


Eötvös Loránd University
Faculty of Informatics

PROGRAMMING

2019/20 1st semester

Lecture 6

ZS. PLUHÁR, H. TORMA

Content

- Function in condition
- Stacking compound data structures
  - Matrix – vector of vectors
  - Vector of records
- Transforming to set
- The SET type


Function in condition

Function in condition

Task: Let's give a vowel from an English word.

Specification (selection):

Input: $N \in \mathbb{N}$, $Word \in \mathbb{C}h^N$

Output: $Vw \in \mathbb{N}$

Precondition: $N>0$ and $\forall i(1 \le i \le N)$: $IsVowel(Word_i)$

Postcondition: $1 \le Vw \le N$ and $IsVowel(Word_{Vw})$

Definition: $IsVowel: \mathbb{C}h \to \mathbb{L}$,

$IsVowel(L) := \forall i(1 \le i \le 5)$: $L = Vowels_i$,
$Vowels \in \mathbb{C}h^5 = ("a","e","i","o","u")$

Function in condition

Algorithm

VowelInWord

```
Vw:=1
not IsVowel(Word[Vw])
    Vw:=Vw+1
```

IsVowel(L:Character):Boolean

```
i:=1
i≤5 and L≠Vowels[i]
    i:=i+1
IsVowel:=i≤5
```

Matrix

Matrix - pictures

Task: Let's double the size of a raster image by pixel-multiplication!

This means each point will be enlarged to a 2x2 same colored area.

Matrix - pictures

How should we represent an image?

› The picture consists of ordered pixels, so it can be represented with some kind of sequence.

However, it would be hard to assign a single sequence number to each pixel.

› It's easier to define the **row** and the **column** number of a pixel, so why don't we use **double indexing**? The double indexed array is called "**matrix**".

Matrix - pictures

$:= (\mathbb{N}^N)^M$

N - count of rows

M - count of columns

Specification

Input: $N, M \in \mathbb{N}, P_{1..N,1..M} \in \mathbb{N}^{N \times M}$

Output: $DP_{1..2*N,1..2*M} \in \mathbb{N}^{2*N \times 2*M}$

Precondition: –

Postcondition:

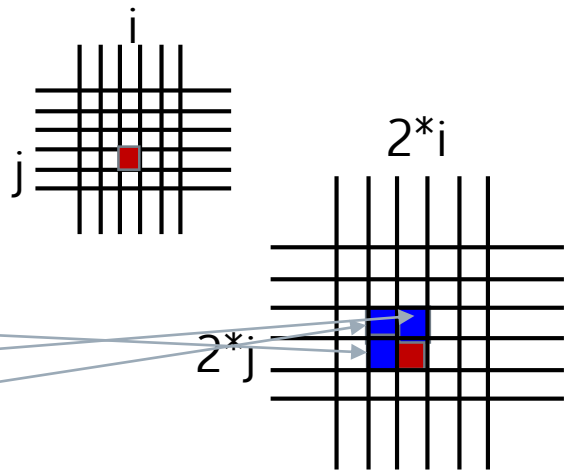
$\forall i (1 \le i \le N): \forall j (1 \le j \le M):$

$DP_{2*i,2*j} = P_{i,j}$ and

$DP_{2*i-1,2*j} = P_{i,j}$ and

$DP_{2*i-1,2*j-1} = P_{i,j}$ and

$DP_{2*i,2*j-1} = P_{i,j}$

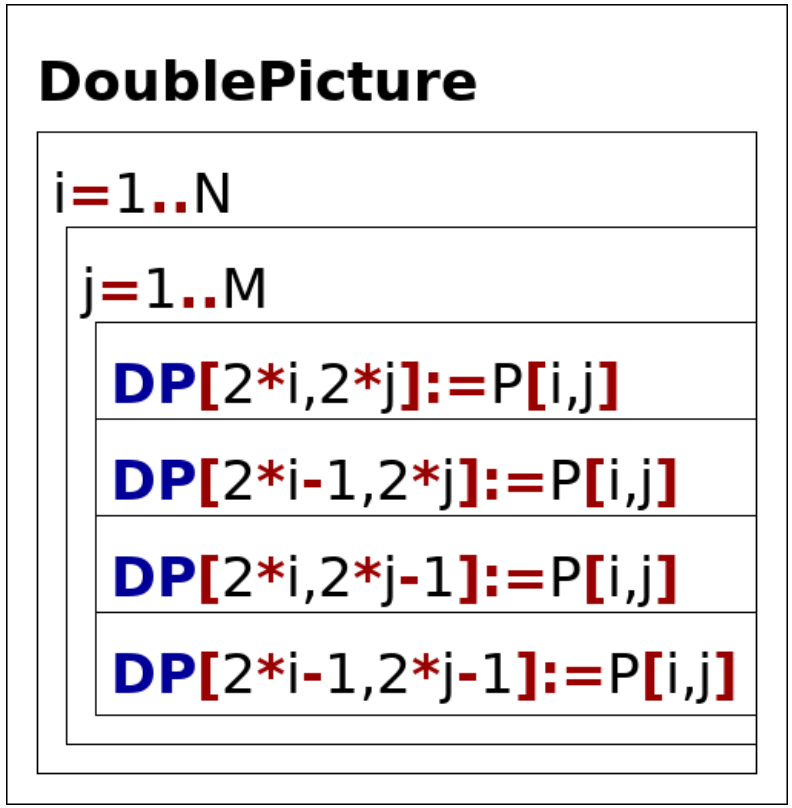


A variation of the **copy** pattern, but from one item, 4 are created.

Matrix - pictures

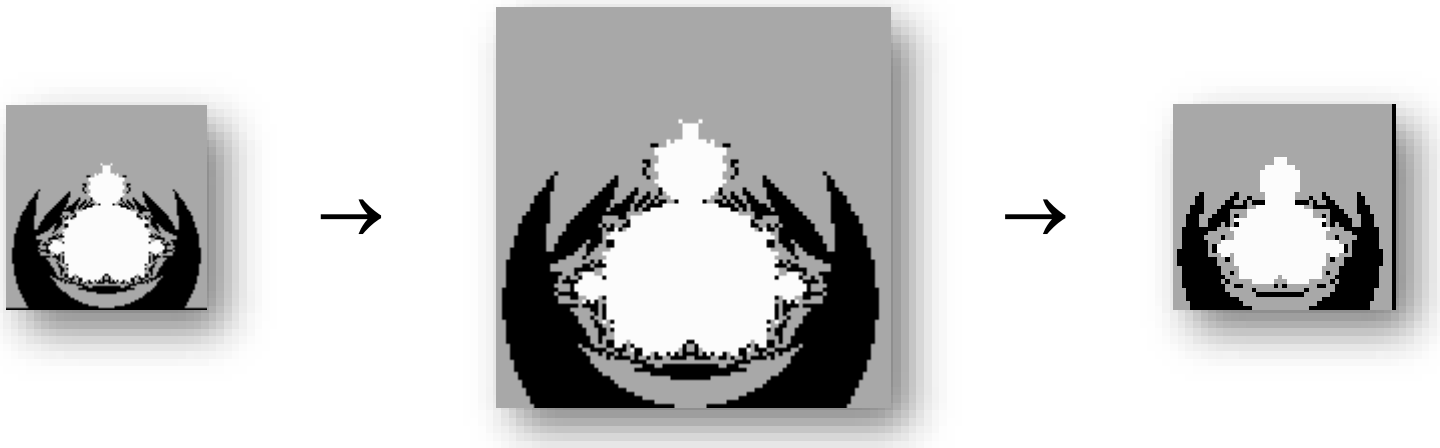
Algorithm

Comment: In different programming languages, different notation may be required to access an element in a matrix
e.g.: C++: `P[i][j]`



Task: Let's halve the size of a raster image by pixel-averaging (to size $N/2 \times M/2$). This means each point of the small image will be the „average” of a 2x2 area

„average”: the average of color codes



Matrix - pictures

Specification

Input: $N, M \in \mathbb{N}, P_{1..N,1..M} \in \mathbb{N}^{N \times M}$

Output: $SP_{1..N/2,1..M/2} \in \mathbb{N}^{N/2 \times M/2}$

Precondition: $\mathrm{isEven}(N),\ \mathrm{isEven}(M)$

Postcondition:

$$\forall i (1 \le i \le N/2):\ \forall j (1 \le j \le M/2):$$
$$SP_{i,j} = (P_{2*i,2*j} + P_{2*i-1,2*j} + P_{2*i,2*j-1} + P_{2*i-1,2*j-1}) / 4$$

