Data Structures in Java

Lecture 21: Introduction to NP-Completeness

12/9/2015

Daniel Bauer
Algorithms and Problem Solving

• Purpose of algorithms: find solutions to problems.

• Data Structures provide ways of organizing data such that problems can be solved more efficiently
  • Examples: Hashmaps provide constant time access by key, Heaps provide a cheap way to explore different possibilities in order…

• When confronted with a new problem, how do we:
  • Get an idea of how difficult it is?
  • Develop an algorithm to solve it?
Problem Difficulty

• We can think of the difficulty of a problem in terms of the best algorithm we can find to solve the problem.

  • Most problems we discussed so far have linear time solutions $O(N)$, or slightly more than linear $O(N \log N)$.

  • We often considered anything worse than $O(N^2)$ to be a bad solution.

  • For some problems we don’t know efficient algorithms.

• What is the best algorithm we can hope for, for a given problem? (for instance, $\Omega(N \log N)$ for comparison based sorting).
Polynomial and Exponential Time

• Two common classes of running time for algorithms:
  • Polynomial: $O(N^k)$ for some constant $k$.
  • Exponential: $O(2^{N^k})$ for some constant $k$. 
Hamiltonian Cycle

- A Hamiltonian Path is a path through an undirected graph that visits *every vertex* exactly once (except that the first and last vertex may be the same).
- A Hamiltonian Cycle is a Hamiltonian Path that starts and ends in the same node.

![Diagram of a Hamiltonian Cycle](image1)

No Hamiltonian Path
Hamiltonian Cycle

- Can check if a graph contains an Euler Cycle in linear time.
- Surprisingly, checking if a graph contains a Hamiltonian Path/Cycle is much harder!
- No polynomial time solution (i.e. $O(N^k)$) is known.
Traveling Salesman Problem (TSP)

Given a *complete*, undirected graph $G = (V,E)$, find the shortest possible cycle that visits all vertices.

Optimal Traveling Salesman Tour through all 48 continental state capitals.

Source: SAP
TSP - How many tours are there?

Given a complete, undirected graph $G = (V,E)$, find the shortest possible cycle that visits all vertices.

We can visit the vertices of the graph in ANY order.

How many possibilities are there?
TSP - How many tours are there?

Given a complete, undirected graph $G = (V,E)$, find the shortest possible cycle that visits all vertices.

We start at D. Because the graph is complete, we can go to any of the other N-1 nodes.
TSP - How many tours are there?

Given a complete, undirected graph $G = (V,E)$, find the shortest possible cycle that visits all vertices.

We start at D. Because the graph is complete, we can go to any of the other $N-1$ nodes.

Once we decide for a node, we can go to $N-2$ remaining nodes.
TSP - How many tours are there?

Given a complete, undirected graph $G = (V,E)$, find the shortest possible cycle that visits all vertices.

We start at D. Because the graph is complete, we can go to any of the other N-1 nodes.

Once we decide for a node, we can go to N-2 remaining nodes.

Once we decide for a node, we can go to N-3 remaining nodes.

…
TSP - How many tours are there?

Given a complete, undirected graph $G = (V,E)$, find the shortest possible cycle that visits all vertices.

We start at D. Because the graph is complete, we can go to any of the other N-1 nodes.

Once we decide for a node, we can go to N-2 remaining nodes.

Once we decide for a node, we can go to N-3 remaining nodes.

\[ (N - 1) \cdot (N - 2) \cdots (1) = (N - 1)! \]
TSP - How many tours are there?

Given a complete, undirected graph \( G = (V,E) \), find the shortest possible cycle that visits all vertices.

There are \((N - 1)!\) possibilities, but we can traverse complete tours in either direction.

There are \(\frac{(N - 1)!}{2}\) complete tours.

\(D\ A\ C\ B = D\ B\ C\ A\)
TSP - Brute Force Approach

Try all possible \( \frac{(N-1)!}{2} \) tours and return the shortest one.

Obviously this algorithm runs in \( O(N!) \)
TSP - Brute Force Approach

Try all possible \( \frac{(N-1)!}{2} \) tours and return the shortest one.

Obviously this algorithm runs in \( O(N!) \)

Better algorithm:

Dynamic Programming algorithm by Held-Karp (1962)

\[ O(2^N N^2) \]

No polynomial time algorithm is known!
How about a greedy approximation?

