

This article is written by students. It may include omissions or imperfections, reported as far as possible by our reviewers in the editing notes.

Topic 1: The Marmots

Bianca Moldovan, Alexandra Maria Rus, Pruneanu Andrei, Bob Clara, Ursache Miruna,
Tancel Rianna, Andréas Plantier, William Seiller, Mélisse Zeidler, Lucille Luzi-Nascimento -
11th grade

Year 2023-2024

Schools : Colegiul Național “Emil Racoviță” Cluj-Napoca and Lycée Val de Durance
Teachers : Ariana-Stanca Văcărețu, Hubert Proal
Researcher: Yves Papegay, Inria Sophia Antipolis

Contents:

1. Research topic.....	1
2. The Algorithm	2
2.1 Input and Output.....	2
2.2 C++ Code.....	2
2.3 Methodology.....	3
3. Experimental approach.....	3
3.1 Practical Uses	3
3.2 Experimental Approaches	4
4. Analytical approach.....	4
4.1 Formula.....	4
4.2 Methodology.....	5
5. Conclusion.....	5
6. Bibliography	6
Annex 1 C++ Code	6
Editing Notes	7

1. Research topic

A group of marmots decide to dig a new burrow for the coming winter, but this year they decide to optimize its layout. The problem is that marmots are light sleepers, which means that two rules need to be considered, plus a further one to stop the structure from caving in:

1. From the entrance, or from the end of each tunnel, only a maximum of 2 tunnels can be dug or the structure may collapse.
2. A marmot cannot sleep at the crossroads or in the middle of a tunnel as it would get trodden on by other marmots living further along the borrow and ruin their hibernation. So marmots only sleep at the end of a tunnel which leads nowhere else apart from his room.

The Marmots - MATH.en.JEANS 2023-2024 [Colegiul Național “Emil Racoviță” Cluj-Napoca
& Lycée Val de Durance Pertuis]

3. Even marmots simply moving about, and the sound of their tiny paws generates vibrations which disturb the others while they are sleeping (they really are **light sleepers!!**) so, as we know how many times each one will wake up and go out of the burrow over the winter, we need to make the total sum of their trips as low as possible.

2. The Algorithm

The methodology involved is using Divide and Conquer, Mathematical Induction, and Greedy Methods to devise a C++ code solution. Mathematical formulas were crafted to model the problem, and optimization techniques were applied to find the most efficient solution. The C++ code was implemented, tested, and validated across various scenarios. Performance analysis was conducted to evaluate efficiency and scalability, with optimizations made as necessary. The entire process was documented comprehensively, emphasizing systematic problem-solving and rigorous validation.

2.1 Input and Output


Input:

- `m[]` - is a vector in which we store the number of awakenings M_1, M_2, \dots, M_n for each marmot

Output:

- the number of tunnels
- the sum of awakenings

For example, for the initial case in the problem, our output is:



```
D:\m05cr\matihand\marmote2\bin\Debug\marmote2.exe
Nr. tunnels=8
3 9 8 8 12
Sum of awakenings=40
Process returned 0 (0x0)   execution time : 0.130 s
Press any key to continue.
```

Figure 1. Our output for the initial case

2.2 C++ Code

Our program (in C++), which can be found in [annex 1](#), could quickly determine the *best layout* based on the number of families and how many times each of them wakes up.

2.3 Methodology

A Greedy Algorithm is a fundamental algorithmic strategy used to tackle optimization problems. It involves making a series of choices, each of which looks the best at the moment, with the aim of finding a globally optimal solution. Given that our problem focuses on optimizing the paths traversed by marmots, the Greedy Method is essential for developing an optimal C++ program.

Divide and Conquer (divide et impera) is a way of breaking up a problem into smaller parts and fixing those smaller parts. This method simplifies problem-solving by dividing the problem, conquering each part independently, and combining their solutions. The Divide and Conquer method operates on three key principles: divide, conquer, and combine.

- First, the original problem is divided into smaller subproblems.
- Each subproblem is then solved recursively.
- Finally, the solutions to the subproblems are combined to solve the original problem.

