Graph implementations

- Typical operations on graph:
  - Add vertex
  - Add edge (parameters?)
  - getAdjacent(vertex)
  - getVertices()
  - String->Vertex (vice versa)

- Different kinds of graphs
  - Lots of vertices, few edges, sparse graph
    - Use adjacency list
  - Lots of edges (max #?) dense graph
    - Use adjacency matrix

Graph implementations (continued)

- Adjacency matrix
  - Every possible edge represented, how many?
- Adjacency list uses $O(V+E)$ space
  - What about matrix?
  - Which is better?

- What do we do to get adjacent vertices for given vertex?
  - What is complexity?
  - Compared to adjacency list?

- What about weighted edges?

Memoization

- How do we avoid solving the same problem?
  - Consider APT BSTs
  - Review student submission
  - Consider similarities to Fibonacci
    - See next slide
  - How to avoid cost?

- How is this relevant to APT?
  - Create map of parameter to solution
  - Avoid recursion/solving when problem already solved

Fibonacci: Don’t do this recursively

```java
public long recFib(int n)
// precondition: 0 <= n
// postcondition: returns the n-th Fibonacci number
{
    if (0 == n || 1 == n) { return 1; }
    else { return recFib(n-1) + recFib(n-2); }
}
```

- How many clones/calls to compute $F(5)$?

- How many calls of $F(1)$?

- How many total calls?
14.5 Shortest path in weighted graph

- We need to modify approach slightly for weighted graph
  - Edges have weights, breadth first by itself doesn’t work
  - What’s shortest path from A to F in graph below?

- Use same idea as breadth first search
  - Don’t add 1 to current distance, add ???
  - Might adjust distances more than once
  - What vertex do we visit next?

- What vertex is next is key
  - Use greedy algorithm: closest
  - Huffman is greedy, ...

14.6 Greedy Algorithms

- A greedy algorithm makes a locally optimal decision that leads to a globally optimal solution
  - Huffman: choose two nodes with minimal weight, combine
    - Leads to optimal coding, optimal Huffman tree
  - Making change with American coins: choose largest coin possible as many times as possible
    - Change for $0.63, change for $0.32
    - What if we’re out of nickels, change for $0.32?

- Greedy doesn’t always work, but it does sometimes
- Weighted shortest path algorithm is Dijkstra’s algorithm, greedy and uses priority queue

14.7 Shortest Path (Unweighted)

- Mark all vertices with infinity (*) exec. starting vertex with 0
- Place starting vertex in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each adjacent vertex marked with *,
     i. process it,
     ii. mark it with source distance + 1
     iii. place it on the queue.

How do we get actual “Path”?

14.8 Shortest Path (Unweighted)

- Mark all vertices with infinity (*)
- Mark starting vertex with 0
- Place starting vertex in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each adjacent vertex marked with *,
     i. process it,
     ii. mark it with source distance + 1
     iii. place it on the queue.

How do we get actual “Path”?
Shortest Path (Weighted): Dijkstra

- Unmark all vertices and give infinite weight
- Set weight of starting vertex at 0 and place in priority queue
- Repeat until priority queue is empty:
  1. Remove a vertex from priority queue
     i. Process and mark (weight now permanent)
  2. For each adjacent unmarked vertex
     i. Set weight at lesser of current weight and (source weight + path weight)
        - May involve reducing previous weight setting)
     ii. Place in priority queue (if not there already)

How do we get actual "Path"?