Chapter 21

Case study: The largest rectangle under a histogram

The case study of this chapter is a notoriously tricky development, and the several approaches to it include using auxiliary sequences or intermediate tree structures. Here however we use proper (not tail-) recursion, for which simple variables suffice.[1]

The problem is to find the area of the largest rectangle under a given histogram, as illustrated in Figures 21.1 and 21.2. The straightforward complexity of the problem is given by the number of possible bases for the rectangle ($\approx N^2$) times the cost of examining each to determine its height ($\approx N$) — a cubic algorithm, in other words.

A divide-and-conquer approach, in which we split the problem into smaller pieces, will lead us first to $N \log N$ complexity. But, with some effort, we shall do even better than that.

21.1 Laying the groundwork

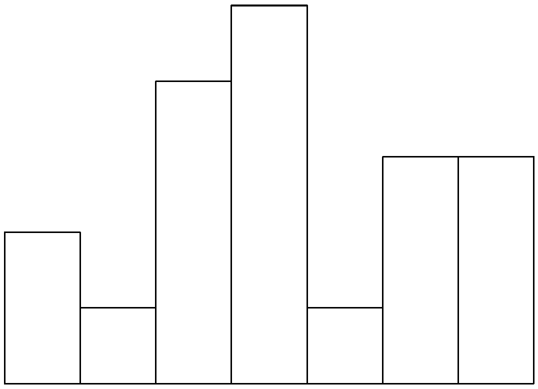
Assume we have a sequence of non-negative integers $hs : \mathbf{seq}_N\, \mathbb{N}$ that represents a *histogram*, as in Figure 21.1, under which we are to find the largest rectangular area, as in Figure 21.2. Since the base of a largest rectangle is sufficient to determine it completely (since it should be as high as the histogram will allow), we consider the problem to be the finding of that base, represented below as a pair of indices l and h denoting its start and end:

$$(\sqcup l, h : \mathbb{N} \mid l \leq h \leq N \cdot (h - l) \times \sqcap hs[l \to h]) \ .$$

In fact, we will need to be more general, looking for largest rectangular areas under subsegments $hs[i \to j]$ of the histogram; thus we define natural number $\mathsf{lr}(i,j)$ to be the largest rectangular area under the histogram $hs[i \to j]$ —

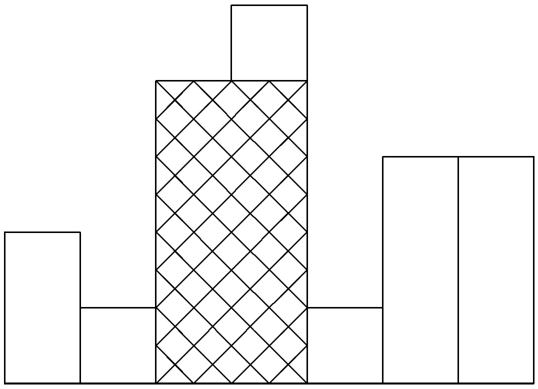
$$\mathsf{lr}(i,j) \ \widehat{=} \ (\sqcup l, h \mid i \leq l \leq h \leq j \cdot (h - l) \times \sqcap hs[l \to h])$$

[1]Naturally the solutions are related, since each approach more or less encodes the structures found in the others.



$hs = \langle 2, 1, 4, 5, 1, 3, 3 \rangle$

Figure 21.1 Example histogram



Rectangle under $\langle 2, 1, 4, 5, 1, 3, 3 \rangle$

Figure 21.2 Largest rectangle

— and with that specify our program

$$a := \mathsf{lr}(0, N) \ , \qquad \text{(i)}$$

in which the variable $a : \mathbb{N}$ will contain the result.

21.2 Divide and conquer

A divide-and-conquer approach to this problem suggests finding a division of hs into two pieces such that solutions to the pieces can be combined into a solution for the whole. In fact we shall be especially interested in splitting at minimum values of hs, because of the following property of lr:

> If $hs[min]$ is a minimum value in hs, then the largest area of a rectangle under hs is the maximum of
>
> 1. the largest under $hs[0 \rightarrow min]$,
> 2. the largest under $hs[min + 1 \rightarrow N]$ and
> 3. the area given by base N and height $hs[min]$.

Thus we need only find a minimum element $hs[min]$ of hs, and then solve recursively the subproblems $hs[0 \rightarrow min]$ and $hs[min{+}1 \rightarrow N]$ on either side. The property above then gives us the solution for the whole.

