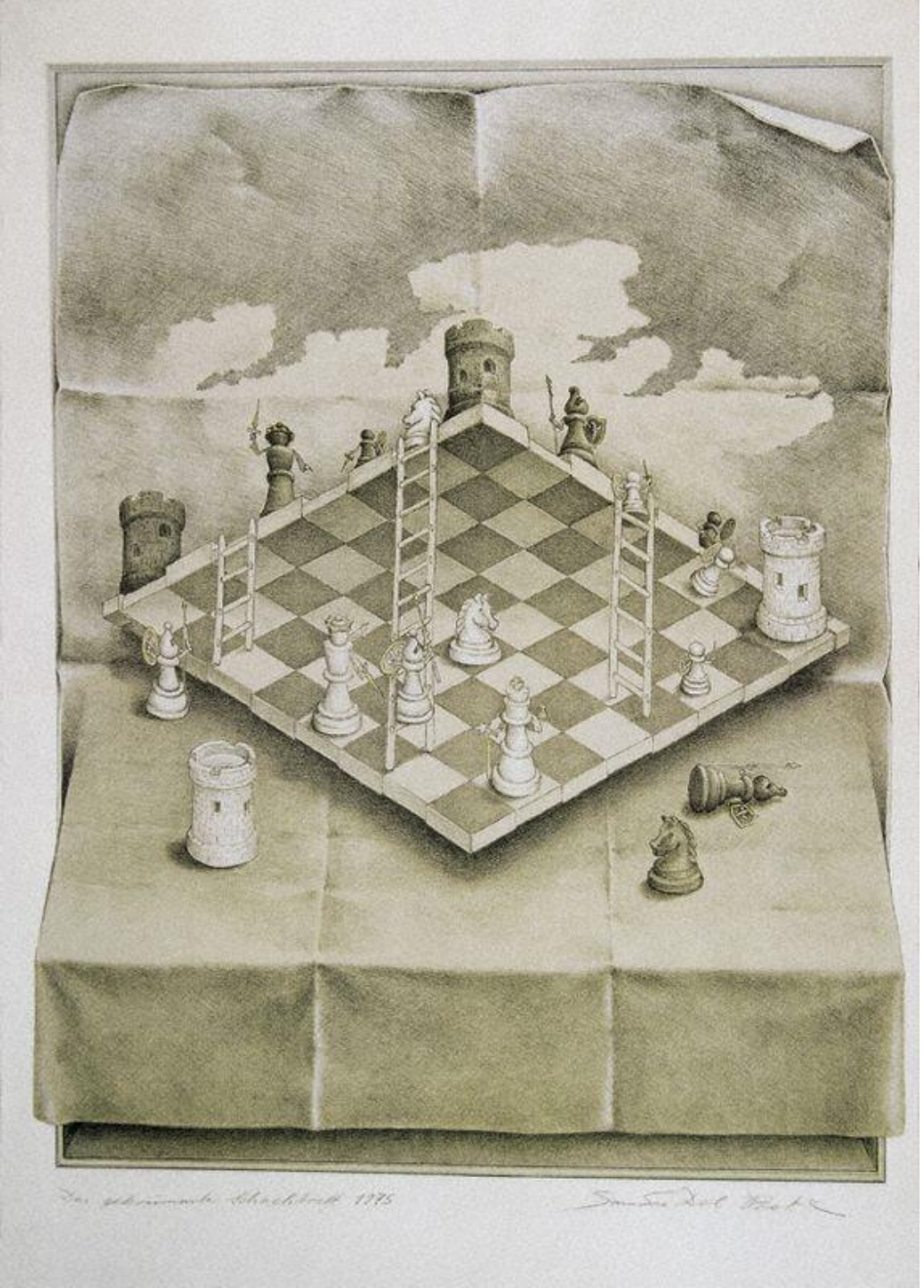

Data Structure Tips And Tricks

Darin Ohashi

- Senior Architect at Maplesoft

Working Example

Working Example

- Read and analyze words from a text file

Working Example

- Read and analyze words from a text file
- Don't know anything about the input
  - Don’t know which words will appear
  - Don't know how many words

Working Example

- Read and analyze words from a text file
- Don't know anything about the input
  - Don’t know which words will appear
  - Don't know how many words
- Unknowns mean we need dynamic data structures
  - Array
    - 1D, index starting at 1
  - Table

Read all the words

```
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:

end:
```

Read all the words

- Store the words in a dynamic Array.
- Create an empty Array by calling the Array constructor with an empty list

```
A := Array( [] ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:

end:
```

