Review: BFS

- Breadth-first search uses a queue

Queue States

Termination

- To ensure termination, must remember states already seen and refuse to enqueue them. This probably requires another structure.

- Simple linked list
- Or, better, a hash table (see section 5.5)
  - Much faster
- Note: checking for game boards that reoccur, not board/predecessor combinations that reoccur!
Hashing

• Hashing computes a numeric signature for the proposed new item.

• It then uses the signature to access an array (called a hash table) of “equivalence classes” of items.

• Each equivalence class ideally has a relatively small number of items in it.
  – The only searching needed is that of searching the small equivalence class, not the whole universe.

How Java Helps

• Java provides method `hashCode()` of `Object`.
  – This gives a (generally large) number for any object.
  – Warning: by default generally based on memory address

• By dividing the hash code by the hash table size, equivalence classes are thereby formed.
  – All objects with the same hash code modulo the table size are considered equivalent.

• Java also provides a class `HashSet` that can be used to implement sets.
  – However, for the assignment it may be as simple (or simpler) to make your own hash function and own array of linked lists.

State Representation

• State representation does not have to be literal.

• Any structure that is sufficient to capture all relevant information will do.

• Often there are many choices.

• The choice may impact search efficiency.

Example of State Representation Choices: Towers of Hanoi

• Mapping from disk to spindle number:
  – `((a 2) (b 2) (c 1))`

• List of disks on each spindle, smallest first:
  – `((c) (a b) ( ))`

• Solving using rex, for example, the second would probably be more efficient:
  – All manipulation takes place at the tops of the stacks
Example of State Representation Choices: Traffic Jam

• Probably best *not* to replicate entire grid structure as state:
  - Heavyweight states
• Sample solution uses sequence of offsets for states:
  - Each offset indicates motion of a car from the original state.

Example of State Representation Choices: Traffic Jam

• Board representation = arrays of offsets:
  - initial state = [0, 0, 0, 0, 0]
  - understood: [red, blue, cyan, green, magenta] by position
  - immediate transitions to:
    - [0, 0, 1, 0, 0] (move cyan right)
    - [0, 0, -1, 0, 0] (move cyan left)
    - [0, 1, 0, 0, 0] (move blue down)
    - [0, 0, 0, 0, 1] (move magenta down)
    - [0, 0, 0, 0, -1] (move magenta up)
  • From each array, *and the initial grid*, we can construct the resulting grid.

States in Computing Machines

• Broad split:
  - Finite-state systems
    - Finite-state machines (chapter 12)
    - Finite-state systems with a very large number of states (most computers)
  - Infinite-state systems:
    - Turing machines
    - Most programming languages
    - Other structured automata

Turing Machines

• A very primitive model of computing
  - Conceptually simple to construct
• A very powerful model of computing
  - A Turing machine can compute any function your PC can.
Alan M. Turing

- Celebrated mathematician/computer scientist, 1912-1954
- Code-breaking machine (play: “Breaking the Code”)
- Reaction-diffusion equations
- Proving programs correct
- Artificial Intelligence
- Theory of computability

Turing Machine Depiction

- Ininitely-extendable tape (defaults to blank cells)
- \[ \square \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ \square \] (finite alphabet)
- \( q \) read/write head
- Control state (one of a finite set)

Control is determined by a finite set of rules (each a 5-tuple):
- (current-control-state, current-read-symbol, next-control-state, write-symbol, head-motion)
with the last three components being functions of the first two.

\texttt{tm} program on turing

- /cs/cs60/bin/tm
- \texttt{Examples in: /cs/cs60/tm/*.tm}
- \texttt{Sample execution:}

```
 turing > \texttt{tm add1.tm}
 1 0 1 1 1 1 1 1 1
 1 1 0 0 0 0 0 [\_]
 end
```

Rule set add1.tm

- (current-control-state, current-read-symbol, write-symbol, head-motion, next-control-state)

```
start _ _ left add1
add1 0 1 right end
add1 _ 1 right end
add1 1 0 left add1
end 0 0 right end
end 1 1 right end
```

Assumes that head starts to the right of the non-blank symbols on the tape.
Trace of add1.tm (use -t flag)

1 1 1 1 1 1 []
add
1 0 1 1 1 1 [1] _
add
1 0 1 1 1 1 [1] 0 _
add
1 0 1 1 1 [1] 0 0 _
add
1 0 1 1 [1] 0 0 0 _
add
1 0 1 [1] 0 0 0 0 _
add
1 0 [1] 0 0 0 0 0 _
add
1 [1] 0 0 0 0 0 0 _
end
1 1 0 [0] 0 0 0 0 _
end
1 1 0 0 [0] 0 0 0 _
end
1 1 0 0 0 [0] 0 0 _
end
1 1 0 0 0 0 [0] 0 _
end
1 1 0 0 0 0 0 [0] _
end
end

Other Examples

• See /cs/cs60/tm/*.tm
  - E.g., complete binary adder
  - See add.tm and add.doc

• Can go on to implement more complex operations
  - Multiplication, comparisons
  - Composition of operations
  - Looping, counters...
  - In particular, all of the so-called partial recursive functions
    on natural numbers.

Amazing Fact

• Any specific Turing machine has a finite description (transition table).
  - So, it is possible to put this description on the tape of another Turing machine.

• There exists a Universal Turing Machine
  - Starting with the tape containing a description of any machine T and an input, can simulate the computation of T on that input.
  - i.e., an interpreter for Turing machines.

Church’s Thesis

• a.k.a Turing's thesis, Church-Turing thesis, ...

• Every intuitively computable function is computable by some Turing machine.
  - (and hence by a universal turing machine)

• Generally accepted, but cannot be proved.
Implications

- The set of all computable functions can be enumerated (there is a “countable” number of them).
- There are non-computable functions.
- There are problems of interest that are unsolvable.

Non-Proof of Turing’s Thesis

- Proving Turing’s thesis formally would require formalizing the notion of computability.
  - This is what Turing set out to do.
  - But that formalization could be argued and to prove or disprove it would require a proof that that formalization completely captured the notion of computability.
  - We’d then be in a position similar to proving Turing’s thesis.

Disproving Turing’s Thesis

- Conceptually this is possible:
  - Find a function that everyone agrees is computable.
  - Prove that no TM can compute it.
- However, it is highly unlikely.
- Every attempt so far to characterize computability is equivalent to the TM.
  - Lambda calculus, partial recursive functions, phrase-structure grammars, register machines, quantum computers, ...