A rough estimate for the average time required by the above approach, in terms of N the size of hs, gives the recurrence

$$\mathsf{time}\, N \leq N + 2 \times \mathsf{time}(N/2) \ ,$$

which leads to $\mathsf{time}\, N \preceq N \log N$. (Think of the fact that the sequence can be halved approximately $\log N$ times, and the time taken altogether on each level is still proportional to the original N.)

With some iteration as well we can do even better than $N \log N$, however. We begin by seeking to maintain a 'running maximum' as longer and longer initial segments of hs are considered. Our first step is therefore the usual

$$
\begin{array}{lll}
\text{(i)} \ \sqsubseteq & \textbf{var}\ i : \mathbb{N} \bullet & \\
& a, i := 0, 0; & \\
& \textbf{do}\ i \neq N \rightarrow & \\
& \qquad a, i \colon [i \neq N \ , \ i \leq N \wedge a = \mathsf{lr}(0, i) \ , \ i_0 < i] & \text{(ii)} \\
& \textbf{od}\ . &
\end{array}
$$

But now we become more adventurous: bearing the property above in mind, we might be interested in increasing i in the iteration body by more than just 1. If $hs[i]$ becomes a minimum value for the longer segment — that is, if i is increased to some j perhaps greater than $i + 1$, and we have that $hs[i]$ is a minimum value of the segment $hs[0 \rightarrow j]$ — then it is particularly easy to re-establish $a = \mathsf{lr}(0, j)$, provided we can calculate $\mathsf{lr}(i + 1, j)$ as required by Item 2 of the property. Thus we proceed

$$\text{(ii)} \ \sqsubseteq \ \textbf{var}\ b, j : \mathbb{N} \bullet$$

$$\left[\begin{array}{c} b, j \\ \hline i < N \\ \hline i < j \le N \\ hs[i] \le hs[0 \rightarrow j] \\ b = \mathsf{lr}(i+1, j) \end{array}\right];$$
$$a, i := a \sqcup b \sqcup j \times hs[i], j \ .$$

We have of necessity adopted an alternative layout for specifications in which the components appear in the same order, but vertically instead of horizontally, and new line is interpreted as conjunction.[2]

The first command sets j as suggested, and the second re-establishes the invariant.

The conjunct $b = \mathsf{lr}(i+1, j)$ in the first command is where we foreshadow a recursion, since it is effectively solving our original problem but on a smaller segment. Given that $a = \mathsf{lr}(0, i)$ (which was Item 1), that $b = \mathsf{lr}(i+1, j)$ (given in Item 2) and the explicit calculation $j \times hs[i]$, it can be seen that the assignment command does re-establish the invariant $a = \mathsf{lr}(0, i)$ for the now-larger i.

But it is not as simple as that, unfortunately; we do have a problem. The first command above is infeasible, because its postcondition implies $hs[i] \le hs[0 \rightarrow i]$, something which assignments only to b and j cannot affect.

21.3 Strengthening invariants to restore feasibility

The dead end is only apparent. If $hs[i] \le hs[0 \rightarrow j]$ is in the postcondition, but cannot necessarily be established by changing only b and j, then the 'missing conjunct' $hs[i] \le hs[0 \rightarrow i]$ must come from the precondition. How do we put it there? (*Strengthening* the precondition is not an example of refinement!)

A similar problem occurs in proofs by mathematical induction, when the proof that the inductive hypothesis holds at $i+1$, given its truth at i, does not go through. If the truth at i does imply truth at $i+1$, but the proof cannot be found, then either the approach is unnecessarily difficult or the mathematician insufficiently skilled. For us that would correspond to a *feasible* iteration body that we nevertheless could not see how to develop further. If on the other hand the truth at i does *not* imply truth at $i+1$, then the inductive hypothesis may be too weak; strengthening it makes its assumption at i more powerful, but the obligation to re-prove it at $i+1$ becomes accordingly more difficult. That corresponds to an iteration body that is *in*feasible: strengthening the invariant then strengthens the precondition (as we wanted, and makes development easier), but also strengthens the postcondition (making development more difficult).