Before employing the Divide and Conquer method, it was necessary to sort the array of marmots in descending order based on the number of awakenings. Only then could we use the method, minimizing the total distance each marmot needs to travel while satisfying the given rules for tunnel connections and marmot room placements.

3. Experimental approach

3.1 Practical Uses

This challenge, which has analogies in a number of real-world applications, is optimising a network layout under certain restrictions. Here are some illustrations of experimental methods and uses:

1. Data Centre Network Architecture:

Issue: Servers in data centres must be networked to maximise data flow while reducing latency and averting bottlenecks.

Usage: The marmot burrow problem can be used to mimic the restrictions on the number of connections and the requirement to prevent key nodes that turn into bottlenecks. A network should be constructed to minimise the overall packet travel time (similar to minimising marmot trips), and servers can only be connected to a limited number of other servers (similar to the 2-tunnel rule).

2. Planning for Public Transportation and Subways:

Issue: Planning a bus or tube system to reduce wait times for passengers and steer clear of congested transfer stops.

Usage: Similar to the maximum of two tunnels, each station can only directly connect to a restricted number of other stations. In order to reduce disruptions and transfer delays, end stations or lines that don't require transfers are preferred—think of them as the sleeping

quarters of marmots. Reducing the total number of transfers and passenger travel time through route optimisation is consistent with minimising marmot excursions.

3. Building Evacuation Plans:

Issue: Planning a building's escape routes so that everyone can leave quickly and safely and without creating traffic jams.

Usage: Just like marmots' sleeping quarters at tunnel ends, exit points (tunnels) from rooms (nodes) should be planned to prevent crowding and guarantee direct pathways to safety. In order to optimise marmot travels, the strategy should minimise the overall travel distance for all inhabitants to evacuate.

3.2 Experimental Approaches

1. Simulation and Modelling:

- Model various layouts and assess their effectiveness using computer simulations.
- Try out different techniques (such simulated annealing and genetic algorithms) to identify the best layouts.

2. Graph Theory:

- Model the burrow (or network) structure using the concepts of graph theory.
- To cut down on overall travel distance, use tree structures, making sure nodes have no more than two offspring and keeping the depth of the tree to a minimum.

3. Optimisation procedures:

- To reduce the total number of trips, apply optimisation procedures such as the shortest path or Travelling Salesman Problem (TSP).
- Constrained satisfaction problems (CSP) can be used to manage the particular regulations pertaining to sleeping arrangements and tunnel connections.

4. Field Experiments:

- To test various layouts and gauge performance, build scaled-down prototypes for real applications, such as small-scale models of buildings or data centres.
- Compile information on real-world usage trends and improve the models using empirical data.

4. Analytical approach

4.1 Formula

To optimize the burrow layout for the marmot families, we must adhere to the following guidelines:

- Connect each marmot's room directly to the entrance with a maximum of 2 tunnels to prevent structural collapse.
- Ensure that each marmot's room is located at the end of a tunnel, with no additional tunnels branching from it.

- Minimize the total distance each marmot must travel to reach the exit.

Therefore, the derived formula, $2n - 2$, where ' n ' signifies the number of marmots, provides a straightforward and practical way to calculate the optimal number of tunnels required.

