Announcements

• Final Class 😭

• Help Opportunities😀
  • TAs available this weekend (see course calendar)
    • Sat. 3-5pm; Sun. 1-5pm
  • Tuesday, Dec. 12, 1:30-2:30 pm in Physics 205
  • Office Hours: M/T/W: 1:30-3:30pm (but see above!)

• Final Exam is Thursday, Dec. 14😬
  • 9:30-noon in Biology 112
  • Cumulative, but focused on second half of course
  • Sample exam and 2-page study sheet are on-line
Last Time

- Maps & Hashing
Today

• One More Problem
• Wrap-up
• SCS Forms
One More Problem!

• Given a graph $G = (V,E)$ where
  • $V = X \cup Y$, with $X \cap Y = \emptyset$
  • Every edge has one vertex in $X$ and one in $Y$
• Find a set of edges $M \subseteq E$ such that
  • No vertex is on more than one edge of $M$
  • $M$ is as large as possible
• $G$ is called a $bipartite$ $graph$ and $M$ is called a $maximum$ $matching$ of $G$
• Fun facts
  • $G$ is bipartite iff the vertices of $G$ can be 2-colored
  • $G$ is bipartite iff every cycle of $G$ has even length
Finding a Maximum Matching

• Idea: Look for *alternating* path between non-matched vertices
• Use it to *augment* the current matching
• Repeat until you can’t find any more of them.

Amazing Fact

If $M$ is a matching in a bipartite graph and there is no alternating path the augments $M$, then $M$ is a maximum matching for the graph!

Not too hard to prove

Uses structure of pairs of matchings
Wrapping Up
## Why Data Structures?

<table>
<thead>
<tr>
<th>Dictionary Structures</th>
<th>put</th>
<th>get</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted vector</td>
<td>(O(n))</td>
<td>(O(n))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>unsorted list</td>
<td>(O(n))</td>
<td>(O(n))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>sorted vector</td>
<td>(O(n))</td>
<td>(O(\log n))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>balanced BST</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>hash table</td>
<td>(O(1)^*)</td>
<td>(O(1)^*)</td>
<td>(O(\text{key range}))</td>
</tr>
</tbody>
</table>

*On average---with good design---Don’t forget!*
Data Structure Selection

• Choice of most appropriate structure depends on a number of factors
  • How much data?
    • Static (array) vs dynamic structure (vector/list)
  • Which operations will be performed most often?
    • Lots of searching? Use an ordered structure
      – If items are comparable!
    • Mostly traversing in arbitrary order? List
  • Is worst case performance crucial? Average case?
    • AVL tree vs SplayTree
Why Complexity Analysis?

• Provides *performance* guarantees
  • Captures effects of scaling on time and space requirements
• Independent of hardware or language
• Can guide appropriate data structure selection
Why Correctness Analysis?

- Provides *behavior* guarantees
- Independent of hardware or language
- Reduce wasted effort developing incorrect code
- A powerful debugging tool
  - Program incorrect: Try to prove it *is* correct and see where you get stuck
  - Frequently, such proofs are *inductive*
Why Java?

What makes it worth having to type (or read!)

Map<Airport,ComparableAssociation<Integer,
    Edge<Airport,Route>>> result = new
Table<Airport,ComparableAssociation<Integer,
    Edge<Airport,Route>>>(());
Why Java?

- Java provides many features to support
  - Data abstraction: Interfaces
  - Information hiding: public/protected/private
  - Modular design: classes
  - Code reuse: class extension; abstract classes
  - Type safety: types are known at compile-time

- As well as
  - Parallelism, security, platform independence, creation of large software systems, embeddability in browsers, ...
Why structure(5)?

- Provides a well-designed library of the most widely-used fundamental data structures
  - Focus on core aspects of implementation
    - Avoids interesting but distracting “fine-tuning” code for optimization, backwards compatibility, etc
  - Allows for easy transition to Java’s own Collection classes
- Full access to the source code
  - Don’t like Duane’s HashMap---change it!
Want to Learn More?

- **CS 237: Computer Organization**
  - Learn about the many levels of abstraction from high-level language → assembly language → machine language → processor hardware

- **CS 256: Algorithm Design and Analysis**
  - We’ve only scratched the surface of what elegant algorithm and data structure design can accomplish. For a deeper dive, go here.

- Many CS electives require one of these two courses
Want to Learn More?

• CS 334: Principles of Programming Languages
  • There are many different types of programming languages: imperative, object-oriented, functional, list-based, logic, ... Why!? What is required to support languages of these kinds?

• CS Colloquium
  • Weekly (Fridays at 2:30pm) presentations from active researchers in CS from across the country

• Duane’s Systems Journal Club
  • Weekly discussion of high-impact research papers
Thanks!

You’ve worked hard, asked great questions, and learned a lot!

Well done!

Any Questions?