In both cases, induction and iteration, the necessary strengthening is usually not apparent until the first attempt at proof or development has already failed; and

[2] That is rather like a Z *schema*, but with a frame.

thus excursions like the above in Section 21.2 are not fruitless. In our case we have discovered that we need the extra conjunct $hs[i] \leq hs[0{\rightarrow}i]$ in the precondition, and so we simply place it in the invariant and try again: our iteration body (ii) becomes (ii') below, and we resume from there, performing the same first steps as before:

$$\sqsubseteq \left[\begin{array}{c} a, i \\ \hline i \neq N \\ \hline i \leq N \\ a = \mathsf{lr}(0, i) \\ hs[i] \leq hs[0{\rightarrow}i] \\ \hline i_0 < i \end{array}\right] \qquad \text{(ii')}$$

$$\begin{array}{l} \sqsubseteq \ \textbf{var}\ b, j : \mathbb{N}\bullet \\ \quad \left[\begin{array}{c} b, j \\ \hline i < N \\ a = \mathsf{lr}(0, i) \\ hs[i] \leq hs[0{\rightarrow}i] \\ \hline i < j \leq N \\ a = \mathsf{lr}(0, i) \\ hs[j] \leq hs[0{\rightarrow}j] \\ hs[i] \leq hs[0{\rightarrow}j] \\ b = \mathsf{lr}(i+1, j) \end{array}\right]; \qquad \triangleleft \\ \quad a, i := a \sqcup b \sqcup j \times hs[i], j \end{array}$$

$$\sqsubseteq \left[\begin{array}{c} b, j \\ \hline i < N \\ \hline i < j \leq N \\ hs[i] \leq hs[i+1{\rightarrow}j] \\ hs[j] \leq hs[i] \\ b = \mathsf{lr}(i+1, j) \end{array}\right]. \qquad \text{(iii)}$$

The only non-immediate reasoning above relies on the implication

$$\begin{array}{ll} & hs[i] \leq hs[0{\rightarrow}i] \wedge hs[i] \leq hs[i+1{\rightarrow}j] \wedge hs[j] \leq hs[i] \\ \Rrightarrow & hs[j] \leq hs[0{\rightarrow}j] \wedge hs[i] \leq hs[0{\rightarrow}j] \ . \end{array}$$

Remember that $hs[j]$ lies just beyond the segment $hs[0{\rightarrow}j]$, and that the first conjunct $hs[i] \leq hs[0{\rightarrow}i]$ of the antecedent comes from the precondition.

The interesting thing about the last step is that a possible recursion has popped up: our original problem is an instance of (iii), provided we define 'formally' that $hs[-1]$ and $hs[N]$ are both -1 (any negative number would do). That is because setting i to -1 initially will force (iii) to establish $j = N$: the postcondition contains $hs[j] \leq hs[i]$, and we know that $hs[0{\rightarrow}N] \geq 0$, and j cannot be -1. (It must be strictly greater than i.) That is, the postcondition then implies

$$b = \mathsf{lr}(0, N) \ ,$$

making b the value we are looking for.

That is all the excuse we need for a recursive procedure.

21.4 Introducing recursion

Because our original problem has reappeared in (iii), we now aim explicitly for recursion by making it a procedure. With some foresight,[3] we provide three parameters:

$$
\begin{array}{l}
\textbf{procedure } \mathit{Hist}\ (\textbf{value } i : \mathbb{Z}; \textbf{result } b, j : \mathbb{Z}) \\
\ \widehat{=}\ \ \text{(iii)}\ .
\end{array}
$$

Since our problem is now more general than before, however, we will need a generalisation of the property we exploited earlier; it is

$$
\begin{array}{lcl}
\mathsf{lr}(l,h) & = & 0, \text{ if } l = h; \text{ and} \\
\mathsf{lr}(l,h) & = & \mathsf{lr}(l,i) \sqcup \mathsf{lr}(i+1,h) \sqcup hs[i] \times (h-l)\ , \\
 & & \text{provided } l \le i < h \wedge hs[i] \le hs[l{\rightarrow}h].
\end{array}
$$

Our original version of this simply had $l = 0$ and $h = N$.

Carrying on, the variant is i, and it is already (in the precondition of (iii) above) strictly bounded above by N:

$$
\text{(iii)}\ \sqsubseteq\ \textbf{variant } I \text{ is } i \bullet
\left[\begin{array}{c}
b, j \\ \hline
i = I < N \\ \hline
i < j \le N \\
hs[i] \le hs[i+1{\rightarrow}j] \\
hs[j] \le hs[i] \\
b = \mathsf{lr}(i+1, j)\ .
\end{array}\right]
$$

