Week 2 Discussion Section Genome 540

Chengxiang Qiu

Agenda

- Q’s about HW1?
- C++ Tips
- Overview of HW2

Brief Explanation

	Genome 1: TACCGA	Genome 2: CGACTA	Genome 2 RC: TAGTCG
1	A	A	AGTCG
2	ACCGA	ACTA	CG
1	CCGA	CGACTA	G
3	CGA	CTA	GTCG
2	GA	GACTA	TAGTCG
2	TACCGA	TA	TCG

For each suffix in the 'forward' strand of genome 1, find the length of the longest matching subsequence in genome 2 (or its reverse complement) -> Report a histogram of these lengths

Histogram

Genome 1: TACCGA

1	A
2	ACCGA
1	CCGA
3	CGA
2	GA
2	TACCGA

Match Length Histogram:

1 2

2 3

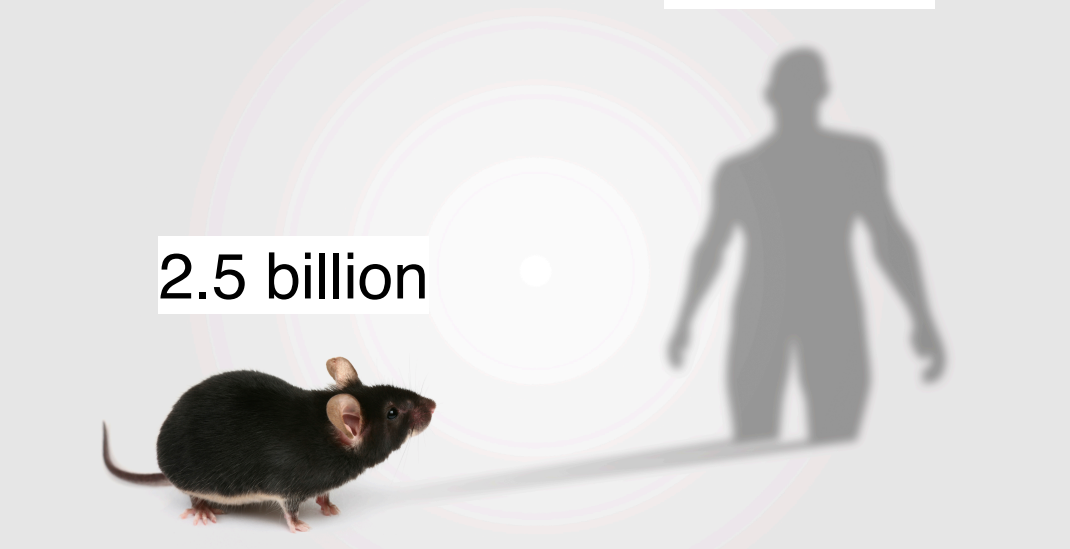
3 1

For each suffix in the 'forward' strand of genome 1, find the length of the longest matching subsequence in genome 2 (or its reverse complement) -> Report a histogram of these lengths

Extra credit question - how many ultraconserved elements (UCEs) in the region?

UCEs; >=200 bp exact match between mouse and human

6.4 billion



Only 481 UCEs

For a selected 10 Mb region, how many UCEs?

Any Questions on HW1?

More C++ Tips

Pointers in C++

```
string food = "Pizza";
string* ptr = &food;

cout << food << "\n";  // Output "Pizza"

cout << &food << "\n"; // Output the memory address (0x7ff7b4173648)

cout << ptr << "\n";   // ??
```

0x7ff7b4173648

```
cout << *ptr << "\n";  // ??
```

Pizza

Pointers in C++

a[3] is the same as *(a+3) or a+3?

&a[3] is the same as *(a+3) or a+3?

