

Sequence Processing

Traversal and Search Algorithmic Schemes

Professors de Pro1

Department of Computer Science

Contents

- 1 Introduction
- 2 The Treat-All Scheme
- 3 The Search Scheme
- 4 Working with Windows
- 5 Windows over Computed Values
- 6 Sequences of Sequences
- 7 Summary

Processing a Sequence

Setting

The input is a sequence of data items (char, string, int, ...).

Roughly, any process over the input sequence falls into one of these categories:

- 1 A process that requires the evaluation of **every** item → **TREAT ALL** problem.
- 2 A process whose goal is to check whether **some** item satisfies a property → **SEARCH** problem.
- 3 (rarely) A combination of the two.

Goal

We study a generic skeleton for each kind of problem, and a technique (the **window**) that helps solve many variants of both.

The Treat-All Scheme

Idea

Every item must be visited and processed in order to build the result. The loop cannot stop early.

```
1 initialize;
2 while (not end_of_seq) {
3     get a new data item d;
4     treat d;
5 }
6 //all data items have been processed
```

Example: counting occurrences

Specification

Given a sequence of strings, count the number of times "anna" appears.

Example

in: anna lluis olga anna pere **out:** 2

```
1 int counter = 0;
2 string name;
3 while (cin >> name)
4     if (name == "anna") ++counter;
5 cout << counter << endl;
```

Example: maximum of a sequence

Specification

//pre: non-empty sequence of non-negative integers

//post: computes the maximum of the sequence

Example

in: 3 1 5 8 0 **out:** 8

```
1  int max = -1;  // "fake" but suitable initialization
2  int n;
3  while (cin >> n)
4      if (n > max) max = n;
5  cout << max << endl;
```

Initialization trick

Since all items are non-negative, -1 is a safe sentinel: any real item will beat it on the first comparison.

The Search Scheme

Idea

We only need to know whether **some** item satisfies a property.

```
1 initialize;
2 bool found = false;
3 while (not found and not end_of_seq) {
4     get new data item d;
5     if (property(d)) found = true;
6 }
7 //found tells whether an item satisfying property exists
```

Key point (very important!)

The code **must stop** as soon as an item with the required property has been found.

Example: search for "pere"

Specification

Given a sequence of strings, check if "pere" appears.

Example

in: pau roger pere marta pere dani pere neus **out:** true

```
1 bool found = false;
2 string name;
3 while (not found and cin >> name)
4     found = (name == "pere");
5     // equivalently: if (name == "pere") found = true;
6 cout << found << endl;
```

Search without a boolean flag

A found variable is not strictly necessary; it just makes the task easier to read. Alternative version:

```
1 string name = "aaa"; // "fake" but suitable initialization
2 while (name != "pere" and cin >> name) {} // empty loop body
3 cout << (name == "pere") << endl;
```

C++ lazy evaluation

false and q is false and q is **not** evaluated; true or p is true and p is **not** evaluated. This is exactly what stops the loop as soon as "pere" is read (or the input ends).

Working with windows

Idea

A **window** is a small, fixed number of consecutive values from the sequence (or values derived from them) that we keep updated while scanning.

On each iteration:

- 1 Use the current window to decide something.
- 2 **Slide** the window forward by reading the next item.

Example

A width-2 window `<from, to>` keeps the last two cities visited by a plane.

Example: counting flights Madrid → Barcelona

Specification

```
//in:  non-empty sequence of city names (chronological),  
//      terminated by the mark "###"  
//out: number of flights from Madrid to Barcelona
```

Example

Palma Madrid Barcelona Palma Valencia Bilbao Madrid Barcelona ### → 2

```
1  int counter = 0;  
2  string from;  
3  cin >> from;  
4  string to;  
5  cin >> to;  // <from, to> is a width-2 window  
6  while (to != "###") {  
7      if (from == "Madrid" and to == "Barcelona") ++counter;  
8      from = to;  // slide the window  
9      cin >> to;  
10 }
```

Example: was there a Barcelona → Madrid flight?

