Week 14 - Monday

COMP 2100

Last time

- What did we talk about last time?
- Sorting visualization
- Timsort
- Tries

Questions?

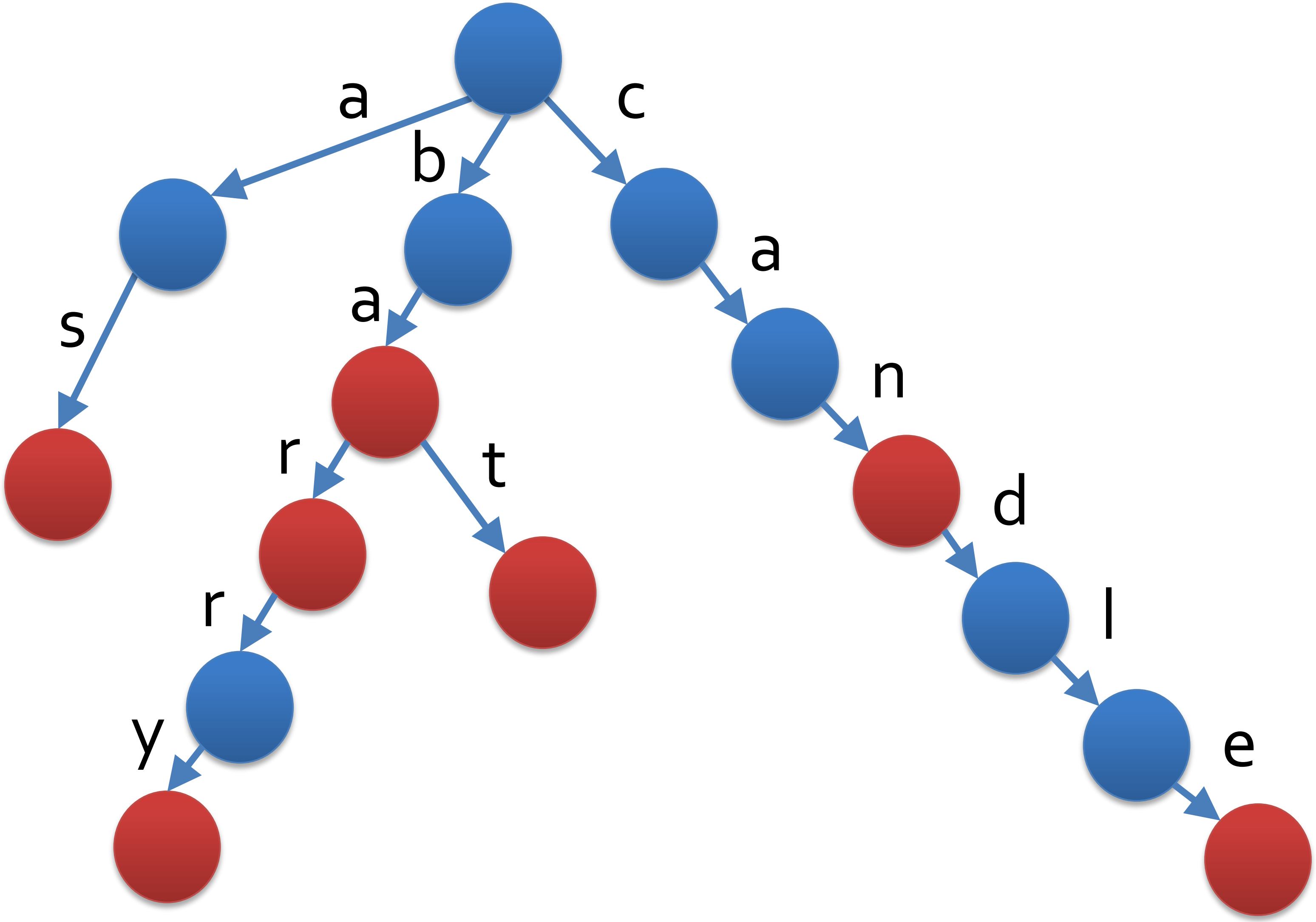
Project 4

Tries

Storing strings (of anything)

- We can use a (non-binary) tree to record strings implicitly where each link corresponds to the next letter in the string
- Let's store:
  - ba
  - bar
  - bat
  - barry
  - can
  - candle
  - as