Again, * is to get value, and & is to get memory address

Difference between pointer to an array and array of pointers

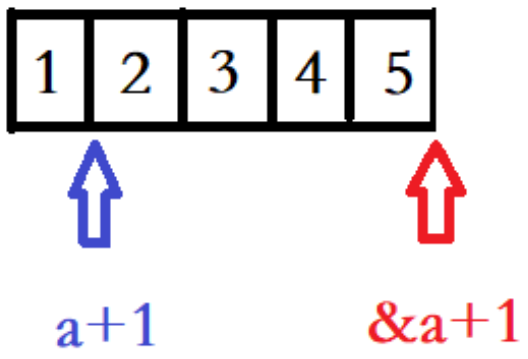
- Pointer to an array - we are using the pointer to access the components of the array

```
// pointer to an array of five numbers;
int (*ptr)[5];
```

- Array of pointers is an array of the pointer variables

```
// declaration of an array of pointers;
int *ptr[5];
```

Practice 1




```
int a[3] = {1, 2, 3};

cout << a << "\n";

cout << &a << "\n";

cout << *a << "\n";

cout << a+1 << "\n";

cout << &a+1 << "\n";

cout << *(a+1) << "\n";

cout << *(*(&a + 1)-1) << "\n";
```

The address of the first element

The address of a

The value of the first element (1)

The address of a[1]

The whole length of the address

The value of the second element (2)

The value of the last element (3)

Practice 2

```
int a[5] = {1, 2, 3, 4, 5};

int* ptr = (int*)(&a + 1);

printf("%d,%d", *(a + 1), *(ptr - 1));
```

2 5

Some feedbacks on HW-1

- If you are looking for the longest match between Sequence 1 and Sequence 2

Suffix array (after sorting)

….. i, i + 1, ….

If suffix i and suffix i+1 are from different Sequences, we count their longest match;
Otherwise, ignoring them.
Finally, reporting the longest match.

Some feedbacks on HW-1

- If you are looking for the longest match from Sequence 2, for every suffix from Sequence 1

Suffix array (after sorting)

$\longrightarrow$

….. i, i + 1, ….

For any suffix from Sequence 1, for example, i,
If i + 1 is also from Sequence 1, what will you do?

….. i, i + 1, i + 2, i + 3, ….

….. i - 3, i - 2, i - 1, i, ….

Again, the sum of counts in your histogram should be the same as the length of Sequence 1

Question: in your histogram, what's the count for the longest match = 1?

Any questions on HW-2 ?

Program: Generate FASTA files of simulated genome using order-0 Markov and order-1 Markov models

- read in a file in FASTA format
- determine the frequencies of the nucleotides and dinucleotides (based on the forward strand) and the length of the sequence
- produce two simulated sequences based on the length and nucleotide or dinucleotide frequency of the original sequence
  - Original Sequence: CGACTA
  - Simulated Sequence: AGATGC
- output two files in FASTA format containing the simulated sequence

order-0 Markov

```
Fasta 1: CP003913.fna
>gi|440453185|gb|CP003913.1| Mycoplasma pneumoniae M129-B7, complete genome
*=816373
A=249201
C=162924
G=163697
T=240551
N=0

Nucleotide Frequencies:
A=0.3053
C=0.1996
G=0.2005
T=0.2947
```

order-1 Markov

```
Dinucleotide Count Matrix:
A=98512 50763 47914 52012
C=53047 36681 26746 46450
G=40870 37148 36764 48915
T=56772 38332 52273 93173
```

	A	C	G	T
A	98512	50763	47914	52012
C	53047	36681	26746	46450
G	40870	37148	36764	48915
T	56772	38332	52273	93173

CGACTA

```
Dinucleotide Frequency Matrix:
A=0.1207 0.0622 0.0587 0.0637
C=0.0650 0.0449 0.0328 0.0569
G=0.0501 0.0455 0.0450 0.0599
T=0.0695 0.0470 0.0640 0.1141

Conditional Frequency Matrix:
A=0.3953 0.2037 0.1923 0.2087
C=0.3256 0.2251 0.1642 0.2851
G=0.2497 0.2269 0.2246 0.2988
T=0.2360 0.1594 0.2173 0.3873
```