Specification

```
//in:  non-empty sequence of city names, chronological,  
//     with NO mark and unknown length  
//out: true if the plane ever flew from Barcelona to Madrid
```

Example

Palma Madrid Barcelona Palma Valencia Bilbao Madrid Barcelona → false

```
1 bool found = false;  
2 string from;  
3 cin >> from;  
4 string to;  // <from, to> is a width-2 window  
5 while (not found and cin >> to) {  
6     if (from == "Barcelona" and to == "Madrid") found = true;  
7     from = to;  // slide the window  
8 }  
9 cout << found << endl;
```

A window can also store a computed value

Remark

A window does not have to be a raw past item: it can be **any** value computed from the data already processed (a running sum, a counter, the maximum so far, ...).

Specification

```
//in:  a sequence of integers  $z_1, z_2, z_3, \dots$   
//out:  $\langle \text{true}, z_i \rangle$  for the first  $z_i$  equal to the sum of  
//      the previous ones ( $z_1 + \dots + z_{i-1}$ );  
//       $\langle \text{false}, 0 \rangle$  if no such element exists
```

Example: element equal to the sum of previous ones

```
1 bool found = false;
2 int prev_sum = 0; // sum of already processed items: the window
3 int z;
4 while (not found and cin >> z) {
5     if (z == prev_sum) found = true;
6     prev_sum += z; // slide the window
7 }
8 cout << found << ' ';
9 if (found) cout << z;
10 else      cout << 0;
11 cout << endl;
```

Mental trace

0 3 1 4 : sums so far are 0, 3, 4, 8; when $z = 4$ the running sum is also 4 $\Rightarrow$ <true, 4>.

Sequences of sequences

Specification

in: a sequence of cases; each case is a plane identifier (string), an integer $n \geq 0$, and a sequence of n cities.

out: the identifier of the plane with the most flights from Madrid to Barcelona (ties broken by the smallest identifier).

Example

```
XQ3652 5 Madrid Barcelona Palma Madrid Barcelona
RF2039 6 Barcelona Madrid Barcelona Bilbao Madrid Palma
AD8348 8 Sevilla Ibiza Madrid Barcelona Palma Madrid Barcelona Palma
ZZZZZZ 0
DW3346 4 Madrid Barcelona Madrid Barcelona
```

result: AD8348

Code walkthrough

```
1 int max_freq = -1;
2 string max_identifier = ""; // "fake" but suitable initialization
3 string identifier;
4 while (cin >> identifier) {
5     int n;
6     cin >> n;
7     string from = "FAKE"; // "fake" but suitable initialization
8     int freq = 0;
9     for (int j = 0; j < n; ++j) {
10         string to;
11         cin >> to;
12         if (from == "Madrid" and to == "Barcelona") ++freq;
13         from = to; // slide the window
14     }
15     if (freq > max_freq) {
16         max_freq = freq;
17         max_identifier = identifier;
18     }
19     else if (freq == max_freq and identifier < max_identifier)
20         max_identifier = identifier;
21 }
22 cout << max_identifier << ' ' << max_freq << endl;
```

Code walkthrough: the two nested schemes

Outer loop

Treat-All over the sequence of cases: every case contributes to the final answer.

Inner loop

A **window** <from, to> slides over the cities of the current case, counting Madrid → Barcelona flights.

Practical conclusion

Treat-All, Search, and the window technique combine freely: complex problems are usually a small composition of these simple schemes.

Summary of the algorithmic schemes

Treat-All

- Every item is processed.
- The loop **cannot** stop early.
- Use for: counting, summing, computing a maximum, ...

Search

- Only need to know if some item qualifies.
- The loop **must** stop as soon as found.
- Use for: existence checks, “does X appear?”

Windows

Keep a small piece of state (raw items or a computed value) updated on each iteration, and slide it forward. Works inside either scheme.

Questions?

Sequence Processing: Traversal and Search Algorithmic Schemes