Read all the words

- Store the words in a dynamic Array
- Create an empty Array by calling the Array constructor with an empty list
- Use ArrayTools:-Append to add elements to the Array

```
A := Array( [] ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   ArrayTools:-Append( A, word ):
end:
```

Read all the words

How many words	numelems(A);

Read all the words

How many words	numelems(A);
First word	A[1];

Read all the words

How many words	numelems(A);
First word	A[1];
First 10 words	A[1..10];

Read all the words

How many words	numelems(A);
First word	A[1];
First 10 words	A[1..10];
Last word	A[-1];

Read all the words

How many words	numelems(A);
First word	A[1];
First 10 words	A[1..10];
Last word	A[-1];
Last 10 words	A[-10..-1];

Read all the words

How many words	numelems(A);
First word	A[1];
First 10 words	A[1..10];
Last word	A[-1];
Last 10 words	A[-10..-1];

Was *word* in the document?	found := false;
	for w in A
	do
	  if w = word then
	  found := true;
	  break;
	  end;
	end;

Read all the words

How many words	numelems(A);
First word	A[1];
First 10 words	A[1..10];
Last word	A[-1];
Last 10 words	A[-10..-1];

Was *word* in the document?
 	
• Use member(*word*, A)!
	found := false;
	for w in A
	do
	  if w = *word* then
	  found := true;
	  break;
	  end;
	end;

Array Operations

- Arrays work with basic maple functions
  - map, seq, add, sort, etc
- ArrayTools
  - Extend, Insert, Remove
  - Higher level algorithms, Partition, Reverse, etc

Array Performance

- Performance of Array operations
  - Append/Extend and Remove from the end of the Array take, on average, time proportional to the number of elements added or removed. Often this is going to be constant time.
  - Insert/Remove from the middle of an Array requires moving all the elements after the position of the change, so these will be proportional to the number of elements in the Array.
- For the best performance add and remove elements from the back of the Array.
- As Arrays can be modified, many function support *inplace* operations
  - If you don't need the original data, working inplace saves memory.
  - map[inplace](f, A) (aka transform)
  - sort[inplace](A)
- One I recently discovered:
  - map[reduce=NULL](print, A) (aka foreach)

Unique Words

- Use a table
  - In other languages: dictionary, map
- Stores key/value pairs
- Constant time operations
  - Insert
  - Delete
  - Has

```
T := table( ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:

end:
```

Unique Words

- Use a table
  - In other languages: dictionary, map
- Stores key/value pairs
- Constant time operations
  - Insert
  - Remove
  - Has
- Don't need to use a value
  - Dynamic set

```
T:= table( ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   T[ word ] := NULL:
end:
```

Unique Words

How many unique words	numelems(T);

Unique Words

How many unique words	numelems(T);
List unique words	[indices(T)];

Unique Words

How many unique words	numelems(T);
List unique words	[indices(T)];
Was *word* in the document?	assigned(T[word]);

Unique Words

How many unique words	numelems(T);
List unique words	[indices(T)];
Was *word* in the document?	assigned(T[word]);
Remove *word* from T	unassign('T[word]');

Word frequency

- Use table again, but now utilize the value

```
T := table( ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:




end:
```

Word frequency

- Use table again, but now utilize the value
- Use assigned to see if a word has been encountered before

```
T := table( ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   if assigned( T[word] ) then

   else

   end:
end:
```

Word frequency

- Use table again, but now utilize the value
- Use assigned to see if a word has been encountered before
- Modify the stored value.

```
T := table( ):
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   if assigned( T[word] ) then
      T[word] := T[word] + 1:
   else
      T[word] := 1:
   end:
end:
```

Word frequency

How many times did *word* appear?	T[word];

Word frequency

How many times did *word* appear?	T[word];

Most frequent word
 	
• for-in on a table works, but eval(T,1) is required because tables are subject to last name evaluation
• Doesn't require creating a list of all keys/values
	M := 0:
	W := NULL:
	for key, value in **eval(T,1)**
	do
	  if value > M then
	  M := value:
	  W := key;
	  end:
	end:

Word frequency

How many times did *word* appear?	T[word];

Most frequent word
 	
• for-in on a table works, but eval(T,1) is required because tables are subject to last name evaluation
• Doesn't require creating a list of all keys/values
	M := 0
	W := NULL:
	for key, value in eval(T,1)
	do
	  if value > M then
	  M := value:
	  W := key;
	  end:
	end:

List words and frequencies (as equations)	[indices(T, pairs)]:

Word frequency

How many times did *word* appear?	T[word];

