CS 231
Naomi Nishimura

Recipes

Module 1

Recipe for exhaustive search:

1. Determine what the possibilities are.
2. Use the possibilities to solve the problem.

Module 2

Recipe for calculating costs:

1. Break the lines into blocks.
2. Determine the cost of each block.
3. Compute the sum of the costs.

Recipe for assessing an algorithm:

1. Determine the (worst-case) running time.
2. Ensure that the algorithm produces the correct output for any instance.

Module 3

Recipe for a greedy algorithm:

1. Build up the solution step by step.
2. Choose an option that is best at the moment.
3. Augment the solution using the chosen option.
4. Define options for the next step.
5. Repeat the process without ever undoing a decision.

Recipe for optimal substructure:

1. Using the optimal solution O for an arbitrary instance I, construct one or more smaller instances.
2. Decompose O into pieces, one for each smaller instance.
3. Show that if any piece O' of O is not an optimal solution for a smaller instance I' of I, then O is not an optimal solution for I.

Module 4

Recipe for iteration method:

1. Apply the definition of $T(n)$ iteratively, expressing $T(n)$ in a general form for any number of iterations.
2. Choose a number of iterations that reduces any smaller term to the base case.
3. Express $T(n)$ in closed form.

Recipe for substitution method:

1. Guess an upper bound for $T(n)$.
2. Use the guess for values on the right hand side.
3. Simplify the right hand side to prove the bound.
4. Check that the bound holds for the base cases.

Recipe for Master method:

1. For $T(n) = aT(n/b) + f(n)$, ignoring floors and ceilings, compare $f(n)$ and $x = n^{\log_b a}$.
2. If $f(n)$ is "smaller than" x, then $T(n)$ is in $\Theta(x)$.
3. If $f(n)$ is "bigger than" x, then $T(n)$ is in $\Theta(f(n))$.
4. If $f(n)$ and x are "the same size" then $T(n)$ is in $\Theta(x \log n)$.

Module 5

Recipe for dynamic programming:

1. Determine how the optimal solution can be formed from optimal solutions to smaller instances.
2. Determine what information should be stored in each table entry.
3. Determine the shape of the table or tables needed to store the solutions to the smaller instances.
4. Determine the base cases.
5. Choose an order of evaluation.
6. Extract the solution from the table.

Module 6

Recipe for determining an adversary lower bound:

1. Specify an adversary strategy.

2. Determine a number of steps T that any correct algorithm must take.

3. Show that after $T - 1$ steps of any algorithm, there will be at least two inputs consistent with the answers given by the adversary, and that they yield different outputs.

Recipe for membership in NP:

1. Give a polynomial-size certificate for each yes-instance.

2. Give a polynomial-time verification algorithm.

3. Show that the algorithm answers "Yes" for any yes-instance and its certificate.

4. Show that the algorithm is not fooled by false certificates for any no-instances.

Recipe for NP-completeness:

1. Prove C is in NP.

2. Select B that is known to be NP-complete.

3. Give an algorithm to compute a function f mapping each instance of B to an instance of C (it needn't map to all of C).

4. Prove that for any string x, if x is a yes-instance for B then f(x) is a yes-instance for C.

5. Prove that for any string x, if f(x) is a yes-instance for C then x is a yes-instance for B.

6. Prove that the algorithm computing f runs in polynomial time.

Module 7

Recipe for backtracking:

1. Specify a partial solution.

2. Determine how the children of a node are formed.

3. Choose when to backtrack.

Recipe for branch-and-bound:

1. Determine what to store at each node.

2. Decide how to generate the children of a node.

3. Specify what global information should be stored and updated.

4. Choose a bounding function.

Module 8

Recipe for analyzing an approximation algorithm:

1. Choose a new measure or problem.
2. Bound the approximate solution in terms of the new measure or problem.
3. Bound the optimal solution in terms of the new measure or problem.
4. Combine the two results to relate the approximate and optimal solutions.