

Game Graphs

- Known from Artificial Intelligence
- Present all steps in a game
- Who-takes-the-last-coin wins
- Wolf, sheep, cabbage

© 2005. 324-1-92

Breadth-First Search

```
bfs( Node n ) {
    Queue< Node > q = new Queue< Node >;
    Set< Node > marked = new Set< Node >;
    marked. add( n );
    q. enqueue( n );
    while( ! q. isEmpty() ) {
        Node u = q. getFirst();
        // do something with u;
        q. dequeue();
        for( Node w: u. getNeighbors() ) {
            if( ! marked. contains( w ) ) {
                marked. add( w );
                q. enqueue( w );
            }
        }
    }
}
```

© 2005. 324-1-93

Traversing Binary Trees

- Preorder
- Postorder
- Inorder

- Runtime: $O(n)$
- Easily done recursively

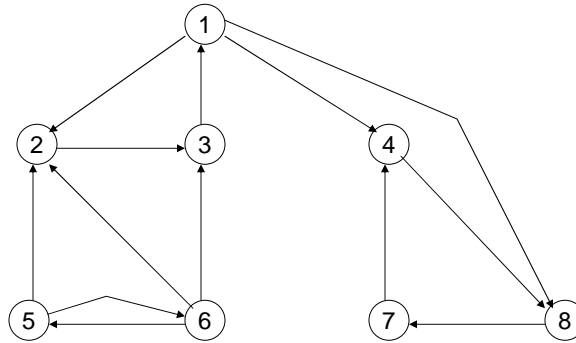
© 2005. 324-1-94

Traversing Graphs

- Depth-first
 - go to neighbor
 - go to neighbor of neighbor
 - recursively
 - need to remember which nodes visited already
- Breadth-first
 - put all neighbors in a queue
 - process each node
 - put its neighbors in queue
 - need to remember which nodes visited already
- Runtime
 - $O(\max(e, n))$
- Both can be used to create a spanning tree

© 2005. 324-1-95

Graph Traversal Example



© 2005. 324-1-96

Infinite Graphs

- Graph with infinitely many nodes and edges
- Cannot be searched completely
- But: can be searched heuristically
 - success depends on search strategy used
 - breadth-first often better!
 - why?

© 2005. 324-1-97

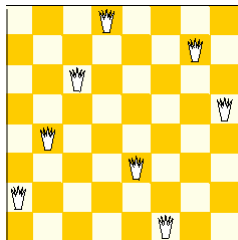
Implicit Graphs

- Graph is too large or too difficult to build
 - we build only that part of the graph we are working on
 - and shift it as needed
- Backtracking
 - finding a solution through depth-first search
 - if a branch doesn't work, back up
 - each path is considered a possible solution
 - partial parts may be discarded

© 2005. 324-1-98

Eight Queens Problem

- Position 8 queens on a chessboard (8*8) so that they do not threaten each other
- Suboptimal approaches:
 - try all combinations
 - try all combinations with none in the same columns
 - try all combinations with none in the same column and row



© 2005. 324-1-99

Eight Queens with Backtracking

- Represent chess boards by 8-vectors
 - each entry in vector: position of queen in a row
- k-Vector is promising if:
 - none of the queens threaten each other
- To solve the problem:
 - we need to find a vector that is 8-promising
- Create a tree of vectors
 - edge from u to v if:
 - u is promising k-vector
 - v is promising (k+1)-vector
 - first k elements are the same in both u and v
 - root: empty vector
 - leaves: solutions or dead ends
- Do backtracking on tree to find solutions
 - tree is implicit

© 2005. 324-1-100

Eight Queens with Backtracking

- Works best with depth-first search
- Performance:
 - using permutations of all positions:
 - $\binom{64}{8}$ = about 4 billion possibilities
 - using permutations of 8-vectors
 - 8^8 = about 16 million possibilities
 - using backtracking
 - number of nodes = 2057

© 2005. 324-1-101

Eight Queens with Backtracking

```
public boolean solve(int y) {
    int i,j;
    boolean r = false;

    for (i=0; i<8; i++) {
        if (grid[i][y] == 0) {           // for each row, try placing a queen
            addQueen(i,y,true);
            if (y == 7) {
                return true;           // we have placed all queens successfully
            } else {
                if (solve(y+1)) {
                    return true;       // we solved it, return
                } else {
                    addQueen(i,y,false); // remove the queen
                }
            }
        }
    }
    return false;                       // unable to solve down this branch -- retreat!
}
```

// By Aaron Davidson <<http://spaz.ca/>>