Most frequent word
 	
• for-in on a table works, but eval(T,1) is required because tables are subject to last name evaluation
• Doesn't require creating a list of all keys/values
	M := 0
	W := NULL:
	for key, value in eval(T,1)
	do
	  if value > M then
	  M := value:
	  W := key;
	  end:
	end:

List words and frequencies (as equations)	[indices(T, pairs)]:
Sort by frequency	sort([indices(T, pairs)], key = rhs);

Word frequency

How many times did *word* appear?	T[word];

Most frequent word
 	
• for-in on a table works, but eval(T,1) is required because tables are subject to last name evaluation
• Doesn't require creating a list of all keys/values
	M := 0;
	W := NULL;
	for key, value in eval(T,1)
	do
	  if value > M then
	  M := value:
	  W := key;
	  end:
	end:

List words and frequencies (as equations)	[indices(T, pairs)]:
Sort by frequency	sort([indices(T, pairs)], key = rhs);
Sort by frequency, space efficient	sort[inplace](indices(T, pairs, output=Array), (x,y)->(rhs(x)<rhs(y)));

Collect More Data

- Collect the index for each instance of a word along with the count

```
T := table( ):
for i
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   if assigned( T[word] ) then


   else


   end:
end:
```

Collect More Data

- Collect the index for each instance of a word along with the count
- Use a Record to store named data

```
T := table( ):
for i
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   if assigned( T[word] ) then



   else
      T[word] := Record( "count"=  ,
                         "index"=                ):
   end:
end:
```

Collect More Data

- Collect the index for each instance of a word along with the count
- Use a Record to store named data
- Use an Array in the Record to store the indices.

```
T := table( ):
for i
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   if assigned( T[word] ) then



   else
      T[word] := Record( "count" = 1,
                         "index" = Array( [ i ] ) ):
   end:
end:
```

Collect More Data

- Collect the index for each instance of a word along with the count
- Use a Record to store named data
- Use an Array in the Record to store the indices.
- Append new values for each instance of the word

```
T := table( ):
for i
do
   word := FileTools:-Text:-ReadString( "input_file" ):
   if word = NULL then
      break
   end:
   if assigned( T[word] ) then
      T[word]:-count := T[word]:-count+1;
      ArrayTools:-Append( T[word]:-index, i ):
   else
      T[word] := Record( "count" = 1,
                         "index" = Array( [ i ] )  ):
   end:
end:
```

Collect More Data

How many times did *word* appear?	T[word]:-count;

Collect More Data

How many times did *word* appear?	T[word]:-count;
What is the first appearance of *word*?	T[word]:-index[1]

Collect More Data

How many times did *word* appear?	T[word]:-count
What is the first appearance of *word*?	T[word]:-index[1]
First word	

Collect More Data

How many times did *word* appear?	T[word]:-count
What is the first appearance of *word*?	T[word]:-index[1]

First word	for key, value in eval(T,1)
	do
	  if member(1, value:-index) then
	  break;
	  end:
	end:

Collect More Data

How many times did *word* appear?	T[word]:-count
What is the first appearance of *word*?	T[word]:-index[1]

First word
• Remember when using an Array, we just needed A[1]
	for key, value in eval(T,1)
	do
	  if member(1, value:-index) then
	  break;
	  end:
	end:

How To Pick The Right Data Structure?

How To Pick The Right Data Structure?

- How much data are you working with?
  - If you know n is small, O(n) can still be fast
  - Constant time operations on Arrays have small constants
  - Constant time operations on Tables have large constants

How To Pick The Right Data Structure?

- How much data are you working with?
  - If you know n is small, O(n) can still be fast
  - Constant time operations on Arrays have small constants
  - Constant time operations on Tables have large constants
- Consider your access patterns
  - What do you need to know about the data?
  - How are you going access the data?

How To Pick The Right Data Structure?

- How much data are you working with?
  - If you know n is small, O(n) can still be fast
  - Constant time operations on Arrays have small constants
  - Constant time operations on Tables have large constants
- Consider your access patterns
  - What do you need to know about the data?
  - How are you going access the data?
- Do you care about the order of the words?
  - Array
- Do you care about accessing a specific word?
  - Table

How To Pick The Right Data Structure?

- How much data are you working with?
  - If you know n is small, O(n) can still be fast
  - Constant time operations on Arrays have small constants
  - Constant time operations on Tables have large constants
- Consider your access patterns
  - What do you need to know about the data?
  - How are you going access the data?
- Do you care about the order of the words?
  - Array
- Do you care about accessing a specific word?
  - Table
- Unless memory is a concern you can always do both

Questions?

Up Next

- **Upper Year Courses - Vector Calculus**
  - **Dave Linder**