Definition: $\mathrm{isEven}: \mathbb{N} \to \mathbb{L}$

$$\mathrm{isEven}(x) := (x \bmod 2) = 0$$

Matrix - pictures

Algorithm

HalvePicture

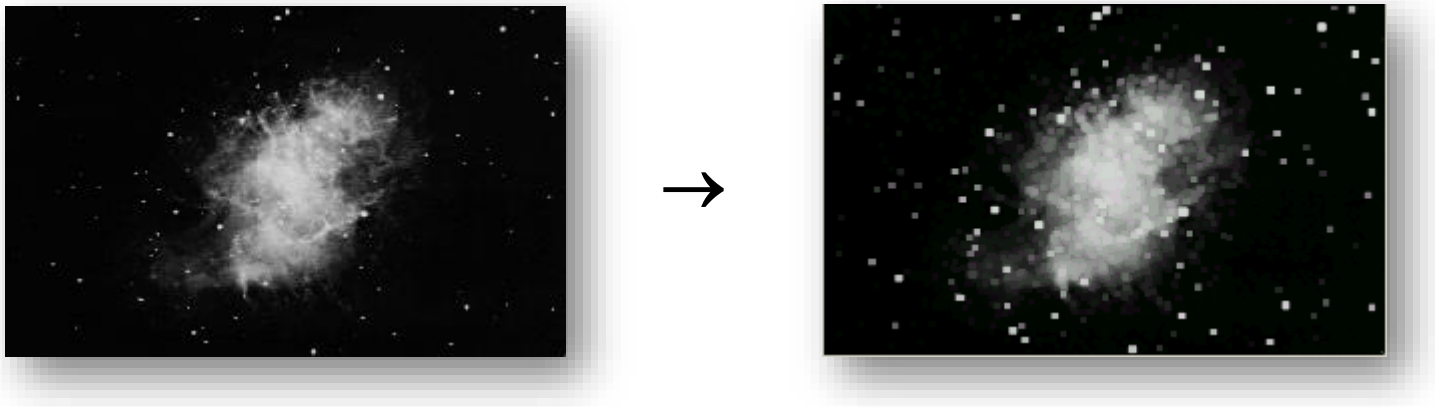
```
i=1..N/2
    j=1..M/2
        SP[i,j]:=(P[2*i,2*j]+P[2*i-1,2*j]
        +P[2*i,2*j-1]+P[2*i-1,2*j-1])/4
```

Comment:

1. With colored images, there might be a problem with averaging. What color is the average of a red and a blue pixel?
2. If using RGB the color is: `Record(red,green,blue:Integer);` and what will be the average?

Matrix - pictures

Task: Let's apply one kind of a Rank-filter to the picture of Crab Nebula (astronomy). This rank filter substitutes each pixel with the maximum of itself and its adjacent 8 pixels.



Matrix - pictures

Specification (copy + maximum selection)

Input: $N, M \in \mathbb{N}, P_{1..N,1..M} \in \mathbb{N}^{N \times M}$

Output: $RP_{1..N,1..M} \in \mathbb{N}^{N \times M}$

Precondition: –

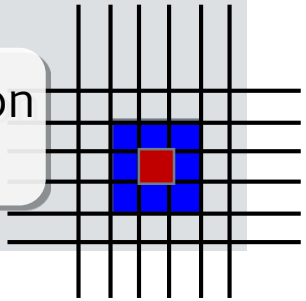
Postcondition: $\forall i (1 < i < N): \forall j (1 < j < M):$

$$RP_{i,j} = \max_{p=i-1}^{i+1} \max_{q=j-1}^{j+1} P_{p,q}$$

and $\forall j (1 \leq j \leq M): RP_{1,j} = P_{1,j}$ and $RP_{N,j} = P_{N,j}$

and $\forall i (1 \leq i \leq N): RP_{i,1} = P_{i,1}$ and $RP_{i,M} = P_{i,M}$

Matrix - pictures



See: maximum selection for value

Algorithm

RankFilter

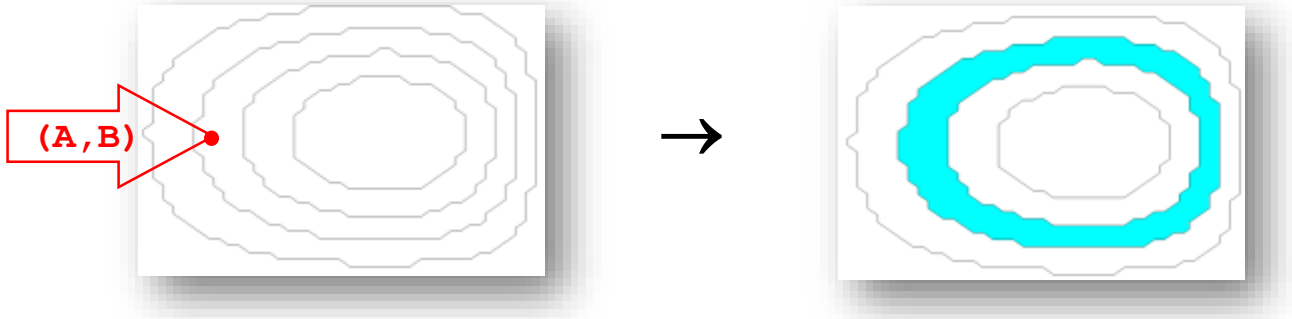
```
i=2..N-1
  j=2..M-1
    Max:=0
    p=i-1..i+1
      q=j-1..j+1
        P[p,q]>Max
        T: Max:=P[p,q]
        F: -
    RP[i,j]:=Max
```

RankFilter(cont...)

```
j=1..M
  RP[1,j]:=P[1,j]
  RP[N,j]:=P[N,j]
i=1..N
  RP[i,1]:=P[i,1]
  RP[i,M]:=P[i,M]
```

Matrix - pictures

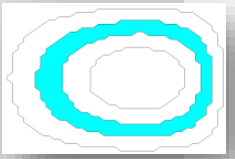
Task: Let's color a range of white pixels of an image to light blue starting from an inner point (A,B).



To be colored: the „inner points", if `Inner(i,j)=True` where

```
Inner: ℕ×ℕ→𝕃
Inner(i,j):=(i=A and j=B or
    White(i,j) and
    Inner(i-1,j) or Inner(i+1,j) or Inner(i,j-1)
    or Inner(i,j+1)) )
```

Matrix - pictures



Specification

Input: $N,M \in \mathbb{N}, P_{1..N,1..M} \in \mathbb{N}^{N \times M}$, $A,B \in \mathbb{N}$

Output: $BP_{1..N,1..M} \in \mathbb{N}^{N \times M}$

Precondition: $A \in (1..N)$, $B \in (1..M)$, P_{AB}=white

Postcondition: $\forall i (1 \leq i \leq N): \forall j (1 \leq j \leq M):$

$Inner(i,j) \rightarrow BP_{i,j}$=lightblue and

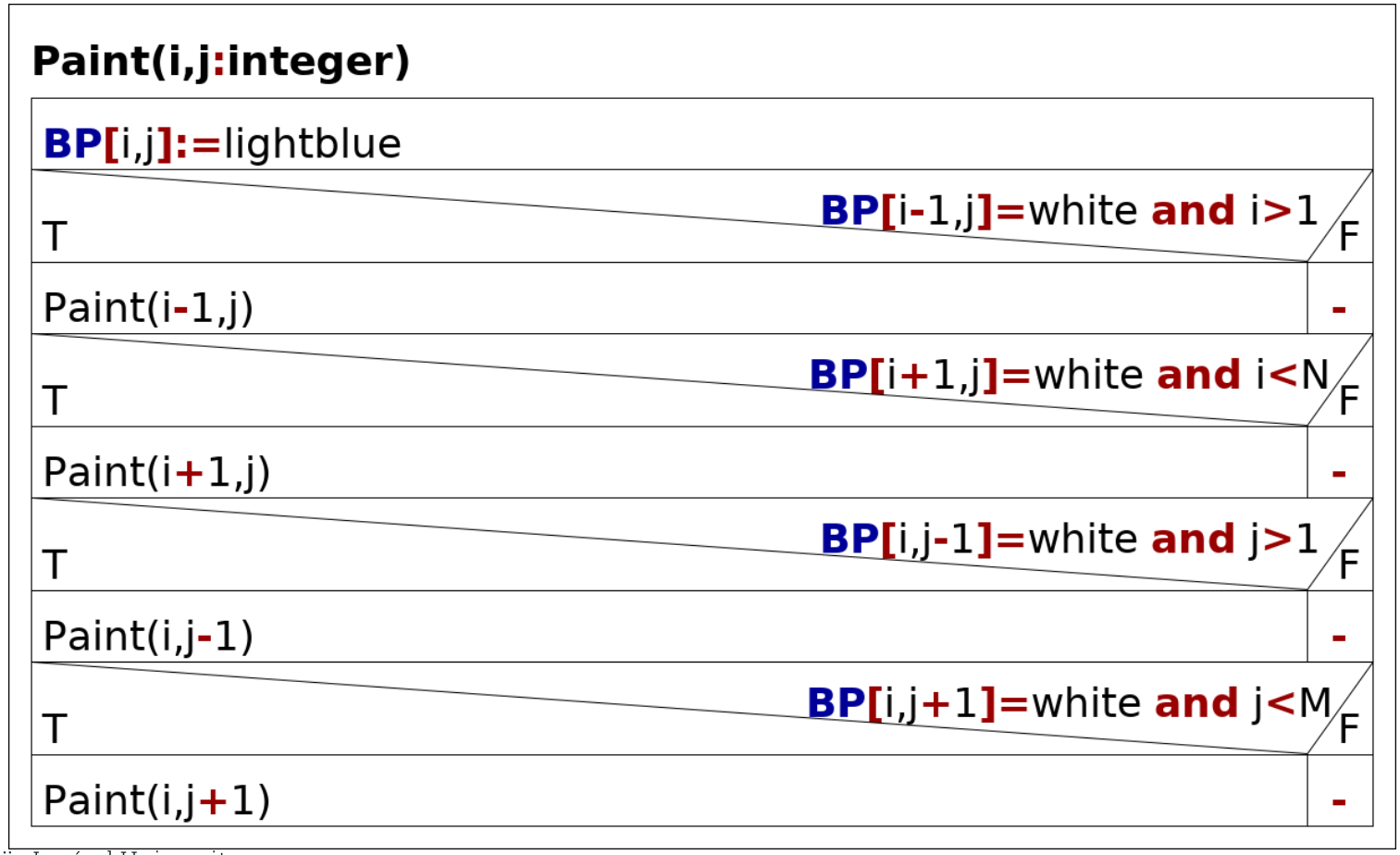
not $Inner(i,j) \rightarrow BP_{i,j}=P_{i,j}$

