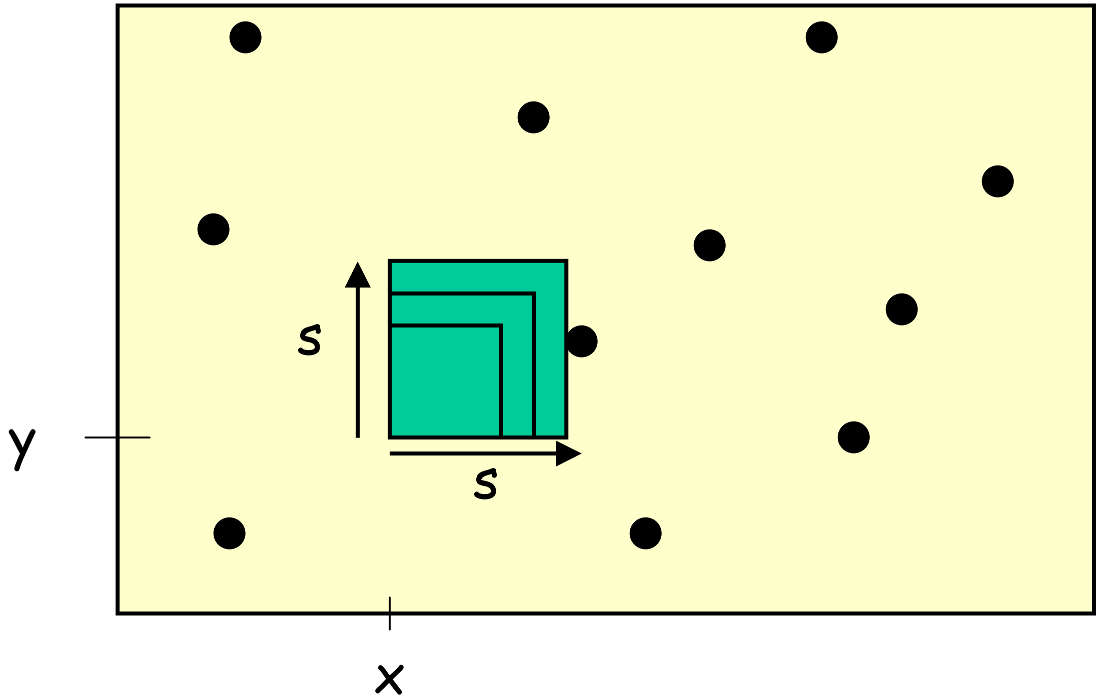
Pippin's Garden Problem

There are many solutions depending how efficient you want to be.

Generate all possible locations (x,y) for the lower left corner, and generate increasing sizes s until something goes wrong:

- The square contains a point
- The square goes outside the outer rectangle

Report the largest square found.



Pippin's Garden Problem

Pseudo code:

```
input width, height and points;
maxSize = 0;                          // saves maximum square size so far
for x = 0 to width-1 {
    for y = 0 to height-1 {           // (x,y) = lower left corner of square
        okay = true;
        s = 0;                        // holds size of the square
        while (okay) {                // while square is still valid
            s = s+1;                  // increment square size
            if ((x+s > width) or (y+s > height)) okay = false;
            for i = 0 to numberOfPoints-1
                if (point[i] is contained within (x,y)..(x+s,y+s))
                    okay = false;
        }
        if (s > maxSize) { maxSize = s;  save (x,y,s); }
    }
}
output saved square: (x, x+s, y, y+s)
```

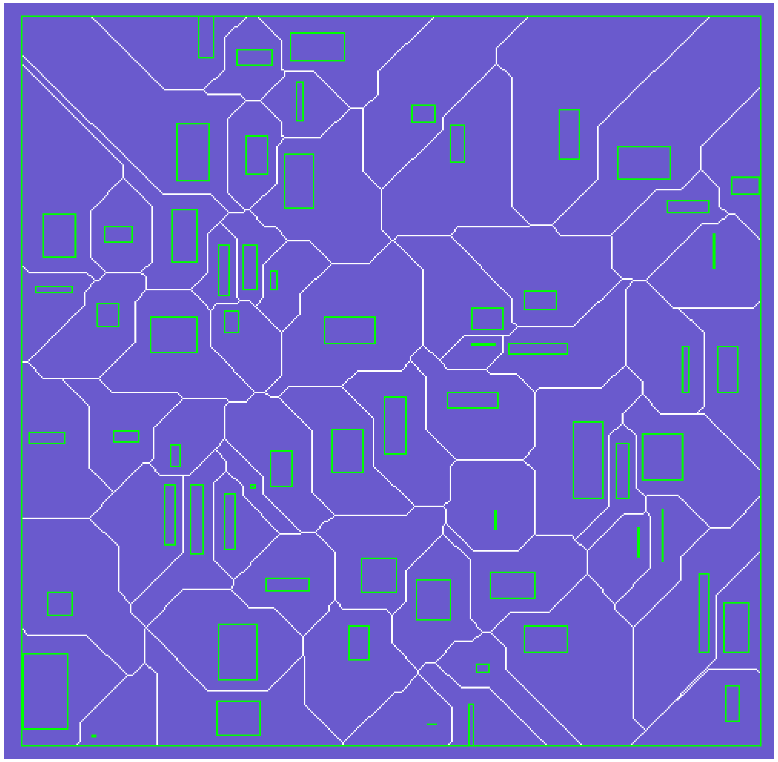
Pippin's Garden Problem

Enhancements:

Early loop termination: Exit the while loop as soon as okay = false;

Limit x and y values: Observe that the choices for x and y can be restricted to the (distinct) x- and y-coordinates of the input points (and 0).

