Writing Pseudocode

Most problems in 421 are written is pseudocode. The idea for pseudocode is to communicate a **high-level** description of an algorithm. Your goal is to be specific and concrete enough that

- A competent programmer (say, one of your classmates) could implement the code and get the right output.
- You can analyze the running time.
- You can prove the code correct.

BUT still be high-level enough that

- A reader can quickly and easily understand the main ideas behind your algorithm.
- A reader unfamiliar with ⟨ your favorite language ⟩ can still understand your algorithm and implement it in their favorite language.

1. Use Universal Code Idioms Explicitly

Loops, conditionals, and especially recursion are generally clearer in code-like structures, rather than in English. The only exception is extremely simple loops/conditionals.

1.1. Examples

Good

for i from 1 to n **do**
    `sum` $\leftarrow$ `sum` $+ i$
    if `sum` is even **then**
        print `i`

Bad

For every integer from 1 to n, add the integer to the running sum. If the sum to that point is even then print that integer.

The English is hard to understand! Is the "If..." inside the loop or outside? Even very ideas like this are probably better in code form. On the other hand, this principle isn't universal.

1.2. Examples

Ok

if $n < 100$ **then**
    `triangleCount` $\leftarrow 0$
    for each vertex u **do**
        for each vertex v **do**
            for each vertex w **do**
                if $u \neq v$ and $v \neq w$ and $u \neq w$ **then**
                    if $(u, v) \in E$ and $(u, w) \in E$ and $(v, w) \in E$ **then**
                        `triangleCount++`
    `trangleCount` $\leftarrow$ `triangleCount` / 6 ▷ divide by 3! for overcounting

Better

if $n < 100$ **then**
    Brute-force check every triple of distinct vertices and count the number of triangles.

Simple code, or code that is conceptually-simple, but takes up a lot of space can often be replaced

2. Use Variable Names

Look at the bad example from the last section. "The running sum" and "the sum" refer to the same thing, but we used different words for them. In everyday English this is ok, but potentially confusing. In the last line is "that integer" the iteration-counting variable (`i`) or the running sum (`sum`)?

In general, anytime

- You need 3-or-more words to refer to a thing (like "the vertex that just came off the queue" or "the edge we are currently examining") OR
- You refer to the same thing with words multiple times ("the integer" and later "that integer")

You probably want a variable name.

3. Use Math Notation and Terminology

Good

Let $\bar{x}$ be the mean of $x_1, ..., x_n$

Good

`mean-x` $\leftarrow \frac{x_1+\cdots+x_n}{n}$

Bad

`sum` $\leftarrow 0$
for i from 0 to $n-1$ **do**
    `sum` $+= x[i]$
`mean` $\leftarrow$ `((double) sum )/ n`

It takes 4-or-so lines of real code to communicate this idea. It takes one line or math or English. Use the thing that's faster to read, write, and understand!

4. Make sure your intention is unambiguous

A sentence like:

for every element: decide using memoization if it's better to keep this element or omit this element by comparing the maximum cost including this element vs maximum cost excluding this element

is not specific enough for pseudocode. What are those values that I'm comparing? Once I know them what does it mean to be better (bigger? smaller? Closer to a target value?). This would be a good 1-sentence of intuition in a comment; it's not specific enough for a programmer to implement.

5. In general...

- If you could put your pseudocode into a python interpreter, and it would pretty much compile, you're probably including too much detail.
- If you ever use three or more words to refer to an object (like "the vertex we're processing now" or "the next vertex to come out of the queue") you probably want to make things more code-like (e.g. a variable name like `curr` or notation like `next = queue.removeMin()`)
- Phrases like "repeat this process until..." or "...and so on" or "make a recursive call" are likely to be ambiguous in English and are much better written in code form.
- Use good style in your pseudocode! Indent and/or use braces for nested structures, use meaningful variable names, you can even write comments!

6. A Bigger Example

Modify BFS to find all vertices v such that there is a walk from u to v using exactly 3 edges (in an unweighted graph G).

Bad
Modify BFS by adding a boolean `reachIn3` and an integer `dist`. Every time you process an edge, update the destination vertex's `dist` to be one more than the other vertex's `dist`. If it's 3, set `reachIn3` to be true. Put the vertex on the queue. Additionally, if a distance ever becomes 4 or more, the algrotihm ends.

This snippet is very hard to understand! What's "the other rvertex"? Which vertex is "the vertex" put onto the queue? Variable names would make this better, but there are still A LOT of changes here to understand exactly what is being said.

good

Place u on a Queue q, set u.`dist` to 0.
while q is not empty **do**
    `curr` $\leftarrow q$.`dequeue()`
    for each edge $(\texttt{curr}, v)$ **do**
        $v.dist \leftarrow$ `curr.dist` $+ 1$
        if `v.dist==3` **then**
            `v.reach3` $\leftarrow$ `true`
        else if `v.dist<3` **then**
            `q.enqueue(v)`
        else
            break

7. In LaTeX

We use the AlgPseudocode package to make our staff solutions. algorithm2e is another commonly-used package. Both have documentation and examples available.

We use an alternate font to refer to variable names or other code-like things. We use \texttt{}, but other fonts (or even the standard font) is fine as long as we can understand.

If you wish to avoid one of the LaTeXpackages and just type what you want to show up, you can try the verbatim environment (put \begin{verbatim} and \end{verbatim} around the text you'd like to appear). Note that you cannot use commands or math-mode inside the verbatim environment. The main advantage of verbatim is that it doesn't "eat" your tabs/spaces and newlines the way normal LaTeXdoes.