BP:=P
Paint(A,B)

The `BP` matrix is global – in the function `Paint` is aviable

Matrix - pictures

Algorithm



Vector (1D array) of records

Vector of records - time

Task: There were N earthquakes on a given day. We know the exact times of each earthquake, in their original order in time. Let's give how much was the duration between earthquakes.

Towards the solution:

› First, let's define time

› We can define time by three values (hour, min, sec):

$$Time = H \times M \times S, \quad H, M, S = \mathbb{N}$$

› Algorithmic pattern: **copy**

Vector of records - time

Specification

Input: $N \in \mathbb{N}, E_{1..N} \in Time^N$,

$Time = H \times M \times S$, $H, M, S = \mathbb{N}$

Output: $T_{1..N-1} \in \mathbb{N}^{N-1}$

Precondition: $\forall i (1 \le i \le N): 0 \le E_i.h \le 23$ and $0 \le E_i.m \le 59$ and $0 \le E_i.s \le 59$ and

$\forall i (1 \le i < N): E_i < E_{i+1}$

Postcondition: $\forall i (1 \le i \le N-1): T_i = E_{i+1} - E_i$

Definition:

$-: Time \times Time \to \mathbb{N}$

$i1 - i2 := \ldots ? \ldots$

$<: Time \times Time \to \mathbb{L}$

$i1 < i2 := \ldots ? \ldots$

Vector of records - time

Difference of times

1. **idea** for solution:

We could regard it as the difference of a **3-digit** number where the **base** of each digit is **different**. Then we convert them to seconds.

2. **idea** for solution:

We express times in seconds, and then we could calculate the difference of two whole numbers.

```
inseconds(i):=i.h*3600+i.m*60+i.s
```

How big integer do we need? (24*3600=86 400) What should be the type? (>2 byte)

Vector of records - time

Specification

Input: $N \in \mathbb{N}, E_{1..N} \in Time^N$,

$Time = H \times M \times S,\ H, M, S = \mathbb{N}$

Output: $T_{1..N-1} \in \mathbb{N}^{N-1}$

Precondition: $\forall i (1 \leq i \leq N): 0 \leq E_i.h \leq 23$ and $0 \leq E_i.m \leq 59$ and $0 \leq E_i.s \leq 59$ and

$\forall i (1 \leq i < N): E_i < E_i + 1$

Postcondition: $\forall i (1 \leq i \leq N-1): T_i = E_{i+1} - E_i$

Definition:

$-: Time \times Time \rightarrow \mathbb{N}$

```
i1-i2:=(i1.h*3600+i1.m*60+i1.s) -
       (i2.h*3600+i2.m*60+i2.s)
```

Vector of records - time

Algorithm

Postcondition: $\forall i\,(1 \le i \le N-1):\ T_i = E_{i+1} - E_i$

Definition:

```
- :Time×Time→ℕ
   i1 - i2 := i1.h*3600+i1.m*60+i1.s -
     (i2.h*3600+i2.m*60+i2.s)
```

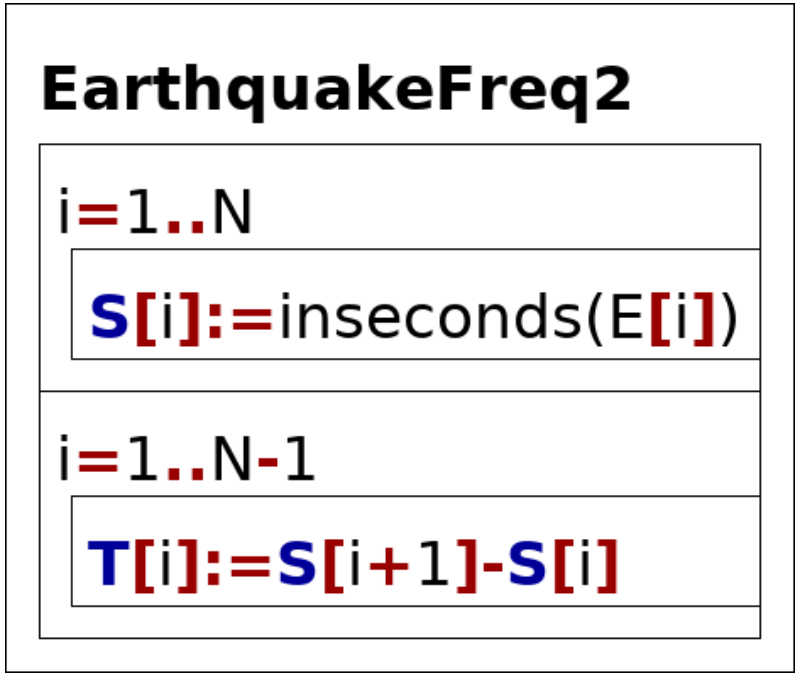
EarthquakeFreq

```
i=1..N
    S[i]:=E[i].h*3600+E[i].m*60+E[i].s
i=1..N-1
    T[i]:=S[i+1]-S[i]
```

- We use an S vector as a helper (auxiliary) object.
- The operator „–” for Times is put indirectly to the algorithm through S.

Vector for records - time

Algoritmh2:

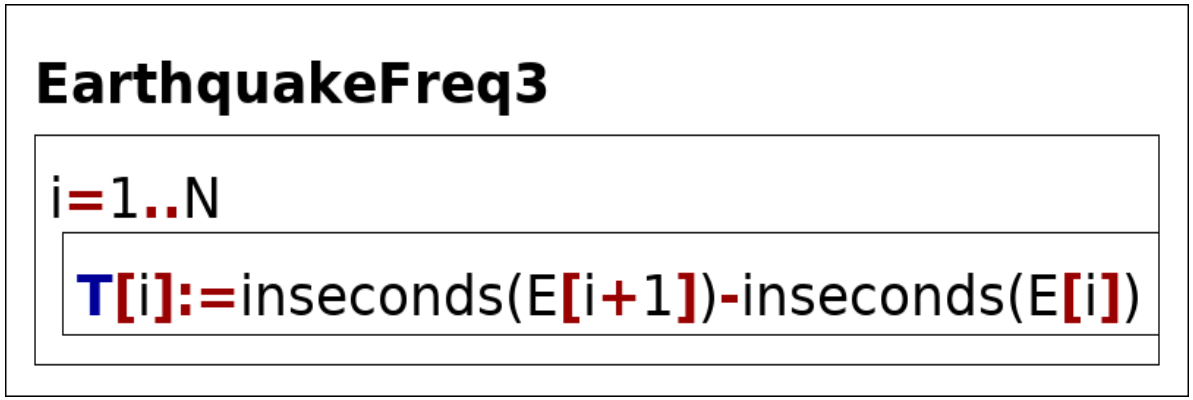


Comments:

- The `inseconds` function has to be implemented!
- If the difference is needed in (hour, minute, second) format, a back transformation is needed.