Limit s values: Rather than testing the size s by a linear search, use a binary search instead.



Implementing all these enhancements leads to an $O(n^2 \log n)$ time solution, where n is the number of points. There is an $O(n \log n)$ solution based on Voronoi diagrams. But this is quite hard.

The Game of Rings

Rules: Three piles of rings. Players take turns removing some numbers of stones from one or more piles. Player who takes the last stone(s) wins.

Game State: The state of the game is determined by:

- The numbers of stones in each pile: (i, j, k)

There are at most $100^3 = 1{,}000{,}000$ different states.

Winning Strategy: Since no draws or randomness involved, the result is fully determined from the state. Our encoding:

S[i, j, k] = W if current player can force a win

S[i, j, k] = L otherwise

Solution: Construct the entire S table. Given the initial numbers of stones (a,b,c), if S[a,b,c] = W then Bilbo wins else Frodo wins.

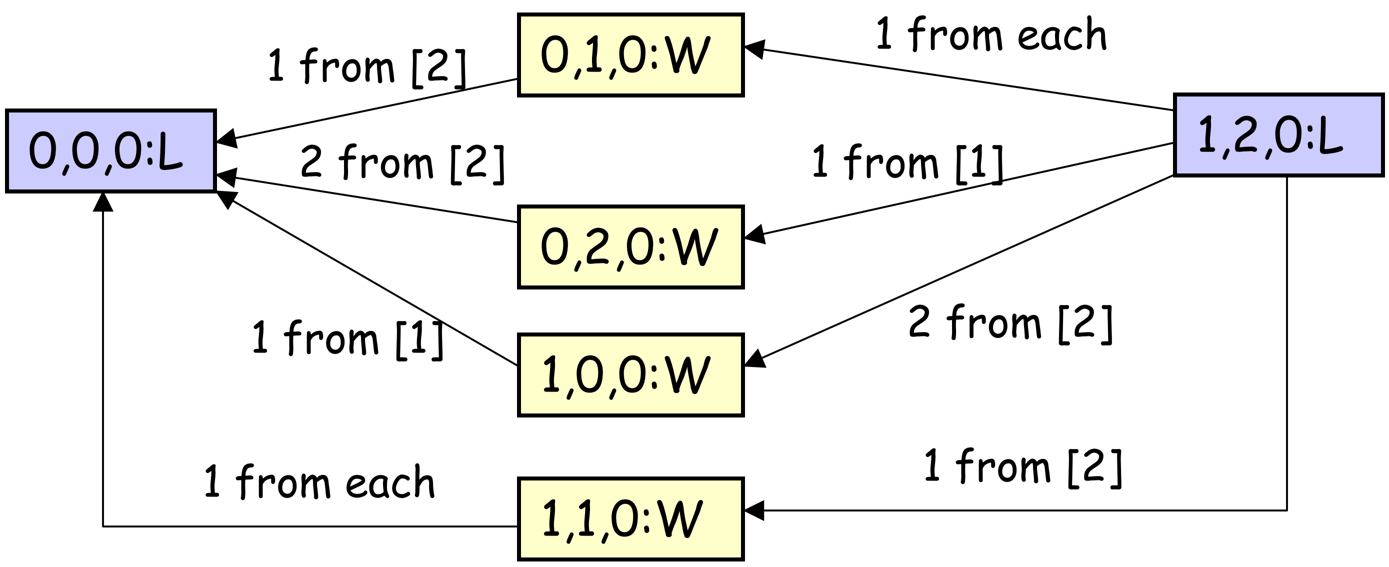
The Game of Rings

Examples:

S[0,0,0] = L (current player loses when stones are gone)
S[1,0,0] = W (by removing last stone from pile 1 we win)
S[0,1,0] = W (by removing last stone from pile 2 we win)
S[0,2,0] = W (by removing last 2 stones from pile 2 we win)
S[1,2,0] = L (every move leads to W state for opponent)

State Transition:

- If any move leads to an "L", then this state is "W"
- If all moves lead to "W", then this state is "L"



The Game of Rings

Pseudo code:

```
input number of stones per pile → (a, b, c)
for i = 0 to a
        for j = 0 to b
            for k = 0 to c
                if (i == j == k == 0)                               S[i, j, k] = L
                else if (S[i-1, j, k] == L)                         S[i, j, k] = W
                else if (S[i, j-1, k] == L)                         S[i, j, k] = W
                else if (S[i, j-2, k] == L)                         S[i, j, k] = W
                else if (S[i, j, k-1] == L)                         S[i, j, k] = W
                else if (S[i, j, k-2] == L)                         S[i, j, k] = W
                else if (S[i, j, k-3] == L)                         S[i, j, k] = W
                else if (S[max(0,i-1), max(0, j-1), max(0,k-1)] == L)
                                                                    S[i, j, k] = W
                else                                                S[i, j, k] = L
if (S[a,b,c] == W) output "Bilbo wins"
else               output "Frodo wins"
```

Design a "smart" subscripting operator that returns "L" if subscripts are negative.