Trie this on for size



Cost

- Let ***m*** be the length of a particular string
- Find Costs:
  - O(***m***)
- Insert Costs:
  - O(***m***)

Trie implementation

```
public class Trie {
  private static class Node {
    public boolean terminal = false;
    public Node[] children = new Node[128];
  }

  private Node root = new Node();
}
```

Trie Contains

Signature for recursive method:

```
private static boolean contains(Node node, String
  word, int index)
```

Called by public proxy method:

```
public boolean contains(String word) {
  return contains(root, word, 0);
}
```

Trie Insert

Signature for recursive method:

```
private static void insert(Node node, String word,
  int index)
```

Called by public proxy method:

```
public void insert(String word) {
  insert(root, word, 0);
}
```

Trie Traversal

```
private static void inorder(Node node, String prefix)
```

Called by public proxy method:

```
public void inorder() {
  inorder(root, "");
}
```

Trie implementations

- Keeping an array of length equal to all possible characters (usually) wastes space
- Alternatives:
    - **Ternary search tries:** A lot like a binary search tree, with smaller characters to the left, larger characters to the right, and continuations from the current character beneath
    - Keeping an array (or linked list) of the characters used, resizing as needed

Substring Search

Substring search

- Finding a string within another string is a fundamental task
- Applications:
  - Finding text on a web page
  - Find/replace while word processing
  - Looking for DNA subsequences within a larger sequence
  - Countless others …

Brute-force substring search

Write a method to find **`needle`** in **`haystack`**, returning the starting index of **`needle`** in **`haystack`** or **`-1`** if not found.

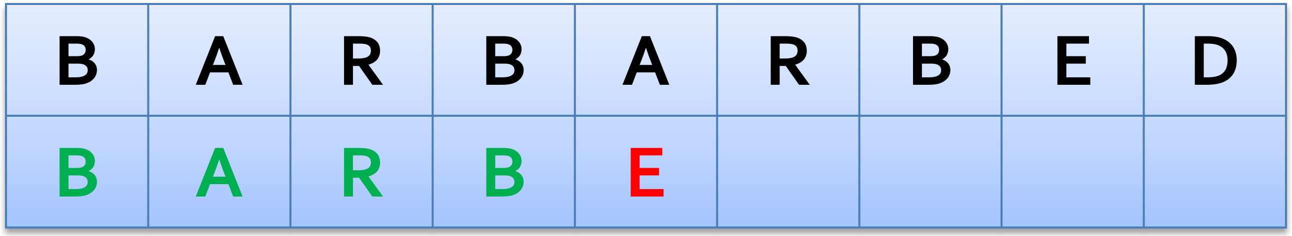
```
public static int find(String needle,
  String haystack)
```

Running time

- How long does the brute-force substring search take if the length of `haystack` is n and the length of `needle` is m?
- There are $n - m + 1$ positions to start looking in haystack, and you have to check m characters for each position
- $m(n - m + 1)$ is $\Theta(nm)$
  - Note that m is usually much smaller than n

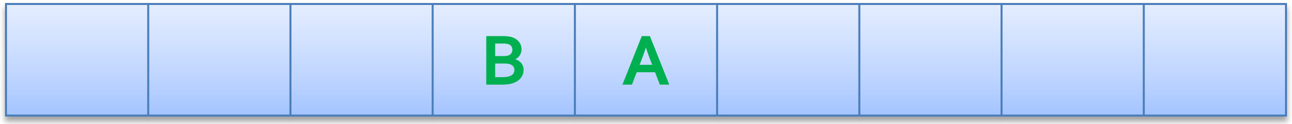
Knuth-Morris-Pratt

- A cleverer approach to substring search uses the observation that the act of matching tells us what to do when we reach a mismatch:
- Needle: BARBED
- Haystack: BARBARBED



B	A	R	B	A	R	B	E	D
B	A	R	B	E				

- On mismatch, skip ahead to:



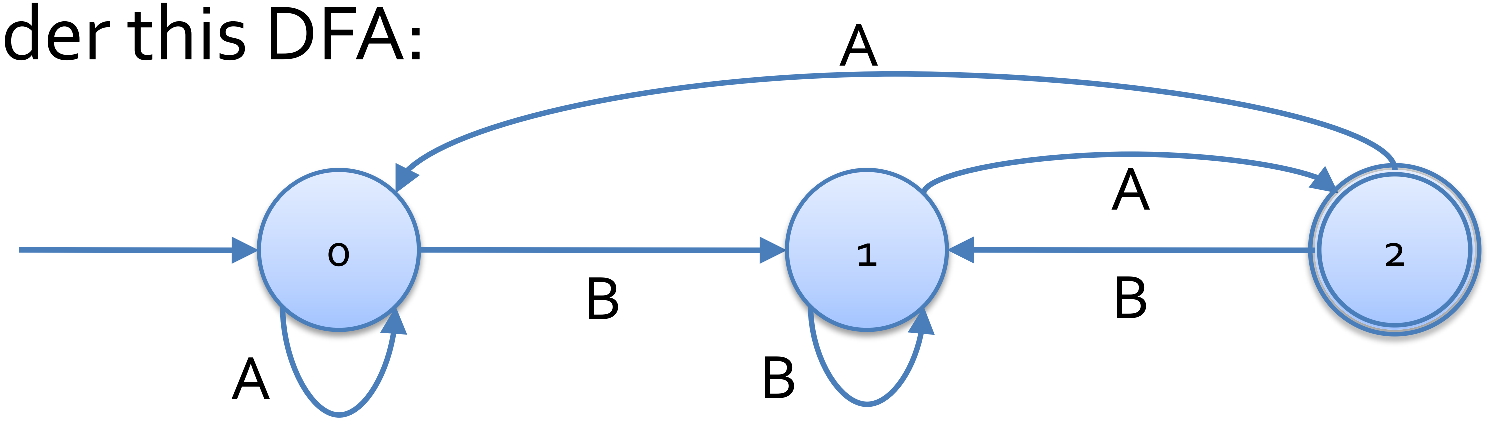
			B	A				

How do we know where to skip to?

- It depends on the structure of needle
- Some strings will have repetitive substrings that will "rematch" part of the substring
- Some strings will need to jump back to the beginning
- We could map these transitions out with a **deterministic finite automaton (DFA)**

DFA example

- Consider this DFA:



- State 0 is the initial state
- The circled state (2) is an accepting state
- Is the string AAAAABBA accepted?
- What about the string BBBBBBAB?
- What's a verbal description for the strings accepted?