Vector for records - time

Algoritmh3:

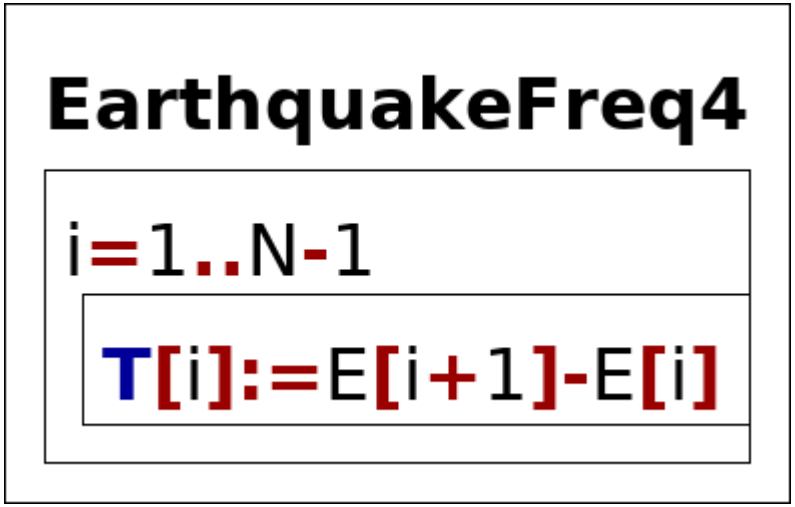


Comment:

By using the `inseconds` function, you can eliminate the S auxiliary vector and the loop for creating the S vector. However, we convert almost all `E[i]` to seconds twice.

Vector for records - time

Algoritmh4:



Comment:

The operator „–” must be defined, in which you could use the inseconds function or its body.

Sequence to set transformation

Sequence → set transformation

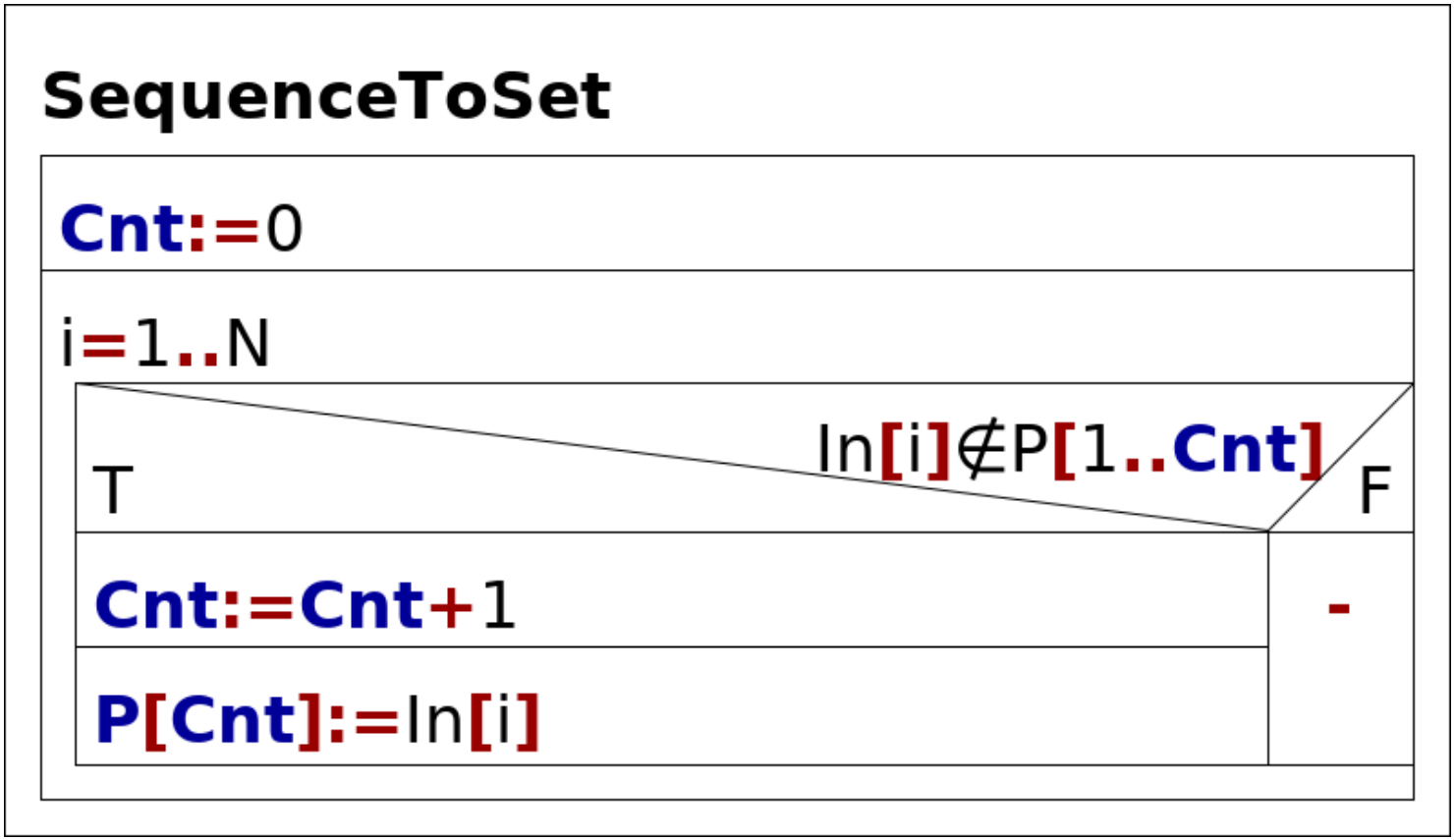
In some tasks, like the ones using the intersection and union pattern of algorithm, the starting data are in a **set**. If we get a sequence as input, we might need to make a set from it.

Example: We know about N shoppings which products customers bought (`In[1..N]`). List the products bought by customers (`P[1..Cnt]`).

Solution: multiple item selection programming pattern – select all the items from the input that have not been put in the result set of the selection yet (decision).

Sequence → set transformation

Algorithm



The SET Type

The SET Type

Value set

- The iteration of the base set (that is defined by the element type) – *„which items can be in the set?”*
- The **element type** is usually a finite, discrete type, sometimes even the count of elements is limited (<256)
- If it does not exist in the language, then the implementation might allow more elements

The SET Type

Operations (mathematical)

- Intersection: `A∩B`
- Union: `A∪B`
- Difference: `A\B`
- Complement: `A'` (not always implementable)
- Element of set: `a∈A`
- Subset: `A⊆B, A⊂B`

The SET Type

Operations (implementation)

- ToSet (puts an element into the set): `S:=S∪{e}`
- FromSet (discards an element from the set): `S:=S\{e}`
- Read (reading in the set)
- Write (writing out the set)
- Empty (creating an empty set) or Empty'SetType pre-defined constant
- Empty? (boolean function)

The SET Type

Representation

› Listing the elements

› A boolean vector - bitmap

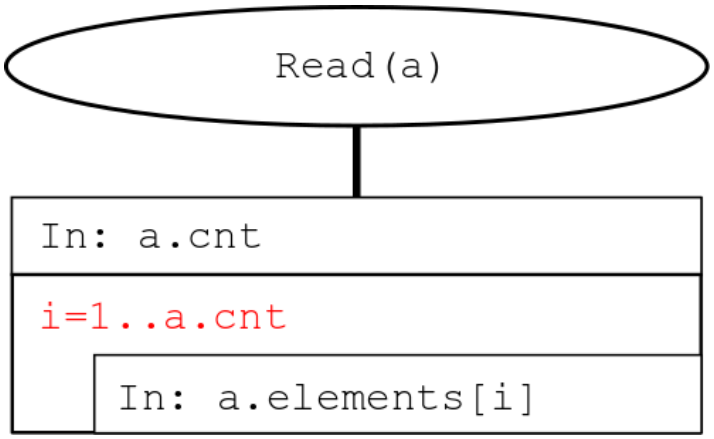
The set type – by listing elements

Representation: by listing the elements

```
Set(ElementType)=
  Record(cnt:integer,
  elements:Array(1..MaxCnt:ElementType))
```

We give the set by listing its elements in an vector whose length is the same as the count of elements of the set (more precisely, in the first Cnt elements).

The set type – by listing elements - READ

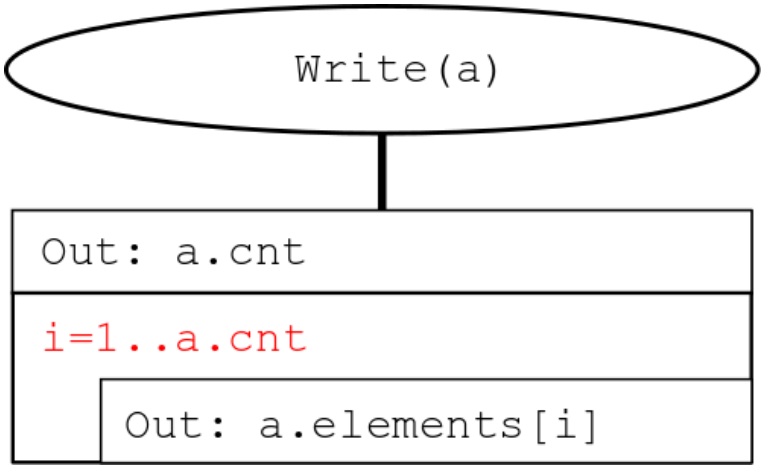


We assume that we read in a **set**.

