A picture containing text, tiled, net, tile

Description automatically generated



**4)** We have array with N\*N, so ups and downs could be approximated to N

\*-1 if it goes higher;

if(a[i][j]<a[i+L][j+H]) then

energy--;

\*+1 if it goes lower;

if(a[i][j]>a[i+L][j+H]) then

energy++;

\*+0 if it moves horizontally.

if(a[i][j]==a[i+L][j+H]) then

//does nothing

Solution for case on image: 0 -1(/) +6(\) -6(/) +10(\) -7(/) +**12(\)** -11(/) +5(\) -11(/) +1(\) => -2

Energy value for each peak: 0; -1; +5; -1; +9 ; +2; +14; +3; +8; -3; -2.

Here it is clear that **goat gains the highest energy on decline after third peak.**

**6)** We have array with N\*N, so ups and downs could be approximated to N

\*For each step, the camel spends 1 unit of water;

water--;

\*if at the same time it rises higher, then it spends another 1 unit of water (two times more water);

if(a[i][j]<a[i+L][j+H]);

water -=2;

\*At the points of the local minimum, it is filled with 10 units of water;

isMinimum = false

if(a[i][j]<a[i+L][j+H]) then

isMinimum = true;

water += 10;

Solution for case on image: 0 -2(/)-6(\)+10(min)-12(/)-10(\)+10(min)-14(/)-24(\)+10(min)-22(/)-5(\)+10(min)-22(/)-1(\)+10(min) =>-68

Water value for each peak: 0; -2; 2; -10; -10; -24; -38; -60; -55; -77; -68; The most beneficial is **last decline**.

A picture containing text, tiled, net, tile

Description automatically generated



**1)** Volume of lake in our case depends how deep it is and its length, bc world is flat and two dimensional. So we can consider Volume as an Area approximation: V≈Length\*Height/2

It is already obvious that the largest lakes are either between 2 - 3 OR 3 – 4 peaks, let’s denote them as V1 and V2 respectively.

V1 = 16\*22/2 ≈176 V2= 19\*20/2 ≈190 Hence, the largest lake is **between 3 – 4 peaks**. Pseudocode follows:

Class LocalMax{ int Height; int Longtitude; boolean isMax;}

Class LocalMin{ int Height; int Longtitude; boolean isMin;}

fori:

if(arr[i]>arr[i-1])

counter++;

if(counter > avgLongtitudeValue)

LocalMax.isMax = true;

if(arr[i]<arr[i+1])

LocalMax.Height = arr[i];

LocalMax.Longtitude = i;

else continue;

//same for min, but reversed

fori:

height[i] = LocalMin[i+1].Height – (LocalMax[i].Height – LocalMax[i+1].Height);

longtitude[i] = LocalMax[i+1].Longtitude – LocalMax[i].Longtitude;

V[i] = height[i]\*longtitude[i] / 2;

maxVolume = V[0];

indexOfMax = 0;

fori:

maxVolume = compare.(maxVolume, V[i]);

indexOfMax = i;

//so by the end of loop maximum Volume and index is found

print: Biggest lake is between i’th and (i+1)’th LocalMax values

A picture containing text, tiled, net, tile

Description automatically generated



2) As in previous task Volume is an Area approximation: V≈Length\*Height/2

Biggest rock is visually the second one: V3 = 22\*16/2 ≈176

By counting manually we prove that **third rock** is the most profitable. Pseudocode follows:

Class LocalMax{ int Height; int Longtitude; boolean isMax;}

Class LocalMin{ int Height; int Longtitude; boolean isMin;}

Class Point{ int Height; int Longtitude;}

fori:

if(arr[i]>arr[i-1])

counter++;

if(counter > avgLongtitudeValue)

LocalMax.isMax = true;

if(arr[i]<arr[i+1])

LocalMax.Height = arr[i];

LocalMax.Longtitude = i;

else continue;

//same for min, but reversed

fori:

height[i] = (LocalMax[i].Height – LocalMin[i-1].Height);

TempPoint = new Point(

LocalMin[i].Longtitude – LocalMin[i-1].Longtitude,

LocalMin[i-1].Height

)

longtitude[i] = LocalMax[i].Longtitude – LocalMin[i-1].Longtitude + TempPoint.Longtitude;

V[i] = height[i]\*longtitude[i] / 2;

maxVolume = V[0];

indexOfMax = 0;

fori:

maxVolume = compare.(maxVolume, V[i]);

indexOfMax = i;

//so by the end of loop maximum Volume and index is found

print: Biggest rock is on i’th LocalMax value