DFA practice

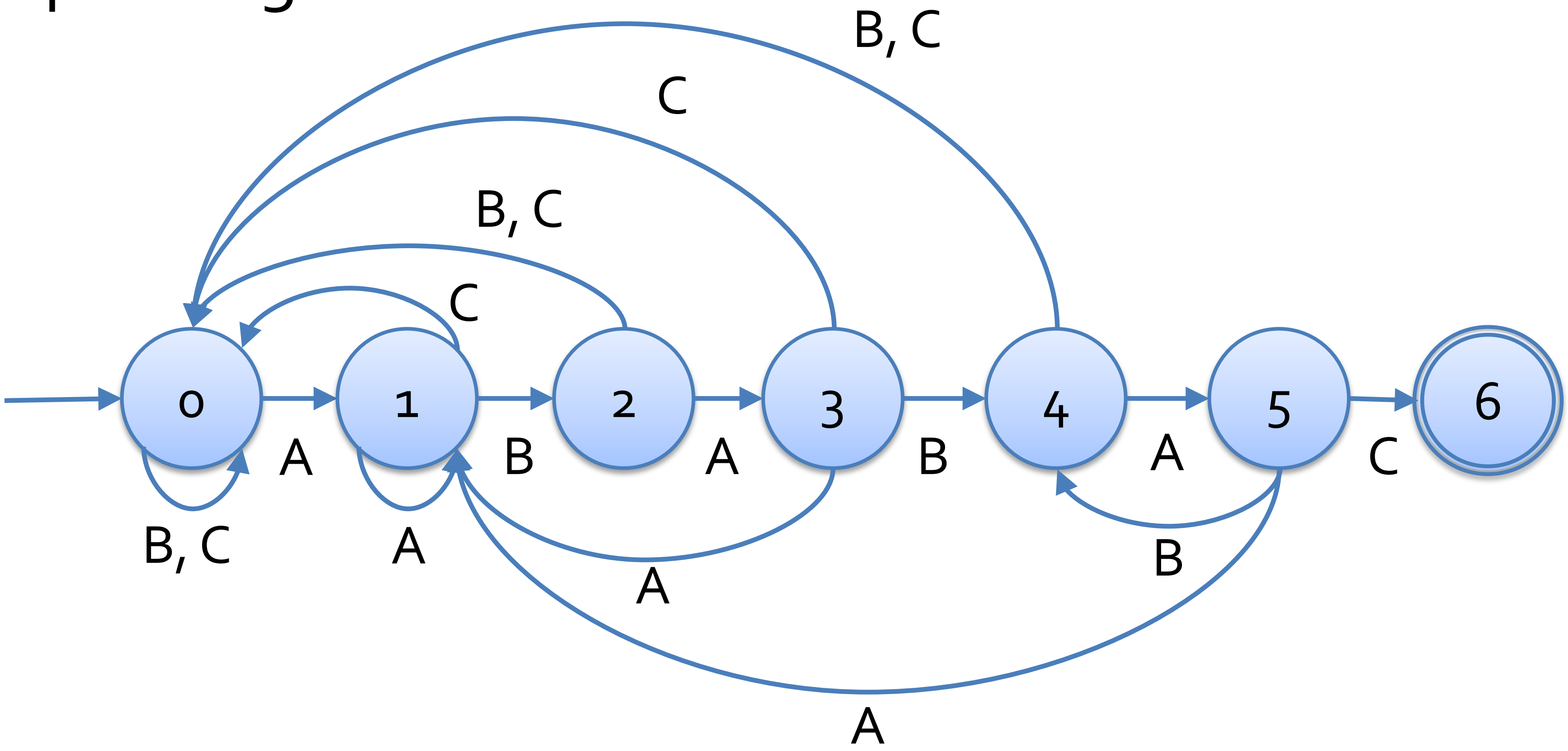
- Make a DFA that accepts all strings that have an even number of A's and an odd number of B's

Using DFAs

- DFAs can be created to accept many different patterns of strings
- They are equivalent to regular expressions
- Fortunately the DFAs needed for the Knuth-Morris-Pratt algorithm are easy to construct

KMP DFA example

- Needle string: ABABAC
- Corresponding DFA:



Making the DFA

- The algorithm for constructing the DFA is not obvious, but the code isn't very complex

```java
public static int[][] makeDFA(String pattern) {
      final int M = pattern.length();
      int[][] DFA = new int[128][M];
      // for all ASCII characters
      DFA[pattern.charAt(0)][0] = 1;
      for (int x = 0, i = 1; i < M; ++i) {
            for(char c = 0; c < 128; ++c)
                  DFA[c][i] = DFA[c][x];
            DFA[pattern.charAt(i)][i] = i + 1;
            x = DFA[pattern.charAt(i)][x];
      }
      return DFA;
}
```

Using the DFA

- Once you have the DFA, you can use it to search

```java
public static int find(String text, int[][] DFA) {
      final int M = DFA[0].length;
      int i, j;
      for(i = 0, j = 0; i < text.length() && j < M; ++i)
            j = DFA[text.charAt(i)][j];

      if(j == M)
            return i - M;
      else
            return -1;
}
```

Running time

- If the length of the pattern is ***m***, it takes ***m*** time to make the DFA
    - Actually, it's like 128***m*** or |***Alphabet***|***m***, but the size of the alphabet will always be constant
- If the length of the text is ***n***, it takes at most ***n*** time to do the search (often better if we make a match)
- Total running time is thus $\Theta(\boldsymbol{n} + \boldsymbol{m})$
- This improvement over brute force can be significant when ***n*** is large (as it often is)

Other approaches

- The KMP algorithm can process the text as a stream (without backing up or looking at more than one character at a time)
- You can't do better than KMP in the worse case
- However, Boyer-Moore substring looks for mismatched characters and can perform better in practice (but relies on analysis of random strings)
- Rabin-Karp constructs a fingerprint (a hash) of a sliding window of ***m*** characters
  - But there's always a chance that you match a substring that just happens to have the same hash

Quiz

Upcoming

Next time…

- Review up to Exam 1 next Monday

Reminders

- Work on Project 4
- Review up to Exam 1
- Have a great Thanksgiving!