Calculation of the operation need:

The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.

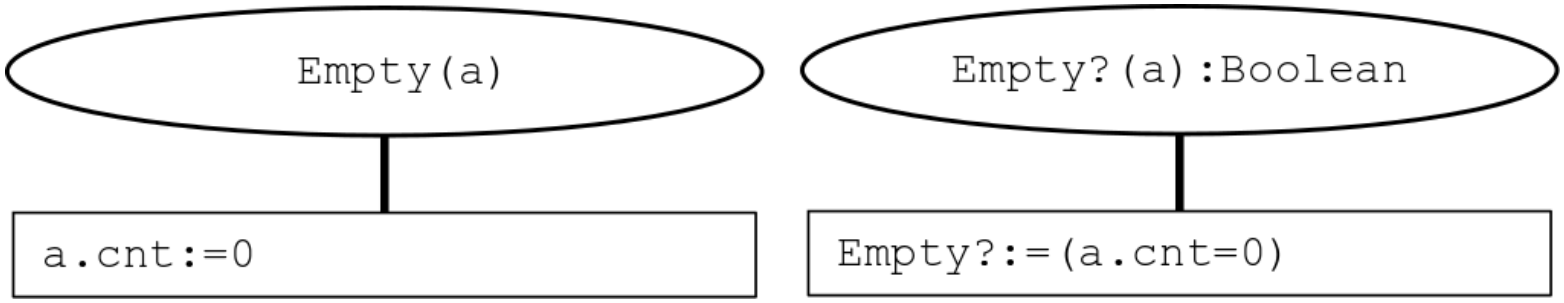
The set type – by listing elements - WRITE



Calculation of the operation need:

The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.

The set type – by listing elements – EMPTY, EMPTY?



Calculation of the operation need:

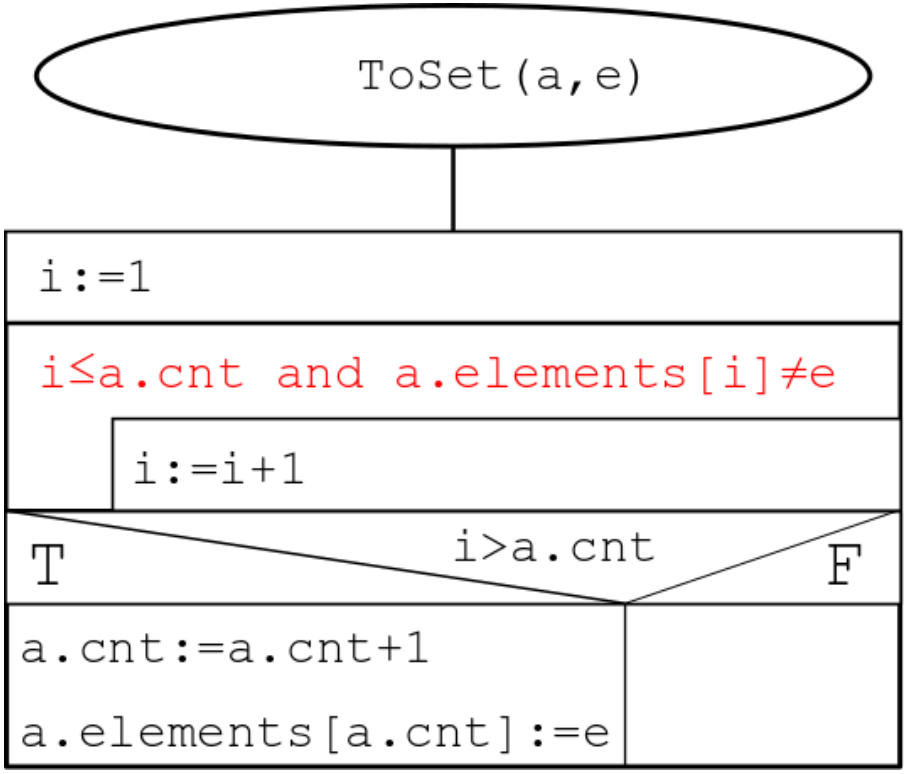
It does not depend on the count of elements of the set.

The set type – by listing elements – ToSet

Applying the
Decision PoA

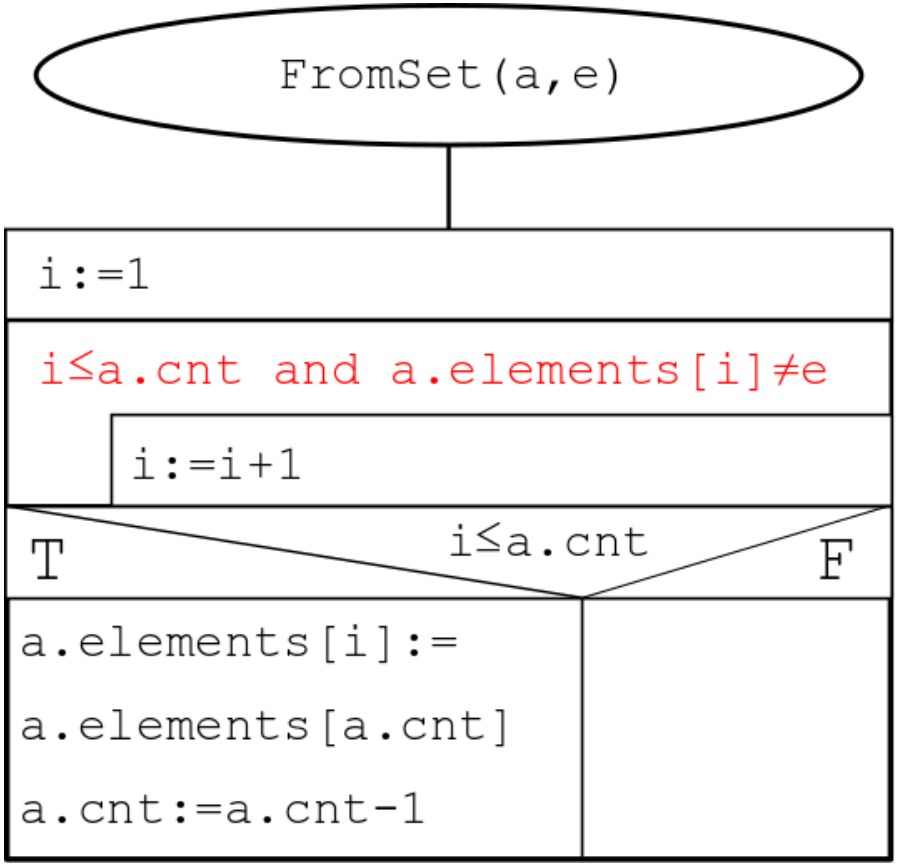
Calculation of the operation need:

The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.



The set type – by listing elements – FromSet

Applying the **Search** PoA

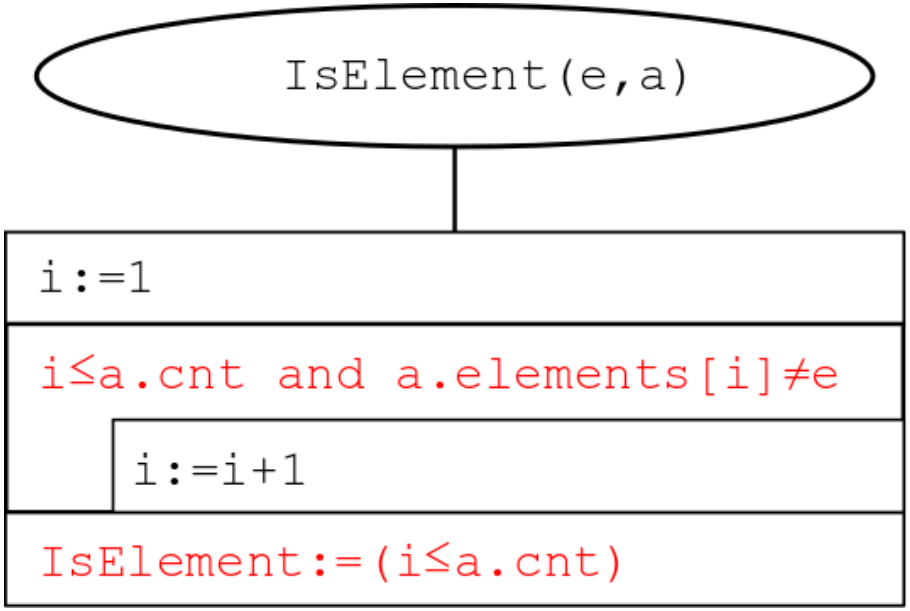


Calculation of the operation need:

The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.