= 1

= 1
= 1
= 1
= 1

Random number generator in C++

```cpp
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
    cout << "RAND_MAX:" << RAND_MAX << endl;
    for (int i = 0; i < 5; i++){
        cout << rand() << endl;
    }
}
```

rand() will return a random number between 0 and RNAD_MAX

```
RAND_MAX:2147483647
16807
282475249
1622650073
984943658
1144108930
```

```
RAND_MAX:2147483647
16807
282475249
1622650073
984943658
1144108930
```

```
RAND_MAX:2147483647
16807
282475249
1622650073
984943658
1144108930
```

Pseudo random number

Random number generator in C++

```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;

int main()
{
    cout << "RAND_MAX:" << RAND_MAX << endl;
    srand((unsigned)time(NULL));
    for (int i = 0; i < 5; i++){
        cout << rand() << endl;
    }
}
```

```
RAND_MAX:2147483647
857283596
897610249
62834768
1649475099
861589770
```

```
RAND_MAX:2147483647
857300403
1180085498
1685484841
486935110
2005698700
```

```
RAND_MAX:2147483647
858241595
1966313913
212092108
1956688783
1651289370
```

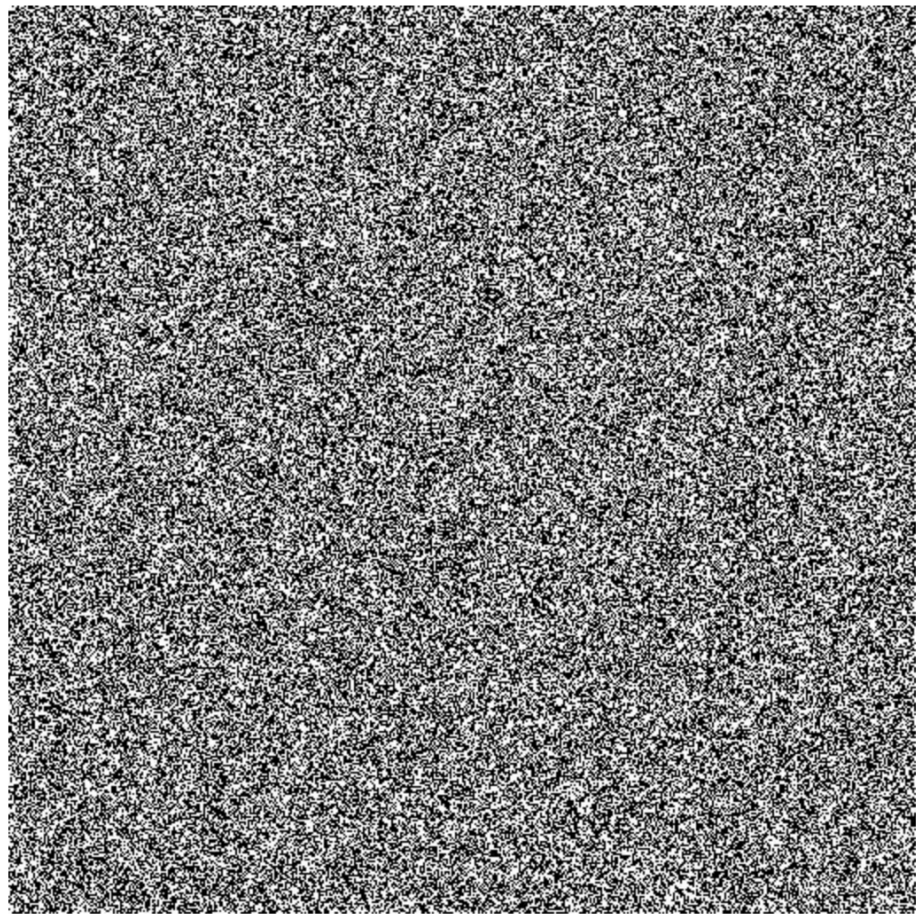
```cpp
cout << rand()/double(RAND_MAX) << endl;
```

[0, 1]

