15 Steps to Iterative Success

establishing the problem

1. specifications
2. examples

brainstorming ideas

3. targets
4. tactics
5. approaches

defining the algorithm

6. main steps
7. exit condition
8. setup
9. wrapup
10. special cases
11. algorithm

showing correctness

12. termination
    - measure of progress
    - making progress
    - reaching the end
13. correctness
    - loop invariant
    - establish the loop invariant
    - maintain the loop invariant
    - final answer

determining efficiency

14. implementation
15. time and space

Common Iterative Patterns

- process input
  - if the input is a collection of things, go through them one at a time
- produce output
  - if the output is a collection of things, produce them one at a time
  - can also apply if the output is something that can be built incrementally, like a sum
- narrow the search space
  - if the task is to locate an element among many, consider how to eliminate elements that aren't what you are looking for
  - not applicable if elements can't be eliminated without examining them directly

The advantage of identifying an applicable pattern is that the pattern also gives you patterns for many of the steps.

Main Steps and Exit Condition

Main steps.

- process input
  - incorporate the next input item into the solution to obtain a solution for one more element
- produce output
  - produce the next output item
- narrow the search space
  - eliminate some other elements

Exit condition.

- process input
  - when all of the input items have been processed
- produce output
  - when all of the output items have been produced
- narrow the search space
  - when the item has been found or the search space is empty

Measure of Progress and Loop Invariant

Measure of progress.

- process input
  - number of input items processed
- produce output
  - number of output items produced
- narrow the search space
  - size of the current search space

Loop invariant.

- process input
  - have a correct solution for the first *k* items
  - haven't gone wrong yet (the solution so far is still consistent with a solution for the whole problem)
- produce output
  - have produced the first *k* items of the output
- narrow the search space
  - either the element is within the current search space or it was never present
  - not all of the solutions (if there are any) have been eliminated

Loop Invariants

- a *loop invariant* is a statement that is true at the beginning of every loop iteration
  - used to show correctness
- a direct statement is to the effect that what has been done so far is correct
- for optimization problems, need both that the solution so far is legal *and* optimal
  - typically need an indirect statement for optimality – too difficult to show that the solution-so-far is optimal, but instead show that it has some property that at the end can be used to show optimality

Developing Iterative Algorithms

In a group of people, it is to be expected that some of them may not want to work with each other. Assuming that each person has at most *d* other people that they don't want to work with, divide the people into *d*+1 groups so that everyone is in exactly one group and no one is in a group with someone they don't want to work with.

Main steps.

- ☐ incorporate the next input item into the solution to obtain a solution for one more element
- ☐ for each element, process it
- ☐ for each person, put them into a group
- ☆ for each person, put them into the first group not containing someone they don't want to work with
- ☐ produce the next output item
- ☐ for each group, add people to that group as long as there isn't anyone in the group that they don't want to work with

process input – go through the input elements one at a time

main steps – loop body

process input version – incorporate the next input item into the solution to obtain a solution for one more element

apply the template to the specific problem

be specific enough about the process

Developing Iterative Algorithms

In a group of people, it is to be expected that some of them may not want to work with each other. Assuming that each person has at most *d* other people that they don't want to work with, divide the people into *d*+1 groups so that everyone is in exactly one group and no one is in a group with someone they don't want to work with.

Exit condition. The loop ends when --

- ☐ all of the input items have been processed
- ☐ when all of the ouput items have been produced
- ☆ when every person has been added to a group
- ☐ when everyone has been assigned to a legal group
- ☐ when no one else can be added to the current group
- ☐ when all of the groups are full
- ☐ when d+1 groups have been formed

process input – go through the input elements one at a time

exit condition – when the loop ends

process input version – when all of the input items have been processed

apply the template to the specific problem

should be a simple condition that is easily tested

does not usually involve properties related to the correctness of the solution

Developing Iterative Algorithms

In a group of people, it is to be expected that some of them may not want to work with each other. Assuming that each person has at most *d* other people that they don't want to work with, divide the people into *d*+1 groups so that everyone is in exactly one group and no one is in a group with someone they don't want to work with.

Measure of progress.

- ☐ something to count the number of people added to groups
- ☐ the number of elements considered
- ☐ the number of elements left to consider
- ☐ number of input items processed
- ☐ number of output items processed
- ☐ number of groups that have been filled
- ☆ number of people that have been assigned to a group
- ☆ number of people not yet assigned to a group

process input – go through the input elements one at a time

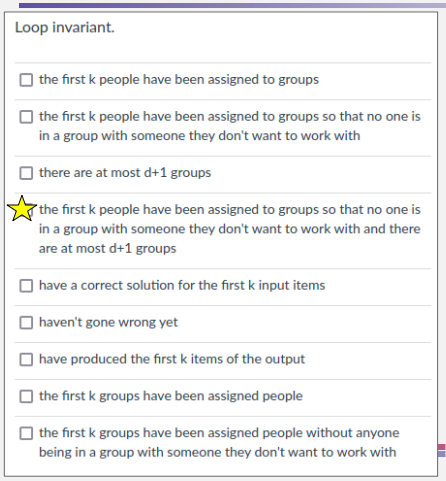
measure of progress – metric that can be used to tell you that you are getting closer to the solution

process input version – number of input items processed

apply the template to the specific problem

identify the actual measure, not that you need a variable

Developing Iterative Algorithms



process input – go through the input elements one at a time

loop invariant – condition true at the start of each iteration – should help establish correctness: loop invariant + exit condition = final answer

process input version – have a correct solution for the first *k* items

haven't gone wrong yet (solution is still consistent with a valid solution)

apply the template to the specific problem

needs to be related to the correctness of the solution