The set type – by listing elements – IsElement

Applying the **Decision** PoA



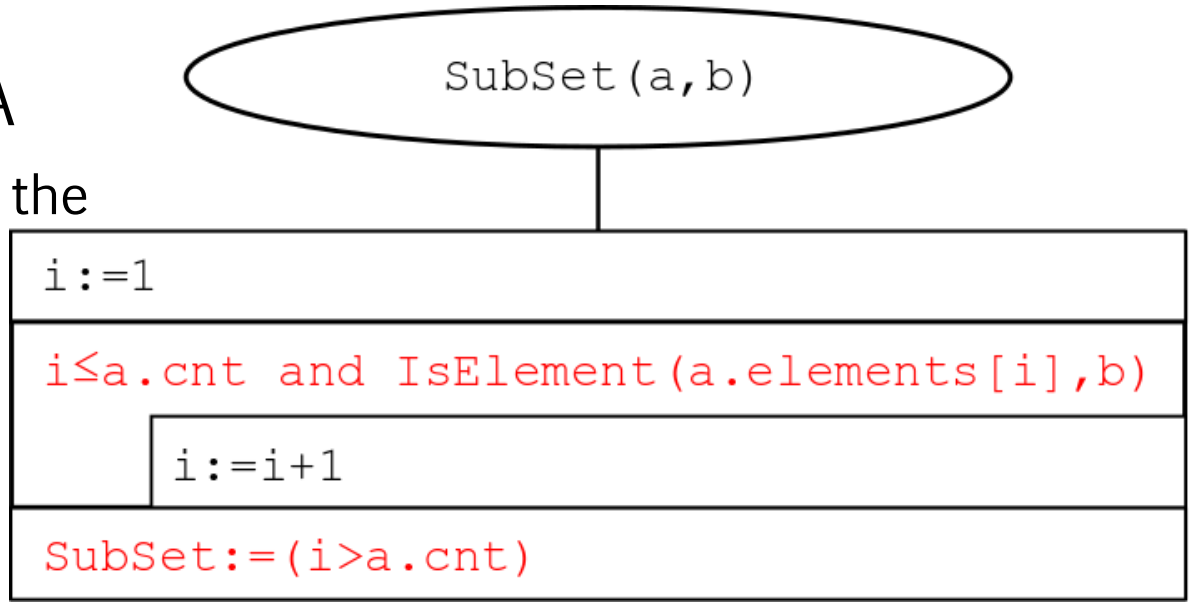
Calculation of the operation need:

The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.

The set type – by listing elements – SubSet

Applying the **Decision** PoA

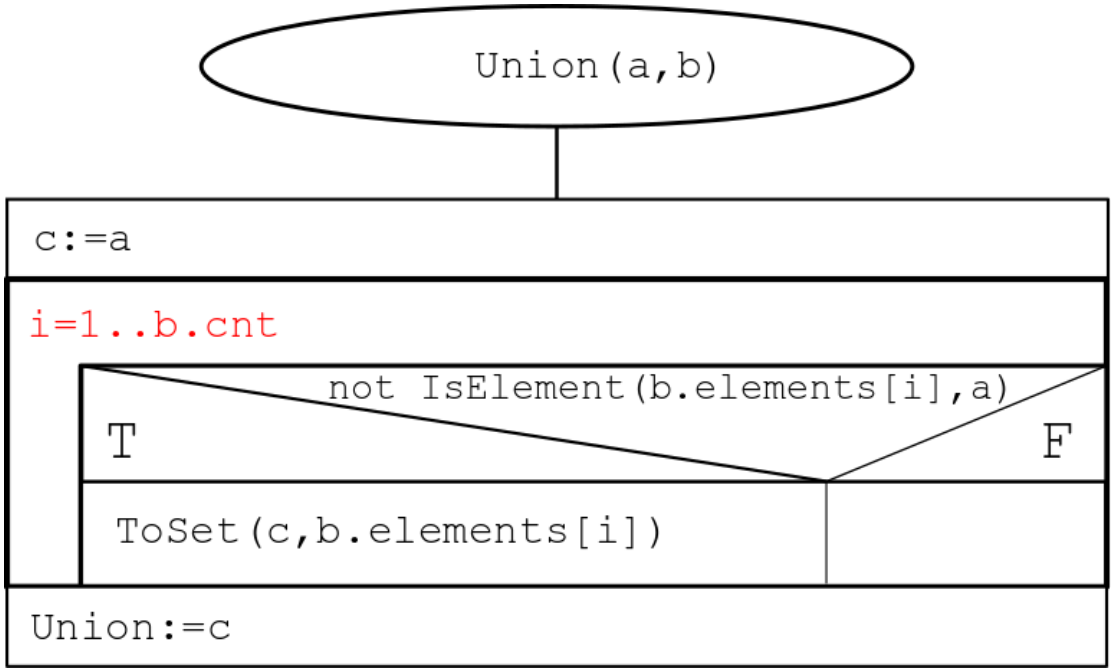
with decision in the conditional statement



Calculation of the operation need:
The loop will run as many times as many elements there are in set A, the IsElement function as many times as many elements there are in set B, thus, the runtime is proportional to the product of the count of elements in the two sets.

The set type – by listing elements – Union

Applying the **Copy, Multiple item selection, Decision** PoA's

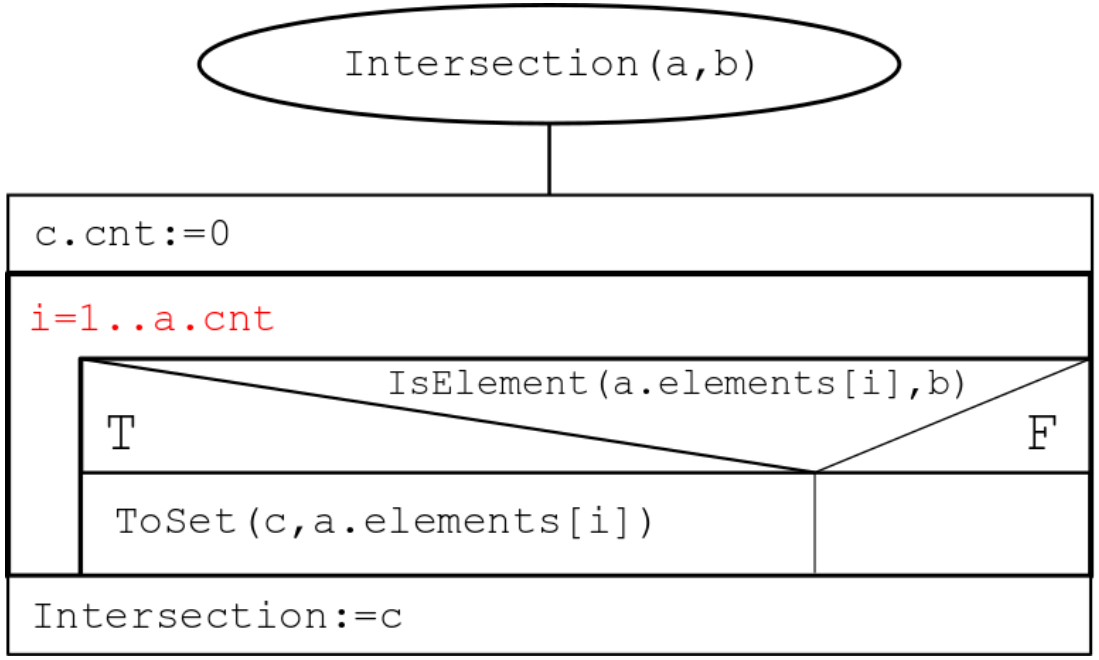


Calculation of the operation need:
The loop will run as many times as many elements there are in set B, the IsElement function as many times as many elements there are in set A, thus, the runtime is proportional to the product of the count of elements in the two sets.

The set type – by listing elements – Intersection

Applying the **Multiple item selection, Decision**

PoA's



Calculation of the operation need:
The loop will run as many times as many elements there are in set A, the IsElement function as many times as many elements there are in set B, thus, the runtime is proportional to the product of the count of elements in the two sets.

The set type – by listing elements

Notes:

- **Problem**: it is not checked if only elements that should be in the set are actually in the set.
- No limit on the type of elements stored in the set, as we can store **anything** in an vector
- **No limit** on the **count of elements** of the base set that the elements of the set derive from. We only limit the count of elements of the specific sets.

