
sy2 = floor((sy+ctr+1)/2);
sx2 = floor((sx+ctr+1)/2);

% pad:
```

```
clarge = [ ...
    large(ly-sy+sy2+1:ly,lx-sx+sx2+1:lx), large(ly-sy+sy2+1:ly,:), ...
        large(ly-sy+sy2+1:ly,1:sx2-1); ...
    large(:,lx-sx+sx2+1:lx), large, large(:,1:sx2-1); ...
    large(1:sy2-1,lx-sx+sx2+1:lx), ...
        large(1:sy2-1,:), ...
        large(1:sy2-1,1:sx2-1) ];

c = conv2(clarge,small,'valid');
```

```
function v=nma_cconv(A,B)
%function nma_cconv(A,B) implement direct circular convolution for A,B
%vectors. Must be of equall length (for now).
%returns V, the convolution vector

%By Nasser Abbasi, Oct 12,2002.
%Written for HW 3, problem 5.
%Math 127, UC Berkeley.

N=length(A);

if(N ~= length(B))
    error('A,B must be of equal length');
end

Bi=1:N;
v=zeros(N,1);

for(i=0:N-1)
    v(i+1)=sum(A .* B(Bi));
    Bi=[N-i:N 1:N-(i+1) ];   %right sided circular shift
end

v=v';
```

Maple code

```
# by Nasser Abbasi, oct 16, 2002.
# how to read clipped data file using MAPLE.

# INPIT: filename, the clipped full path file name.
# OUTPUT: a Matrix that contains the clipped data. it has
#         as many rows as there are in the clipped file.

epg := proc(filename)
   local line,y,f,x,k;
```

```
   try
      f := fopen(filename,READ);
   catch:
      printf("Failed to open file %s\n",filename);
      return;
   end try;

   line := readline(f);
   line := readline(f);

   line := readline(f);
   x := Matrix();
   k:=1;
   while( line <> 0 ) do
      y := sscanf(line,cat("%a" $ length(line)));
      y := y[3..nops(y)];
      y := convert(y,Vector[row]);
      if(k>1) then
         x := < x , y >;
      else
         x:= y;
      end if;
      k:= k+1;
      printf("read line %d\n",k);
      line := readline(f);
   end do;

   fclose(f);
   return x;

end proc;
```

```
hw3:=module()
  export solve,nma_cconv;
  local normalize_it;

###############
#
#
###############
   solve := proc(x::string,y::string)

   local N,DNA,V,i,t1,c;

   N:= length(x);
```

```
   if(N <> length(y)) then
      error("Length of sequences must match");
   end if;

   DNA:="ATCG";
   V:=Vector(N);

   for i from 1 to length(DNA) do
       c:=DNA[i];
       t1 := normalize_it(x,c);
       print(t1);
       #V:=V+nma_cconv(normalize_it(x,DNA[i]),normalize(y,DNA[i]));
   end do

   end proc;

######################
#
######################
   #nma_cconv:=proc(x::string,y::string)

   #end proc();

###############
#
#
###############
   normalize_it := proc(s::string,c)

     local S,i,T;

     S:=StringTools[UpperCase](s);
     T:=Vector(length(S));

     print("s = "); print(s);
     print("S="); print(S);
     print("length S is"); print(length(S));
     print("c="); print(c);

     for i from 1 to length(S) do
         print("check on S[i] , c"); print(S[i]); print(c);

         if(S[i]=c) then
            print("set T[i] to 1");  print("i=",i);
            T[i]:=1;
            print("T now is "); print(T);
         else
```

```
            print("set T[i] to 0");
            T[i]:=0;
         end if;
     end do;

     print("T=",T);
     return T;

   end proc;
end module;
```