And, forewarned by our having had to strengthen the invariant above, we choose here an invariant 'stronger than the obvious' in that it includes the conjunct $hs[j] \le hs[i+1{\rightarrow}j]$. The iteration will establish (the negation of its guard) that $hs[j] \le hs[i]$, giving

$$
\begin{array}{l}
\sqsubseteq\ \text{“invariant is middle formula of iteration body”} \\
\quad b, j := 0, i+1; \\
\quad \textbf{do } hs[j] > hs[i] \rightarrow \\
\qquad \left[\begin{array}{c}
b, j \\ \hline
hs[j] > hs[i] \\ \hline
I < j \le N \\
hs[i] \le hs[i+1{\rightarrow}j] \\
hs[j] \le hs[i+1{\rightarrow}j] \\
b = \mathsf{lr}(i+1, j) \\ \hline
j_0 < j
\end{array}\right] \qquad \triangleleft \\
\quad \textbf{od}
\end{array}
$$

[3]The prophesy is only apparent, since we would later discover in any case what the parameters should be.

$$\sqsubseteq \textbf{var } c, k : \mathbb{N}\bullet$$

$$\left[\begin{array}{c} c, k \\ \hline I < j < N \\ \hline j < k \leq N \\ hs[j] \leq hs[j+1 \rightarrow k] \\ hs[k] \leq hs[j] \\ c = \mathsf{lr}(j+1, k) \end{array}\right]; \qquad \triangleleft$$

$$b, j := b \sqcup c \sqcup (k - i - 1) \times hs[j], k \ .$$

It has to be admitted that the second step above conceals a certain amount of detailed working out of implications. But it was not altogether necessary to see beforehand that they *would* work out, since the first stage of our development had already suggested that we proceed that way, with the following assignment given just a transliteration of the one we met before.

One genuinely obscure step, however, is the apparent strengthening of the precondition, from $j \leq N$ to $j < N$. That is necessary to match the offered recursion, where $i < N$ occurs in the precondition; and it is justified by the fact that if $hs[j] > hs[i]$, then j cannot be N.

And that's it: we finish off with

$$\sqsubseteq \ \textit{Hist}\ (j, c, k) \ .$$

21.5 Wrapping up

We have now reached code, if by a slightly circuitous route. The first stage of the development produced an iteration body which generalised the original problem; with that encouragement, we were able to continue the development by introducing a recursion. Rather than calling the procedure within the iteration, however, we now exploit the generalisation by discarding the outer iteration, replacing the whole thing by a procedure call. In other words, we redo the first part of our development in a stroke:

$$a := \mathsf{lr}(0, N)$$

$$\sqsubseteq \left\{ \begin{array}{l} \text{assume } hs[-1] = hs[N] = -1 \\ \textbf{var } j : \mathbb{N} \end{array} \right. \bullet$$

$$\boxed{\text{Specification (iii) above}}$$

$$\sqsubseteq \ \textit{Hist}\ (-1, a, j) \ .$$

The declaration of j is necessary only to supply the third parameter of the procedure call: the result returned there is not used.

That concludes the development, and the code is shown in Figure 21.3. Exercise 21.1 investigates whether all this effort has improved the execution time over the straightforward $N \log N$.

$$
\begin{array}{l}
\textbf{procedure}\ \mathit{Hist}\ (\textbf{value}\ i : \mathbb{Z};\ \textbf{result}\ b, j : \mathbb{Z})\ \widehat{=} \\
|[\ \ \textbf{var}\ c, k : \mathbb{N} \bullet \\
\quad b, j := 0, i+1; \\
\quad \textbf{do}\ hs[j] > hs[i] \rightarrow \\
\qquad \mathit{Hist}\ (j, c, k); \\
\qquad b, j := b \sqcup c \sqcup (k - i - 1) \times hs[j], k \\
\quad \textbf{od} \\
]|
\end{array}
$$

with 'main program' $|[\ \textbf{var}\ j : \mathbb{Z} \bullet \mathit{Hist}\ (-1, a, j)\ ]|$.

Figure 21.3 Collected code.

21.6 Exercises

Ex. 21.1 ♡ What is the running time of the algorithm, in terms of the size of hs? Count calls of *Hist*. *Hint*: What values of i are passed successively to *Hist* as the algorithm executes?

Ex. 21.2 ♡ Specify and develop a program that finds 'the largest true rectangle' in a two-dimensional array of Boolean values: the largest area of any rectangle, within the array, that contains only `true` values.

Ex. 21.3 ♡ Specify and develop a program that finds 'the largest true *square*' in a two-dimensional array of Boolean values: the largest area of any square, within the array, that contains only `true` values. *Hint*: There may be a much simpler approach than was required for the largest true rectangle. What is the essential property of a square that prevents that simpler approach working for rectangles as well?