The set type – as boolean vector

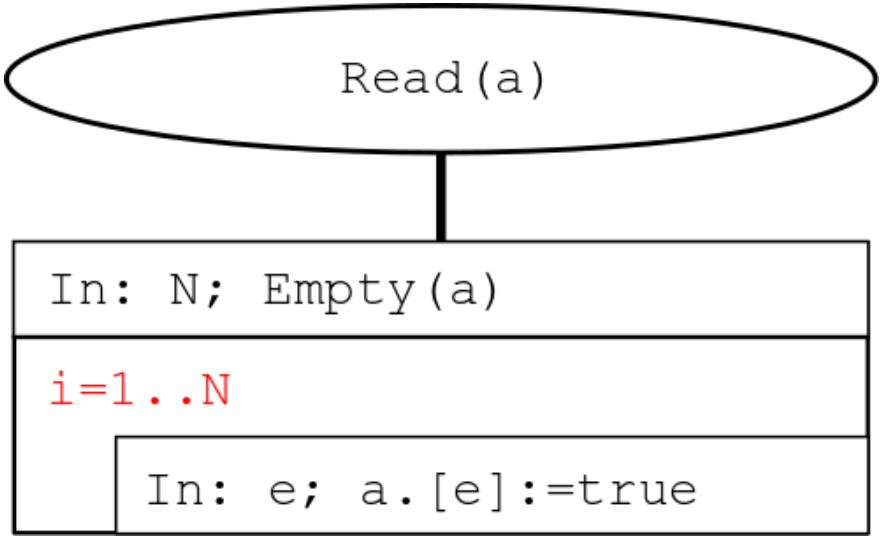
Bit map – boolean vector:

```
Set(ElementType)=Array(Min'ElementType.
.Max'ElementType:Boolean)
```

We interpret the set as a vector of `{true,false}` elements, where we use the value of the element as index.

Such a set is always sorted.

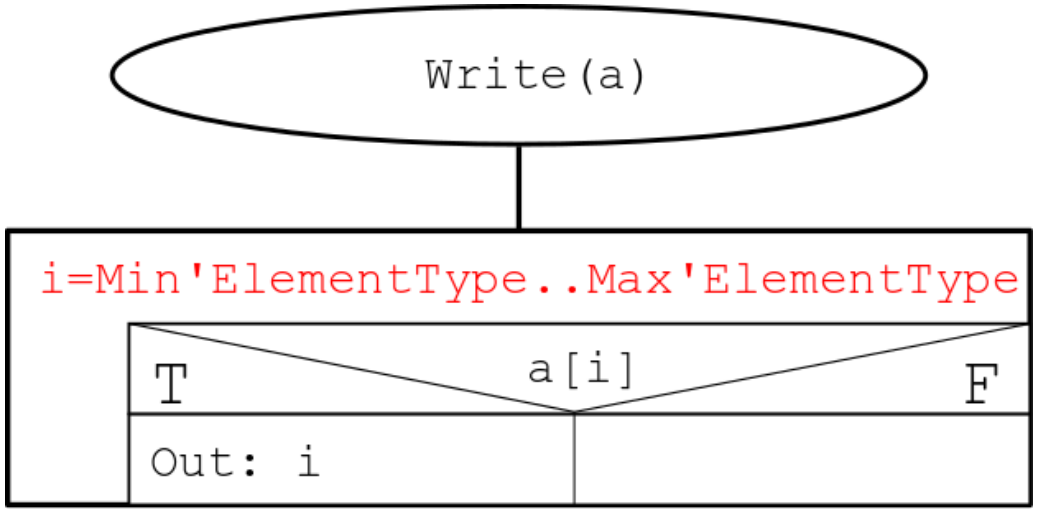
The set type – as boolean vector - READ



Calculation of the operation need:

The operation need of Empty(a) and the loop. The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set

The set type – as boolean vector - WRITE

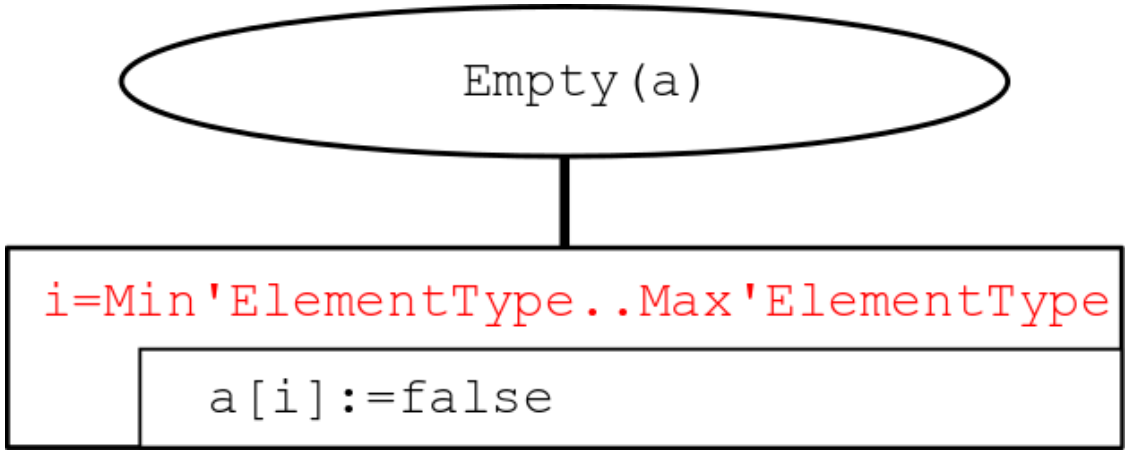


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

What if we stored the maximum and minimum element of the set?

The set type – as boolean vector - EMPTY

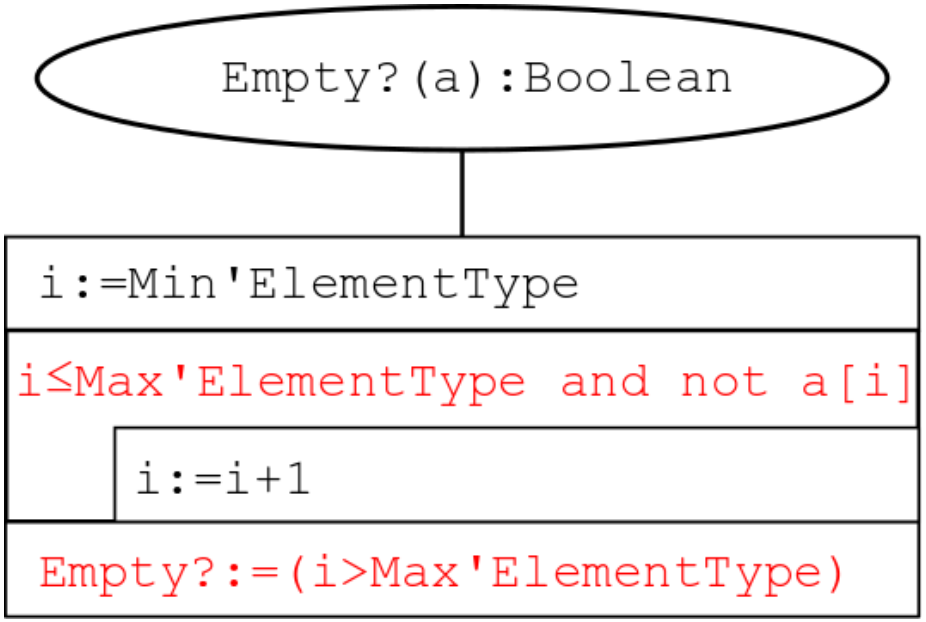


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

The set type – as boolean vector – EMPTY?