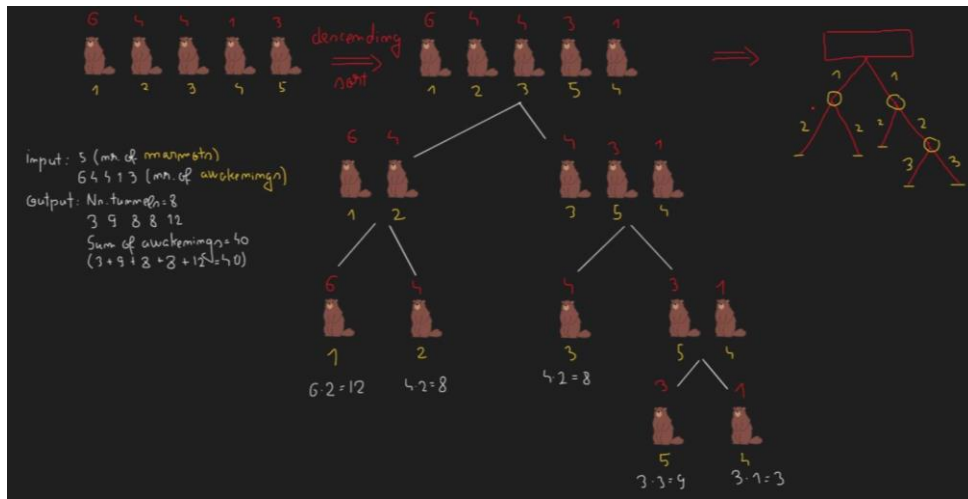


Figure 2. The code explained on the initial example

4.2 Methodology

To demonstrate the formula $x(n)=2n-2$ for the number of tunnels for a family of marmots with n members using mathematical induction, we follow these steps:

Base Case

First, we verify the formula for the initial values.

$$x(1)=2 \cdot 1 - 2 = 0$$

$$x(2)=2 \cdot 2 - 2 = 2$$

$$x(3)=2 \cdot 3 - 2 = 4$$

From the table, we see these values are correct.

Inductive Step

Assume that for some integer k , the formula holds: $x(k)=2k-2$

We need to show that the formula holds for $k+1$: $x(k+1)=x(k)+2$

Using the inductive hypothesis:

$$x(k+1)=(2k-2)+2$$

$$x(k+1)=2k-2+2$$

$$x(k+1)=2k$$

$$x(k+1)=2(k+1)-2$$

Thus, the formula holds for $k+1$.

Conclusion

By mathematical induction, we have shown that $x(n)=2n-2$ is true for all integers $n \geq 1$.

5. Conclusion

Having discussed various solving methods and approaches, we conclude our article on the study of the marmots' behavior. These conclusions should highlight key aspects of the problem and provide insight into how construction optimization can bring significant benefits in terms of efficiency.

6. Bibliography

[1] https://en.wikipedia.org/wiki/Greedy_algorithm

[2] https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm

Annex 1 C++ Code

```
#include <iostream>
#include <fstream>
using namespace std;
ifstream fin("marmote.in");

int n;

struct marmota
{
    int indice, nr_treziri, grad;
} m[101];

void citire()
{
    /// <summary>
    /// input
    /// </summary>
    fin >> n;
    for (int i = 1; i <= n; i++)
    {
        fin >> m[i].nr_treziri;
        m[i].indice = i;
    }
}

void sorting()
{
    ///Sort descending
    ///We sort in descending order according to the number of awakenings
    for (int i = 1; i < n; i++)
```

```

    for (int j = i + 1; j <= n; j++)
        if (m[i].nr_treziri > m[j].nr_treziri)
            swap(m[i], m[j]);
}

//number of tunnels based on inner grade
//If they have the highest possible number of awakenings, they must have the lowest possible
number of tunnels to the exit
int det_numar_canale_grad_int(int st, int dr, int grad)
{
    //divide et impera
    //the grade would be 0
    //nr tunnels 2*n-2
    if(st==dr)
    {
        m[st].grad=grad;
        return 1;
    }

    else
    {
        int m=(dr+st)/2;
        int s1=det_numar_canale_grad_int(st, m, grad+1);
        int s2=det_numar_canale_grad_int(m+1, dr, grad+1);
        return s1+s2+1;
    }
}

void solve()
{
    cout<<"Nr. tunnels="<<det_numar_canale_grad_int(1, n, 0)-1<<endl;
    int sum=0;
    for(int i=1; i<=n; i++)
    {
        cout<<m[i].nr_treziri*m[i].grad<<" ";
        sum+=m[i].nr_treziri*m[i].grad;
    }
    cout<<endl<<"Sum of awakenings="<<sum;
}

int main()
{
    citire();
    sorting();
    solve();
    fin.close();
    return 0;
}

```

}

Editing Notes