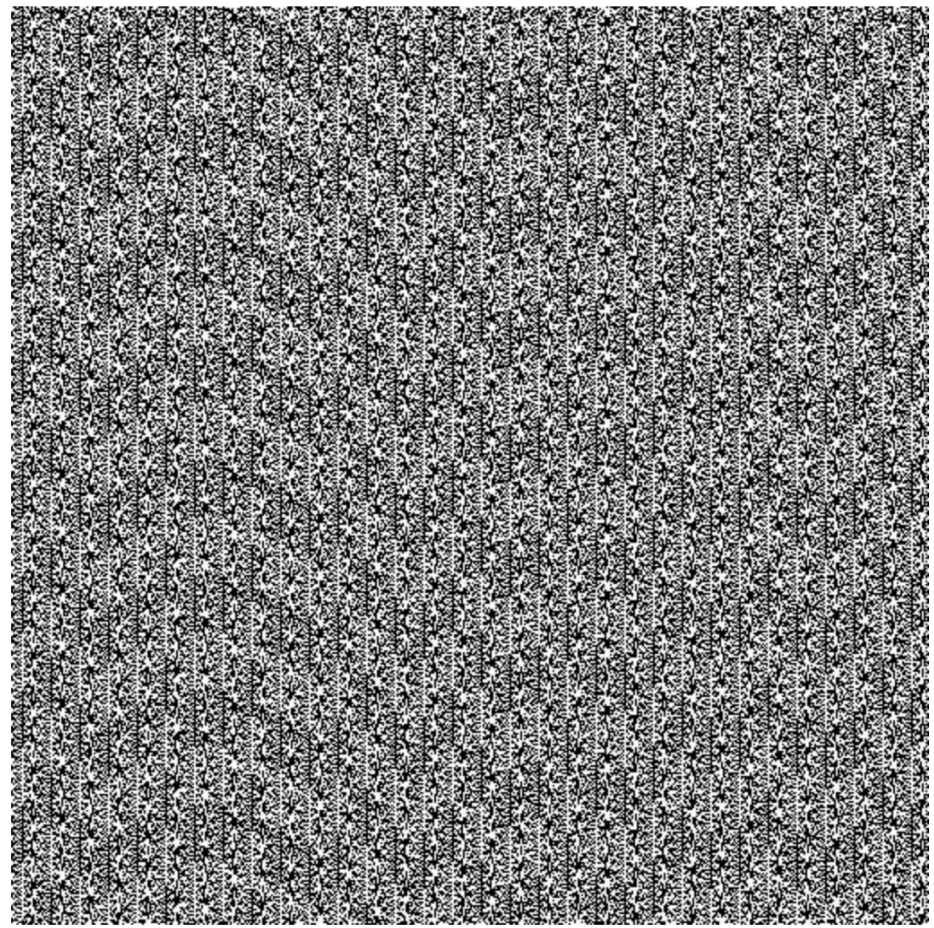
```cpp
cout << (rand() % (b-a+1))+a << endl;
```

[a, b]

C# System.Random



php rand()



Using random() instead of rand()

```
#include <random>
#include <iostream>

int main()
{
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<double> dist(1.0, 10.0);

    for (int i=0; i<16; ++i)
        std::cout << dist(mt) << "\n";
}
```

rand() is typically a low quality pRNG. <random> provides a variety of engines with different characteristics suitable for many different use cases.

% generally biases the data and floating point division still produces non-uniform distributions. <random> distributions are higher quality as well as more readable.

srand() only permits a limited range of seeds. Engines in <random> can be initialized using seed sequences which permit the maximum possible seed data.

The 2nd part

- Run your HW1 program twice
  - Human sequence & Simulated mouse sequence using order-0 Markov model
  - Human sequence & Simulated mouse sequence using order-1 Markov model

  - what can you conclude about the statistical significance of matches between the orthologous mouse and human regions in homework 1?

- Due Jan 23, 11:59pm

See you next week!

- Reminder: HW1 due THIS Sunday, 11:59pm
- Please have your name in the filename of your homework assignment