Applying the **Decision** PoA



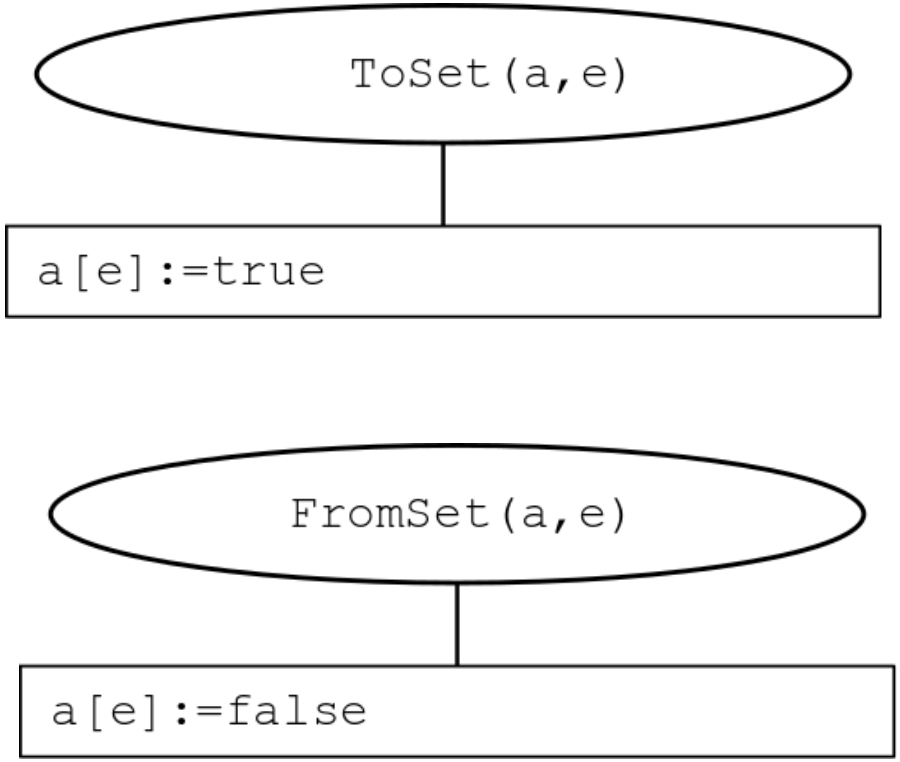
Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

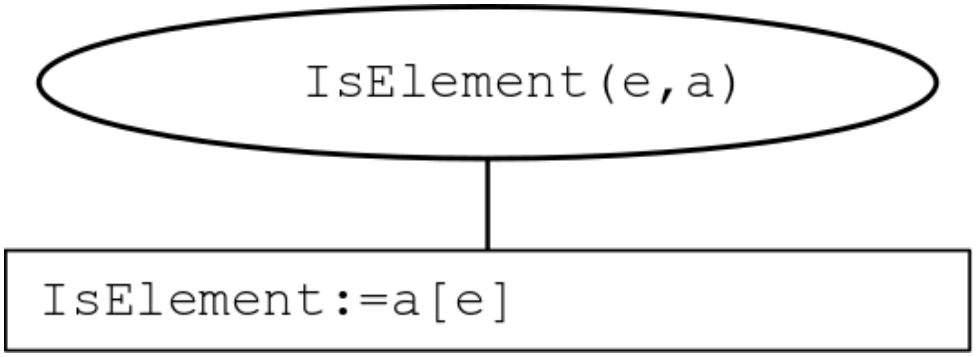
The set type – as boolean vector – ToSet, FromSet

Calculation of the operation need:

It does not depend on the count of elements of the set.



The set type – as boolean vector – IsElement

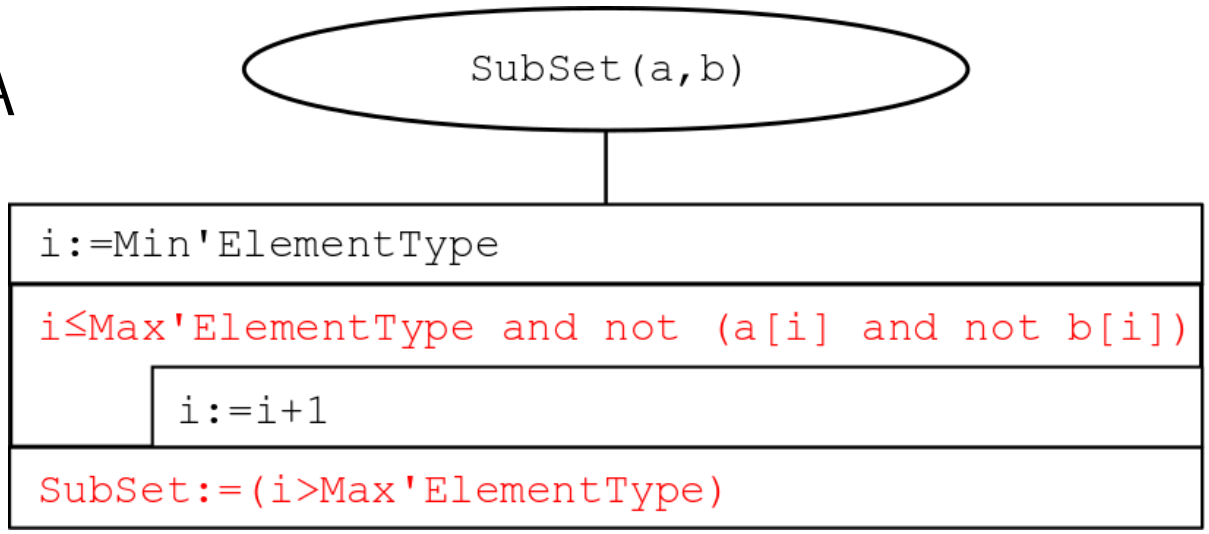


Calculation of the operation need:

It does not depend on the count of elements of the set.

The set type – as boolean vector – SubSet

Applying the **Decision** PoA

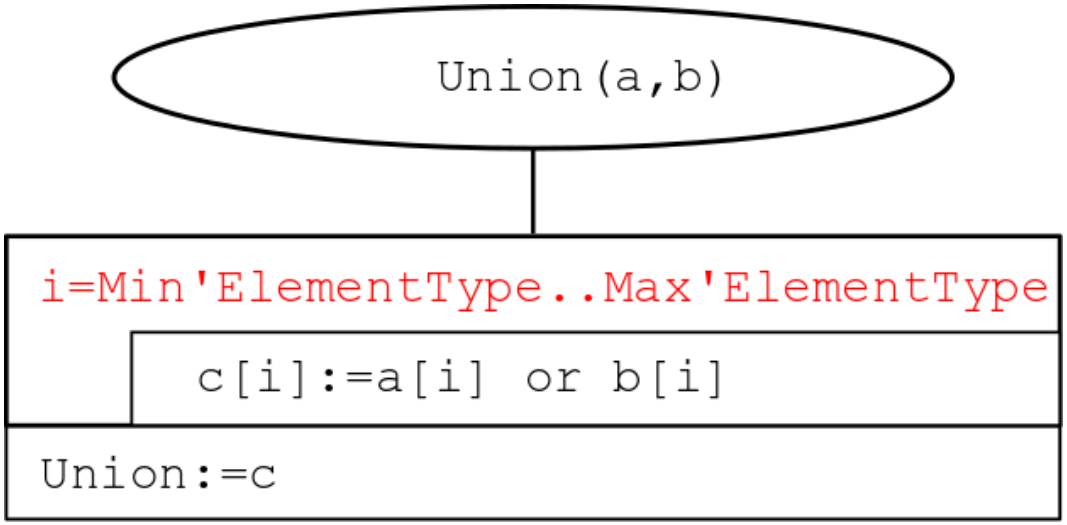


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

The set type – as boolean vector – Union

Applying the **Copy** PoA

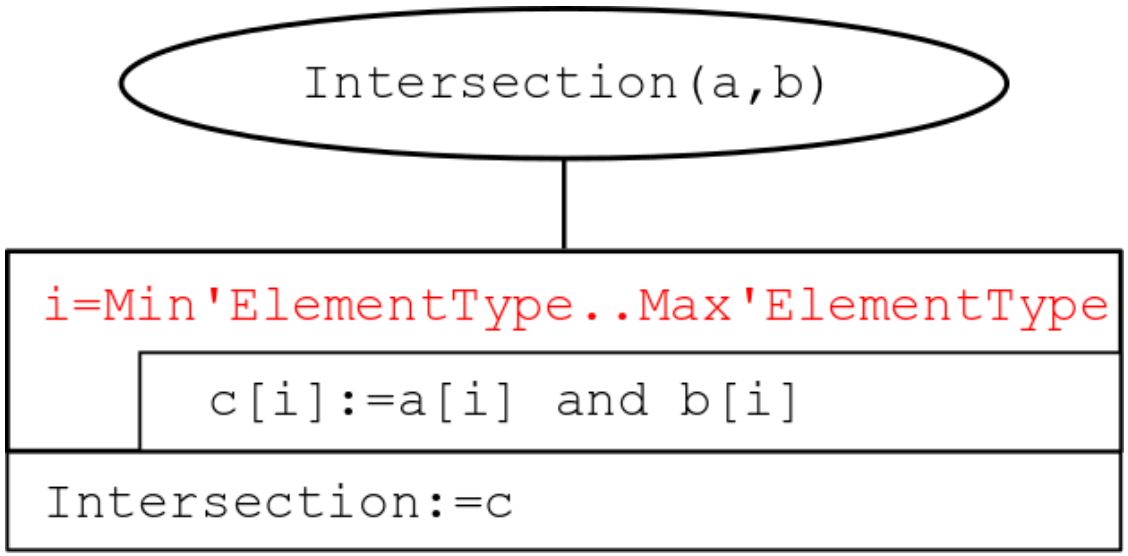


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

The set type – as boolean vector – Intersection

Applying the **Copy** PoA



Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.