Start with node D, always follow the lowest edge until all vertices have been visited.
TSP - Greedy Approximation

How about a greedy approximation?

Start with node D, always follow the lowest edge until all vertices have been visited.
TSP - Greedy Approximation

How about a greedy approximation?

Start with node D, always follow the lowest edge until all vertices have been visited.
TSP - Greedy Approximation

How about a greedy approximation?

Start with node D, always follow the lowest edge until all vertices have been visited.
TSP - Greedy Approximation

How about a greedy approximation?

Start with node D, always follow the lowest edge until all vertices have been visited.

Unfortunately, this is not guaranteed to find an optimal solution.

cost = 36
**TSP - Greedy Approximation**

How about a greedy approximation?

Start with node D, always follow the lowest edge until all vertices have been visited.

Unfortunately, this is not guaranteed to find an optimal solution.

\[ \text{cost} = 34 \]
Combinatorial Optimization

• Many of the graph problems we discussed are *combinatorial optimization* problems.

• Select the “best” structure from a set of output structures subject to some constraints.

• Examples:
  
  • Select the minimum spanning tree from the set of all spanning trees.
  
  • Select the lowest-cost traveling salesman tour from the set of possible tours through a complete graph.
Bin Packing Problem

• You have:
  • N items of sizes $s_1, \ldots, s_N$
  • Any number of bins of fixed size V.

• Goal: pack the items into bins such that the number of bins needed is minimized. The sum of the item sizes in each bin must not exceed V.
Bin Packing Problem

- You have:
  - \( N \) items of sizes \( s_1, \ldots, s_N \)
  - Any number of bins of fixed size \( V \).

- Goal: pack the items into bins such that the number of bins needed is minimized. The sum of the item sizes in each bin must not exceed \( V \).
Knapsack Problem

I can only carry weight 10.
What should I take to maximize value.

W=9
$400

W=1
$300
W=4
$1,000

W=5
$200
W=1
$900
W=2
$600
W=2
$10
Knapsack Problem

• Given N items, each with a value $v_i$ and a weight $w_i$.

• Select a subset of the items to pack in a knapsack, such that
  • the total weight does not exceed some limit $W$
  • the sum of values is maximized.
Decision Problems

• A decision problem has, for each input, exactly two possible outcomes, YES or NO.

• “Does this Graph contain an Euler Circuit”
  “Does this Graph contain a Hamiltonian Cycle”
From Combinatorial Optimization to Decision Problems

• Any combinatorial optimization problem can be rephrased as a decision problem by asking if a decision that is better than a certain threshold exists.

• For instance, for TSP:
  “Is there a simple cycle that visits all vertices and has total cost $\leq K$”

• Observation:
  Solving the optimization problem is at least as hard as solving the decision problem.
Deterministic and Non-Deterministic Machines

- The “state” of a computation consists of all current data (input, memory, CPU registers,…) and the last program instruction.

- Given any state, a deterministic machine goes to a unique next instruction.
Deterministic and Non-Deterministic Machines

- A non-deterministic machine could be in ANY number of states at the same time.

- Equivalently, a non-deterministic machine contains an “oracle” that tells it the optimal instruction (of several multiple instructions) to execute in each state.
TSP with an Oracle

• State of the computation: Visited nodes, previous path.
• Same algorithm as greedy algorithm, but now the oracle tells us which edge to follow next.
TSP with an Oracle

- State of the computation: Visited nodes, previous path.
- Same algorithm as greedy algorithm, but now the oracle tells us which edge to follow next.
TSP with an Oracle

- State of the computation: Visited nodes, previous path.
- Same algorithm as greedy algorithm, but now the oracle tells us which edge to follow next.
TSP with an Oracle

- State of the computation: Visited nodes, previous path.
- Same algorithm as greedy algorithm, but now the oracle tells us which edge to follow next.

This algorithm produces an optimal tour in linear time!

Unfortunately, a real oracle is not realistic. (But we can have a limited amount of parallelism).
The Class of NP Problems

• NP (Non-deterministic Polynomial Time) is the class of problems for which a polynomial running time algorithm is known to exist on a non-deterministic machine.

• How do we know that a problem is in NP.

• Are there problems that are not in NP?
How Do We Know If a Problem Is in NP?

