Recitation Week 3: Data Structures: Arrays
Data Structures

• Store and organize data on computers
• Facilitate data processing
• Data structure is designed according to data and data processing characteristics
  – Remark: If one knows a problem very well, then one can find a problem specific solution, which is very often better than a general solution. So choosing the right data structures for different tasks is important.
Regular data structures: *arrays*

- Identical data elements tightly packed
  - For example: an array of integer numbers 1, 2, 3, ... N

- Direct access through indexing
  - Index must be within array bounds
  - For many programming language, the index often starts from 0. So one has to use index in the range 0 to N-1, where N is the length of the array.

- Structure is implicit
  - Neighbors are found through indexing
  - No need to waste storage space for structure
1-D array

• 1-D array
  – Example 1: text in a sentence is an array of characters. “The weather is getting cold.”

  The weather is getting cold.

  – Example 2: Fibonacci sequence:
  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 | ...

Question: How do you know the number after 377?
Answer: use the formula \( f[n] = f[n-1] + f[n-2] \), where \( f \) is the array
2-D array

• 2-D array
  – Rows and columns, rows of rows, array of arrays, ...
  – Matrix

• Example 1:
  – How do you indicate the positions on the earth?
    • Two dimensions: latitude and longitude
    • Row: corresponds to same latitude
    • Column: corresponds to same longitude
  – Associate each position a value, e.g. temperature or the altitude, then you will get a 2-D array
Example 1

- Positions on the earth
Example 1

- The 2-D array of temperatures

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>76</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>76</td>
<td>76</td>
<td>75</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>76</td>
<td>75</td>
<td>76</td>
<td>75</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>75</td>
<td>76</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>76</td>
<td>76</td>
<td>75</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>77</td>
<td>75</td>
<td>76</td>
<td>76</td>
<td>77</td>
<td>77</td>
<td>78</td>
<td>75</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>77</td>
<td>76</td>
<td>78</td>
<td>75</td>
<td>76</td>
<td>75</td>
<td>76</td>
<td>79</td>
<td>76</td>
<td>79</td>
<td>78</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>78</td>
<td>79</td>
<td>82</td>
<td>83</td>
<td>82</td>
<td>80</td>
<td>82</td>
<td>80</td>
<td>80</td>
<td>82</td>
<td>80</td>
<td>82</td>
<td>80</td>
<td>79</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
<td>81</td>
<td>80</td>
<td>82</td>
<td>81</td>
<td>82</td>
<td>82</td>
<td>83</td>
<td>82</td>
<td>81</td>
<td>82</td>
<td>82</td>
<td>81</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>82</td>
<td>82</td>
<td>82</td>
<td>82</td>
<td>83</td>
<td>82</td>
<td>83</td>
<td>82</td>
<td>83</td>
<td>83</td>
<td>83</td>
<td>83</td>
</tr>
</tbody>
</table>
2-D array

• Indexing: to read the value from the 2-D array, one should use $A[m][n]$
  – Question: what are the ranges of $m$ and $n$?
  – Answer:
    • $m$: 0 - # of rows
    • $n$: 0 - # of columns

• Example 2: Matrix transposition

$$A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix} \quad B = A^T = \begin{bmatrix}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9
\end{bmatrix}$$

– So we have $B[m][n]=A[n][m]$
3-D array

• 3-D array
  – Stack of 2-D arrays, array of matrices, array of arrays of arrays...
  – Tensor

• Example 1:
  – Consider the previous example: temperatures on the earth
  – If you get the temperatures on day 1, day 2, day 3,..., you will get a series of “temperature matrices”
  – Put them together, you will get a 3-D array.
    • Question: how do you get the value of the temperature on day 3 at position (3,4)?
    • Answer: T[2][3][4]. Note: the 1st index denotes the date, and it starts from 0 representing day 1!
3-D array

Example 2: consider the image example in the lecture:

If you use a video camera to shoot, you will get a series of images (24 frames per seconds perhaps), so the video clip is a 3-D array, if you use another dimension to denote the time (or the # of frame)
Higher dimensional arrays

- 1-D array
  - A row
- 2-D array
  - Rows and columns, rows of rows
- 3-D array
  - Stack of 2-D arrays
- 4-D arrays? And even 5-D, 6-D, ...?
4-D arrays

- 4-D arrays
  - Rows of cuboids
  - D[1] means second cuboid
  - D[1][0] bottom 2-D array in second cuboid
  - D[3][3][1][2] means element in cuboid 4, slice 4, row 2, column 3
Regular data structures: *arrays*

- 4-D arrays alternative visualization
  - A 2x2x2x3 4-D array with 24 elements
  - Let’s call the array C
  - C[1][0][1][2] is 7

- N-D arrays are possible
N-D arrays

• Higher dimensional:
  – You can generalize the idea of 3-D, 4-D arrays. Namely, you can put 4-D arrays in an array to form a 5-D array.
  – In general, by putting n-D arrays in an array, you get a (n+1)-D array.
  – In the alternative view, you should have n+1 layers of boxes.
Advantages of arrays

- Advantages
  - Direct access to elements
    - By using index
  - Implicit structure, no storage wasted for structure
    - The indices are not saved separately, so no more space cost for the structure.
  - Simplicity
Disadvantages of arrays

• Disadvantages
  – Data size needs to be known a priori
    • Increasing array size is possible but expensive
  – Inserting / deleting elements is expensive
  – Does not model well irregular, non-uniform data
    • Huge room with mostly empty floor plan?
    • Airline route map?
    • Genealogical tree?
Disadvantages of arrays

• Example: Find and replace
  In a text editor, the find and replace function is very useful. But if the text is stored in an array, the replace operation is not very convenient.

Consider the sentence:
“The weather is getting cold.”
Now we replace “weather” by “water”. How do you do this?
Example: Replace

The weather is getting cold.

• Solution:
  – Let the array be ‘a’
  – Step 1: change the content in a[4] ~ a[8] to ‘water’
    The waterer is getting cold.
    The water is getting cold. d.
  – Step 3: reallocate the size of the array to 26 (the original length is 28)
    The water is getting cold.
Array modification

- Allocate array of new length
- Copy data from old array as needed
- Copy new data as needed
- Release old array
- Example: given an array A with 50 elements, insert 10 elements after element with index 15
  - Allocate array B with 60 elements
  - Copy elements 0-14 from A to B at 0-14
  - Copy new elements 0-9 to B at 15-24
  - Copy elements 15-49 from A to B at 25-59
  - Release old array A
Any question?