• Assume a decision problem produces YES on some input and some proof/“certificate” for this result.

• A decision problem is in NP if we can verify, in deterministic polynomial time, that the proof for a YES instance is correct.

• Examples:
  • An algorithm determines that a graph contains a Hamiltonian cycle and provides such a cycle as proof.
  • A spanning tree of cost < K is proof that such a spanning tree exists in a graph.
Undecidable Problems

• Are there problems that are impossible to solve?

• Halting Problem:
  
  • Given a program description and some input, determine if the program will terminate (halt) or run forever (loop).

    This problem is \textit{recursively undecidable}.

Turing 1936
The Halting Problem

- Assume we wrote a program called HALT(p,i)
  - HALT outputs “YES” and halts if $p$ halts on $i$.
  - HALT outputs “NO” and halts if $p$ loops on $i$. 

```
HALT
p i
p halts on i
p loops on i
```

```
output “YES”

output “NO”
```

```
halt

halt
```
The Halting Problem

Write another program called LOOP(q)
- LOOP halts if HALT(q,q) returns “YES”
- LOOP loops if HALT(q,q) returns “NO”
The Halting Problem

What happens if we run \text{LOOP(LOOP)}?

- Assume \text{LOOP(LOOP)} halts
  - Then \text{HALT(LOOP,LOOP)} must have returned “NO”.

- Assume \text{LOOP(LOOP)} loops.
  - Then \text{HALT(LOOP,LOOP)} must have returned “YES”.

\text{HALT}(q,q) \text{ returns “NO”} \rightarrow \text{halt}
\text{HALT}(q,q) \text{ returns “YES”} \rightarrow \text{loop}
A decidable problem that is (probably) not in NP

• Consider the problem of deciding if a graph does NOT have a Hamiltonian cycle.

• No NP algorithm is known for this problem.

• Intuitively, a proof would require to list all possible cycles. Verifying the proof means to show that none of them is Hamiltonian, one by one.
NP Problems

Decidable Problems

Undecidable Problems
P and NP Problems

• The class P contains all problems that are solvable in polynomial time on a deterministic machine (most of the problems discussed in this course are in P).

• Clearly, all problems in P are also in NP.

• Surprisingly, it is unknown if there are problems in NP (i.e. with proofs that can be verified in polynomial time), that cannot be SOLVED in polynomial time.

\[ P = NP \quad \text{vs.} \quad P \neq NP \]
P and NP Problems

Decidable Problems

P

NP Problems

if $P \neq NP$
P and NP Problems

Decidable Problems

P = NP Problems

if P = NP
NP-Complete Problems

• An NP problem is NP-Complete if it is at least as hard as any problem in NP.

• How do we know that a given problem $p$ is NP complete?

  • Any instance of any problem $q$ in NP can be transformed into an instance of $p$ in polynomial time.

  • This is also called a reduction of $q$ to $p$. 
Reductions

• Provide a mapping so that any instance of q can be transformed into an instance of p.

• Solve p and then map the result back to q.

• These mappings must be computable in polynomial time.
Problem Classes

Decidable Problems

NP Complete

NP Problems

P

if P ≠ NP
Importance of the NP-Complete Class

- Any other problem in NP can be transformed into an NP-Complete problem.

- If a polynomial time solution exists for any of these problems, there is a polynomial time solution for all problems in NP!

- To show that a new problem is NP-Complete, we show that another NP-complete problem can be reduced to it.
P and NP Problems

Decidable Problems

P = NP = NP Complete

if P = NP
Example Reduction

• Assume we know the Hamiltonian Circuit problem is NP-Complete.

• To show that TSP is NP-Complete, we can reduce Hamiltonian Circuit to it.

Hamiltonian Circuit (known to be NP-Complete) → Traveling Salesman Problem
Reducing Hamiltonian Circuit to TSP

• We want to know if the input graph $G = (V,E)$ contains a Hamiltonian Circuit.

• Construct a complete graph $G'$ over $V$.

• Set the cost of all edges in $G'$ that are also in $E$ to 1.0. Set the cost of all other edges to 2.0.
Reducing Hamiltonian Circuit to TSP

• Resulting TSP decision problem:
  • Does $G'$ contain a TSP tour with cost $\leq |V|$?
  • $G$ contains a Hamiltonian Circuit if and only if $G'$ contains a TSP tour with cost $= |V|$